V4Design

Visual and textual content re-purposing FOR(4) architecture, Design and virtual reality games - H2020-779962

D6.3

Operational prototypes and user interfaces for architecture and VR game design application

Dissemination level:	Public
Contractual date of delivery:	Month 12, 31 December 2018
Actual date of delivery:	Month 12, 27 December 2018
Work Package:	WP6: System integration and tool development for content re-purposing
Task:	T6.2: Development of VR and 3D game authoring tool T6.3: Tool development for architects and designers T6.4: System integration
Type:	Report
Approval Status:	Final version
Version:	1.0
Number of pages:	78
Filename:	D6.3_V4Design_OperationalPrototypesAndUserInterfaces_20181227.pdf

Abstract

This deliverable presents the UI and UX prototypes, which will be designed for the architecture and video game design applications of the V4Design platform. The document also describes the technical components and infrastructure of the initial Operational Prototype for the V4Design platform. It provides an overview of the demonstration application prototypes, the organisation

and composition of the different modules and the hosting infrastructure. The Operational Prototype will be the scaffolding on which the platform will be built iteratively, adding functionality and depth on top of the dummy-based setup which marks this first milestone

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	31/10/2018	ToC creation and content definition	Ayman Moghnieh
0.2	07/11/2018	Initial feedback integration	Ayman Moghnieh
0.3	29/11/2018	Partner's contributions integration	Ayman Moghnieh, Yash Shekhawat
0.4	13/12/2018	Changes in document's structure	Ayman Moghnieh, Yash Shekhawat
0.5	17/12/2018	1st integrated draft prepared and sent for internal review	Ayman Moghnieh
0.6	20/12/2018	Internal review	Jens Grivolla
0.7	21/12/2018	Incorporation of internal review suggestions	Ayman Moghnieh
1.0	27/12/2018	Preparation of the final draft	Ayman Moghnieh, Konstantinos Avgerinakis

Author list

Organization	Name	Contact Information
McNeel	Ayman Moghnieh	aymanmoghnieh@gmail.com
McNeel	Luis Fraguada	luis@mcneel.com
McNeel	Verena Vogler	verena@mcneel.com
NURO	Yash Shekhawat	yash.shekhawat@nurogames.com
NURO	Boris Irmshcher	boris.irmshcher@nurogames.com
NURO	Sebastian Krauss	Sebastian.krauss@nurogames.com
CERTH	Spyridon Symeonidis	spyridons@iti.gr
CERTH	Elissavet Batziou	batziou.el@iti.gr
CERTH	Konstantinos Avgerinakis	koafgeri@iti.gr
CERTH	George Meditskos	gmeditsk@iti.gr
CERTH	Stefanos Vrochidis	stefanos@iti.gr
UPF	Simon Mille	simon.mille@upf.edu
KUL	Jens Derdaele	jens.derdaele@kuleuven.be
KUL	Maarten Vergauwen	maarten.vergauwen@kuleuven.be

Executive Summary

D6.3 presents a demonstration of the envisioned platform by means of an operational prototype. The prototype shows a rough sketch of the User Interface/User Experience (UI/UX) and dummy implementations of the functionalities of the system. The operational prototype is tested within four use cases: a) Architectural design, related to existing or historical buildings and their environments, b) Architectural design, related to artworks, historic or stylistic elements, c) Design of virtual environments, related to TV series and VR video games and d) Design of virtual environments, related to actual news for VR (re-) living the date. Two of the use cases (a, b) will implement the exterior and interior space using the V4Design architecture authoring tool, while the other two (c, d) will utilize the V4Design video game authoring tool to create the interior and exteriors of a Virtual Reality (VR) video game.

This document provides a brief technical reference for the D6.3 prototype deliverable of the V4Design platform. First, it presents the architecture and the modules involved. Then, the prototype applications testing the two aforementioned use cases are presented. In the next sections, the code organization and the infrastructure are detailed. Finally, this document provides links to live demo of the prototype and to the code repository.

Abbreviations and Acronyms

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CAD	Computer Aided Design
DB	Database
DLL	Dynamic Link Library
GUI	Graphic User Interface
HLURs	High-level User Requirements
JMS	Java Message Service
JSON	JavaScript Object Notation
RDF	Resource Description Framework
SQL	Structured Query Language
TRs	Technical Requirements
UI	User Interface
UIMA	Unstructured Information Management Architecture
URI	Unique Resource Identifier
URs	User Requirements
UX	User Experience

Table of Contents

1 INTRODUCTION

The V4Design project aims to repurpose visual and textual content for use in various industries, including applications in architecture, design, and video game development. The mechanisms which allow the repurposing of the content into 3d models and assets useful to designers in these industries is but one aspect of the project which has been described in previous deliverables (D6.1, D6.2). These mechanisms strive for a system architecture which is robust enough to handle a variety of inputs and generate useful assets. This deliverable will also focus on how these assets are made available to end users through a series of tools which integrate into industry standard applications typically used in design and production workflows for architectural design and video game development.

In D6.1, a general roadmap (Figure 1) and technical vision for the implementation of the V4Design platform was established. The user requirements (URs#) were presented in D7.2, while their correlation with the technical requirements (TRs#) and the technical vision of V4Design platform were introduced in D6.2, where the global architecture of the system and its subsystems, workflows and interfaces were defined.

							MS2								MS3								MS4								MS5
	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
Rhino3D authoring tool							prototype						V2								V3								V4		
Nuro VR authoring tool							prototype						V2								V3								V4		
V4Design REST API							prototype						V1								V2								V3		
KB Population	prototype						V1								V2								V3								V4
Reasoning	prototype						V1								V2								V3								V4
STBOL	prototype										V1										V2								V3		
Aesthetic extraction (AE)	prototype									V1											V2							V3			
Texture Proposals (TP)	prototype									V1											V2							V3			
3D-Reconstruction	prototype														V1								V2					V3			
TALN-LG							prototype				V1																		V2		
TALN-LA							prototype						V1															V2			
Data storage & retrieval	prototype						V1						V2								V3							V4			

Figure 1: Technical roadmap

The purpose of this document is to provide a brief technical reference for the D6.3 deliverable, which is the first technical milestone of the project. D6.3 contains a first rough UI/UX for V4Design platform and dummy implementations of the major services, processes and workflows.

Section 2 introduces the global architecture of the platform and the definition of its main component types, and presents an overview of the operational prototypes, discussing their role, configuration, and development roadmap, showing in each case a sample output generated by the component.

Section 3 contains a description of the demonstrator applications: the architecture integration demonstration, the video-game authoring tool, and the architecture authoring tool.

Section 4 provides a walk-through of the structure of the code, to assist in the navigation of the Subversion repository

Section 5 contains links and details for accessing the demonstrator application for reviewers.

Section 6 presents a brief summary and conclusions.

2 PROTOTYPE ARCHITECTURE

In the following chapter, we describe the global architecture of the V4Design platform, discussing its conceptual designs, components, and integration model.

The global architecture is introduced in section 2.1 starting by its conceptual design, then a generic definition of a V4Design service is discussed to illustrate the standardization of these components, which in turn are grouped in three tiers according to their role and integration in the platform cycle. We discuss how the platform communication model chains these services communication to implement this cycle, and the input/output model to illustrate how data is processed and created.

In section 2.2, we discuss the service prototypes, including concepts, technical requirements and development plans, and show sample output examples to illustrate the added value of each service.

In section 2.3, we discuss the platform middleware components, elaborating on their implementation, configuration, functionalities, and performance.

In section 2.4, we discuss the platform's authoring tools for video games and architecture, including their user profile and user data management policies, and the related model derivatives database.

In section 2.5 we discuss the content extraction pipeline of the platform, by which raw data is extracted from data sources, and used to create the assets envisioned in V4Design.

2.1 Global Architecture

The architecture model chosen for the V4Design platform is that of a distributed processing-oriented architecture. Accordingly, each of the services conceived for the platform can be hosted and managed independently from the others, and services communicate via the platform's message bus and share middleware components such as the data storage and retrieval system, and the platform API component.

Each component of the architecture is a self-contained unit with its own connector, queue, local storage, and processing policy. When it comes online, it notifies the message bus of its availability, and starts receiving messages from other services whose processing results proceed and are required by the component. When notified, the service retrieves the related data from the platform's data storage and retrieval system, processes it, and then stores the results back on the data storage and retrieval.

The platform architecture is designed in a manner that enforces a separation of concerns among its different layers and components. From a high-level perspective, the interactive components, otherwise referred to as the user tools, are segregated from the processing components that are designed to process data efficiently and autonomously. The two sets are connected via the platform's API, which allows communication to be established among the two sets in an orderly fashion. The interactive components are responsible for managing users, servicing their requests, and storing user-related data, while the processing components are responsible for storing raw data and processing it to generated high-end assets of relevance to the designated user profiles. The communication among both sets allows to retrieve data from the platform storage, and to channel occasional user petitions that launch specific processing components explicitly.

In concrete, the platform's front-end side is composed of two tools that are developed in the course of the project: the Rhino3D tool and the authoring tool for virtual reality. These tools communicate with an API designed to channel their data request to the platform back-end. The API also facilitates the sharing of data, or more precisely user data, among the tools. It acts as the platform's back-end interface with the user tools, and consequently can execute data queries and retrieve data directly from the platforms data storage and retrieval system. In addition, the API can send requests to specific services as messages through the message bus. The platform's message bus standardizes and supports communication among all of the platform's components. The data storage and retrieval system groups all of the platforms data storing and servicing modules, implements the platforms data management policy, allowing the platforms modules to exchange data and to rely a consistent and high performing storage solution for the assets extracted and generated by the platform. Finally, the platform's services provide the functionalities required to support the platform's data transformation process, which implements the different algorithms and mechanisms conceived for the project. These services are explained in detail in the following sections. The following Figure 2 shows the conceptual design of the platform's architecture.

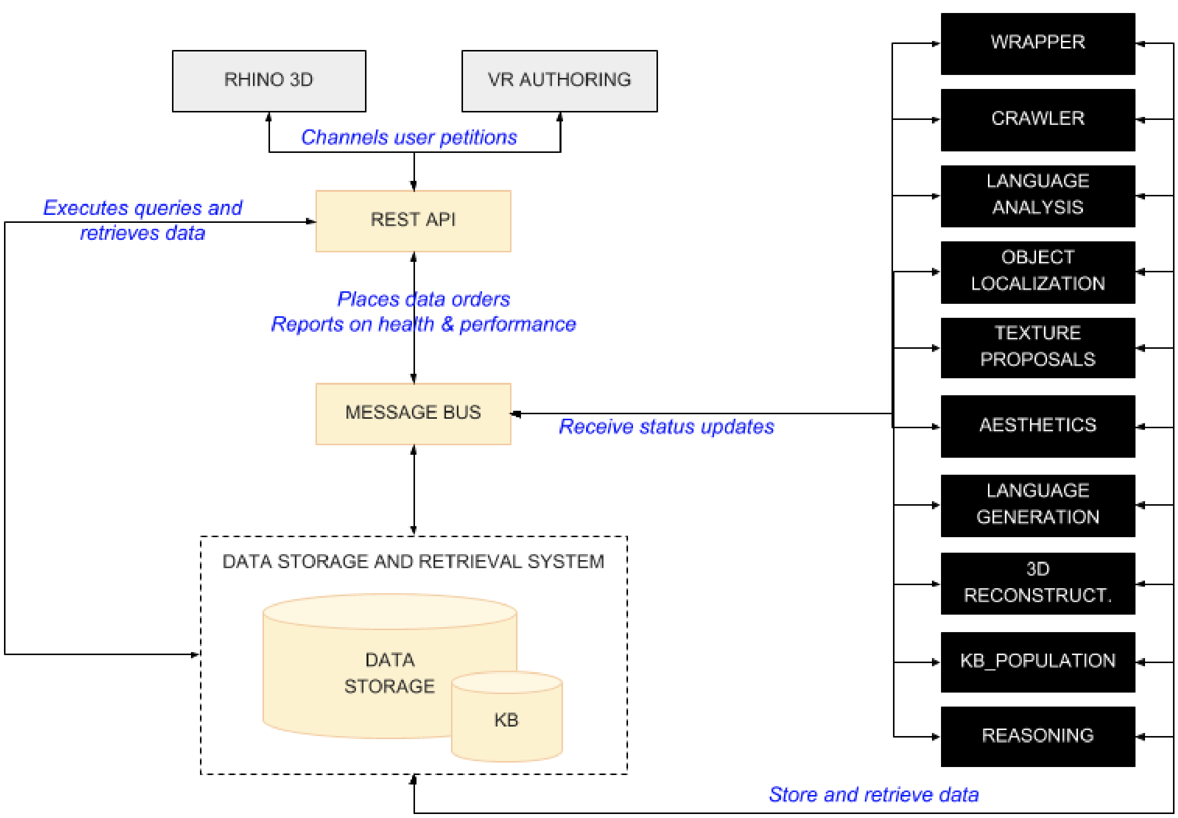


Figure 2: Conceptual design of the platform's architecture

The platform services are designed and implemented according to a generic definition of a V4Design service devised to illustrate the technical and functional requirements that each service needs to meet in order to integrate seamlessly in the platform's ecosystem.

According to this design, a service comprises several technical modules, each performing a specific role. In order to communicate with the rest of the platform's modules and components, a service has two interface mechanisms: first is the message coder and decoder module, which establishes communication with the message bus by sending and receiving messages; and second is the data IO module, which establishes data exchange with the data storage and retrieval system through GET and POST requests. A service must have an authentication mechanism allowing the message bus to identify and trust this service before sending and receiving messages. in addition, a service could have a Queue to store incoming messages if needed. The service core implements the algorithm or the mechanism that the service is created to provide. It generates the service output, which is relevant to other services and/or to the user. Finally, the service could have a local storage module that is used to store local data, or data that is not relevant to any other service or component in the platform.

This conceptual design which provides a generic definition of a V4Design service, is shown in the following Figure 3.

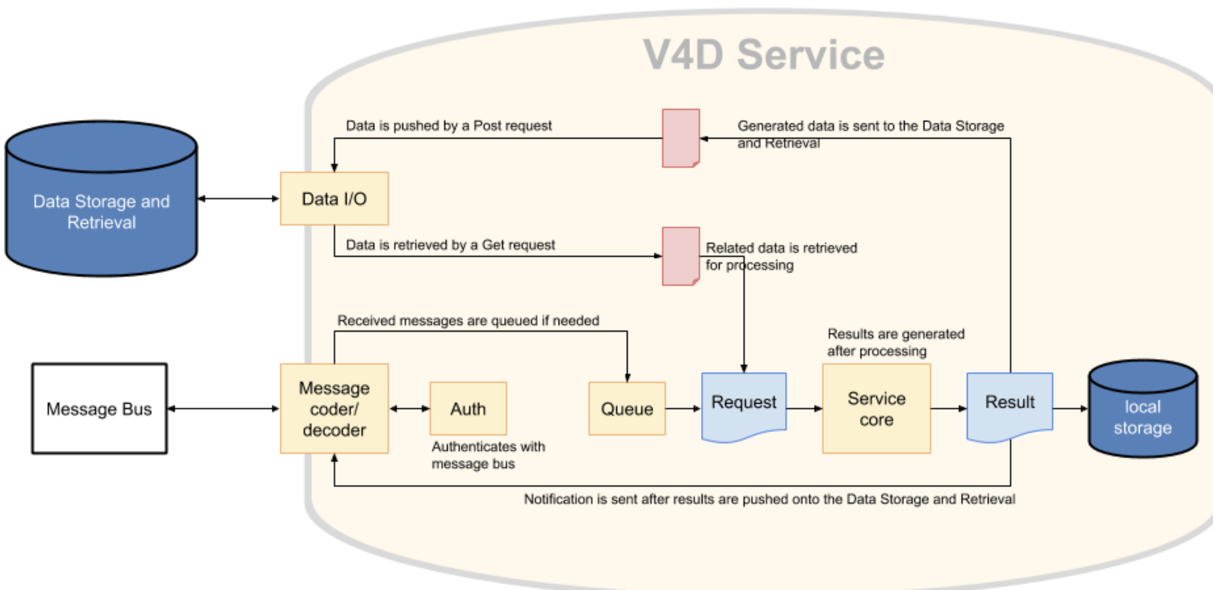


Figure 3: Generic definition of a V4Design service

The platform services can be grouped into three tiers according to their position in the process executed on the acquired data, and the role each performs.

Tier 1 services are basic extraction and data transformation services that operate on the raw data ingested in the system and generate essential elements for other services. These services are not directly related to the platform users operating the tools, who are interested in assets that are more elaborate than those generated by Tier 1 services.

Tier 2 services are designed to generated user-oriented assets, such as reconstructed 3D models, short descriptions, and extracted textures and aesthetics. Tier 2 services require the output generated by Tier 1 services to perform their tasks, and therefore are chained after Tier 1 services.

Finally, Tier 3 services centre on generating intelligence geared to facilitating the user tasks related to asset foraging, asset discovery, and asset compatibility assessment, among others. Tier 3 services are mainly concerned with Tier 2 output but can also utilize the output of Tier 1 services.

This is illustrated in the following Figure 4.

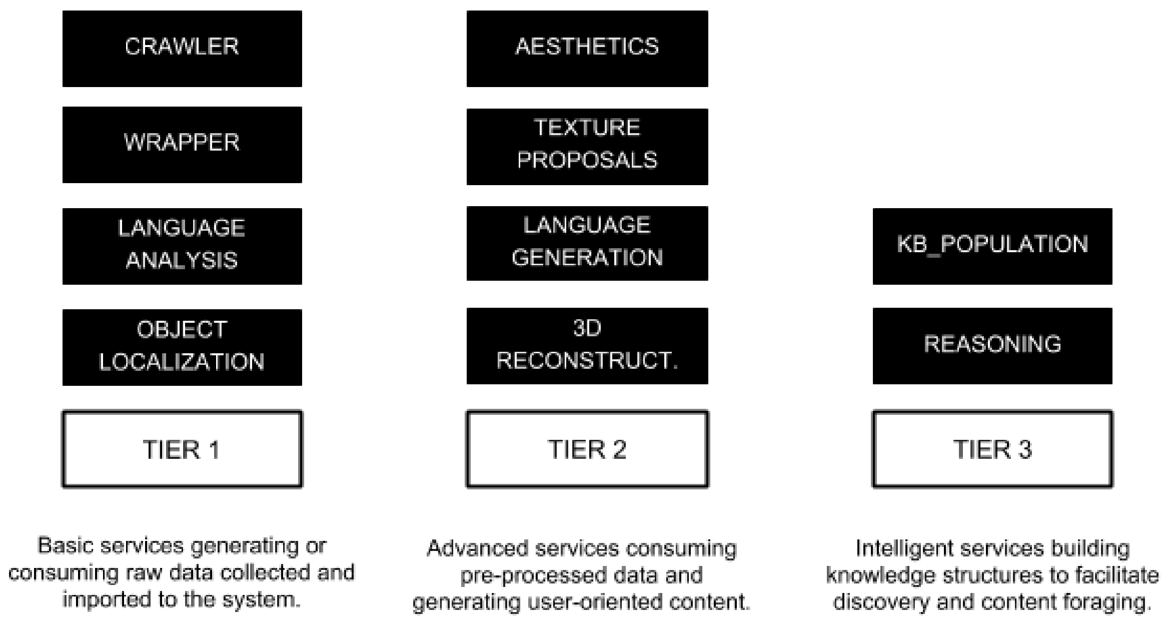


Figure 4: Generic definition of a V4Design service

These three tiers represent a conceptual template that governs the platform processing cycle, by which newly ingested assets are processed to empower users to find and re-use multimedia assets. This cycle chains Tier 1, Tier 2, and Tier 3 services sequentially in the processing of a single asset.

The platform cycle starts when new data arrives to the system. This data is retrieved via the crawler or the wrapper, which send a “New Data” message, announcing the arrival of new raw data to process. Tier 1 services retrieve this data sequentially and process it item per item in parallel. When a Tier 1 service finalizes and produces an output, it sends a message to the concerned Tier 2 services, for instance when the Language Analysis services extract text descriptors and other semantic knowledge, it sends a “Text Analysed” message to the Language Generation, which in turn retrieves the data (original data, and the output generated by Language Analysis), and processes it.

When a Tier 2 service finalizes the processing of an item and produces a result, it sends a message to Tier 3’s Knowledge Base, which monitors and structures the knowledge generated around processed assets and finalizes the platform cycle for each of these assets. A complementary platform cycle is invoked by the API when a request for texture extraction is sent by the user. This runs the platform's cycle starting from Texture Proposals instead of the Crawler or Wrapper, and only executes it partially.

This is explained in detail in the following Figure 5.

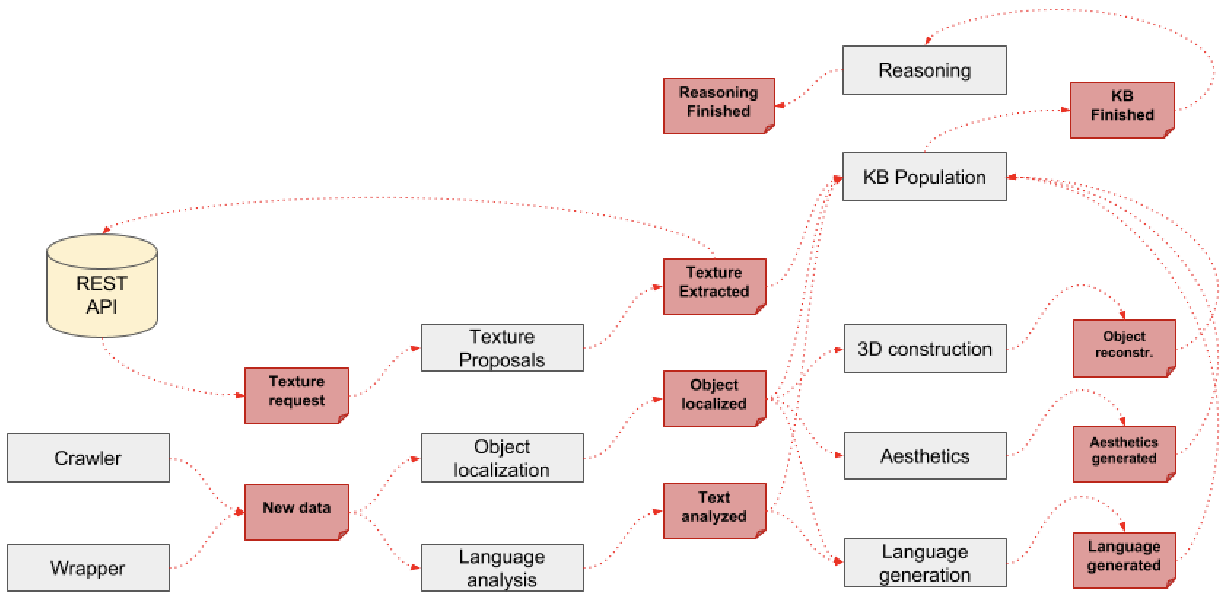


Figure 5: The platform communication model

From a data management perspective, each service extracts its input from the data storage and retrieval system and then pushes its output on the same component. Most of the services read a single asset at a time, but some can read an array of assets if necessary, for instance in the case of the 3D reconstruction service that requires large collections of images in order to perform its tasks. Each service generates a specific output which is then hosted on the data storage and retrieval system according to its type.

All services are triggered by the availability of corresponding data, which is communicated through messages containing, not only information about the status of proceeding processes but also identifiers of the related data objects and assets. This allows the service to retrieve the data for processing without having to query the data storage and retrieval system.

This is illustrated in the following Figure 6.

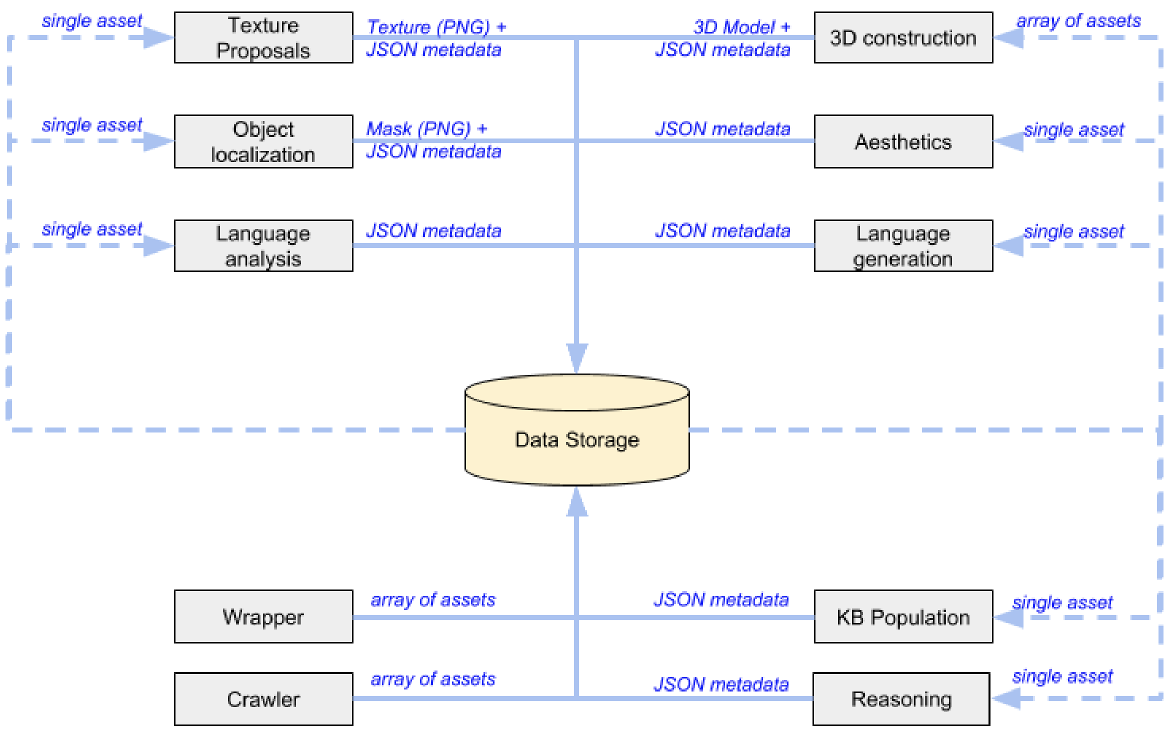


Figure 6: The platform input/output model

This description of the platform architecture details its design and integration approach, which takes into consideration the requirements in terms of maturity level that the platform has to reach by the end of the project, and the possibility to add new services and tools to expand the platform in the future. In addition, this approach concerns such as security, scalability and performance.

In the context of the operational prototype, or the first integrated version of the platform, the design and integration approach of the platform architecture acts as guidelines for the developers of the architectural components of V4Design, especially the services.

2.2 Service prototypes and their development roadmaps

In the following section we present the V4Design service prototypes, each described according to its function and technical requirements. The development roadmap envisioned for each service and primarily described in D6.1 is revisited and updated according to the current state of development and the progress achieved in the project.

2.2.1 The Language Analysis

The Language Analysis module captures and analyses the textual information associated with a retrieved asset and creates structured ontological representations. It combines multilingual dependency parsers and lexical resources, and a projection of the extracted dependency-based linguistic representations into ontological ones.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 1: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_LA_1	Extract knowledge from textual data to be able to map it to the KB	Linguistic Analysis	Tokenization, Part-of-speech tagging, Lemmatization, Surface-syntactic parsing.
TR_LA_2	Extract knowledge from textual data to be able to map it to the KB	Concept extraction	Word Sense Disambiguation, Entity linking.
TR_LA_3	Extract knowledge from textual data to be able to map it to the KB	Relation Extraction	Deep-syntactic parsing, Conceptual relation extraction.

The Language Analysis pipeline comprises the following modules: tokenization (TR_LA_1), PoS tagging (TR_LA_1), lemmatization (TR_LA_1), word sense disambiguation (TR_LA_2), entity linking (TR_LA_2), concept extraction (TR_LA_2), surface syntactic parsing (TR_LA_3), semantic parsing (TR_LA_3) and conceptual relation extraction (TR_LA_3).

When new textual content has been crawled/scraped, the Language Analysis pipeline receives a message and starts processing the document(s) in the following order:

- Concept extraction, entity linking and disambiguation;
- PoS tagging, lemmatization, morphological analysis and Syntactic parsing;
- Semantic parsing;
- Conceptual relation extraction;

The output generated by Language Analysis for the following input text:

“It is recognised by UNESCO as a World Heritage Site.”

Table 2: Output example of Language Analysis

JSON file


```
"UNESCO_1" : {
	"sameAs" : ["http://dbpedia.org/page/UNESCO”],
	“organization": "true"
},
"World_Heritage_Site_1" : {
"type" : [“http://dbpedia.org/ontology/WorldHeritageSite”],
},
"Delphi_1" : {
"sameAs" : [“http://dbpedia.org/page/Delphi”],
“Type” : [“http://dbpedia.org/ontology/Place”, “yago:Sanctuary”],
	"location": "true"
},
```

```
"recognize_1" : {
 "type" : [“http://conceptnet.io/c/en/recognize”],
        "properties" : {
          "involvesAgent" : ["UNESCO_1"],
          "involvesPatient" : ["Delphi_1"],
          "involvesCoPatient" : ["World_Heritage_Site_1"]
        }
}
```


The roadmap is the same as described in D6.1:

V1 [M12]: Operational prototype. The language analysis pipeline will be able to output language-independent representations starting at least from English, for a limited set of input sentences. V1 is particularly focused on all TR_LA_1 and TR_LA_3 modules, and on the concept extraction module of TR_LA_2.

V2 [M16]: Basic version of multilingual language analysis. The analysis pipeline will be operational for at least three languages, and its coverage will be improved according to the specifications of the different UCs. The quality of the outputs will be evaluated.

V3 [M34]: Final version of multilingual language analysis. The analysis pipeline will have an improved coverage and will be able to handle all the V4Design languages (English, Spanish, Greek, and German). Efforts will be dedicated to ensure the reusability of the developed tools outside of V4Design.

In the following we discuss the current status of this prototype.

Surface-syntactic analysis:

- Current language: English
- Next languages (ready, to be integrated): Spanish and Greek
- Current formalism: Penn Treebank style
- Alternative formalism (ready, to be integrated): Universal dependencies.
- Tools and resources used: off-the-shelf dependency parser and Penn Treebank corpus

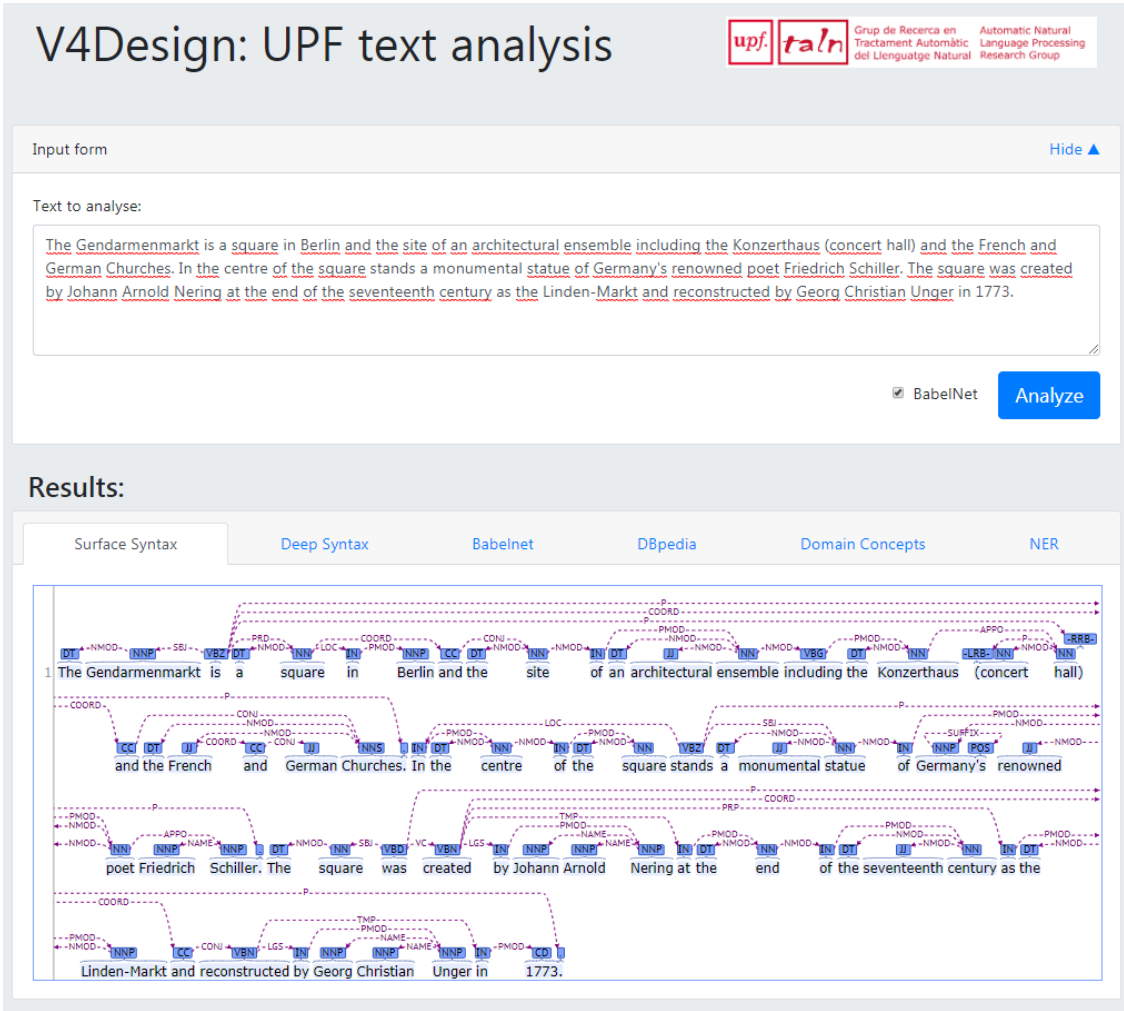


Figure 7: Surface-semantic analysis configuration

Semantic analysis

- Current language: English
- Next language: Spanish (ready, to be integrated) and Greek (to be developed)
- Current formalism: Meaning-Text Theory
- Alternative formalism (ready, to be integrated): Universal Dependency-based deep structures
- Current level of abstraction: Deep Syntax - language-specific
- Next target for level of abstraction: Conceptual - language-independent (under development)
- Tool and resources used: UPF graph-transduction grammars and lexical resources

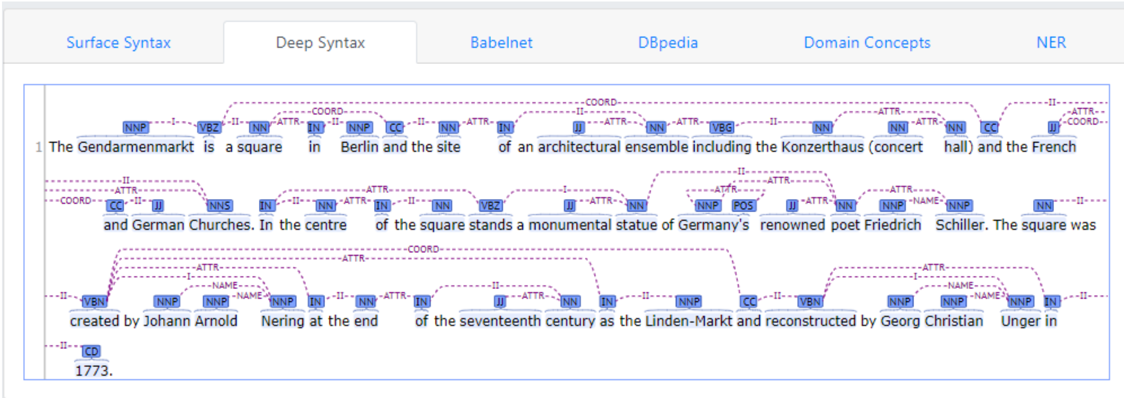


Figure 8: Semantic analysis configuration

Word Sense Disambiguation (part of concept extraction)

- Current language: English
- Next languages (ready, to be integrated): Spanish, Greek, German
- Current resources used: BabelNet, Wikipedia, WordNet
- Currently evaluating various approaches (baseline and experimental) fed with our candidate detection (see below).

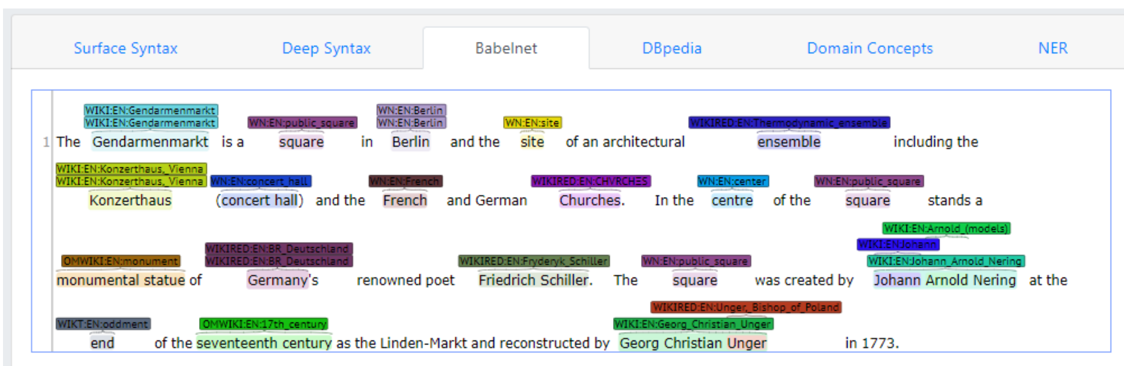


Figure 9: Word Sense Disambiguation configuration

Entity linking (part of concept extraction)

- Current language: English
- Next language (ready, to be integrated): Spanish
- Current resource used: DBpedia
- Currently using off-the-shelf DBpedia Spotlight fed with our candidate detection (see below)

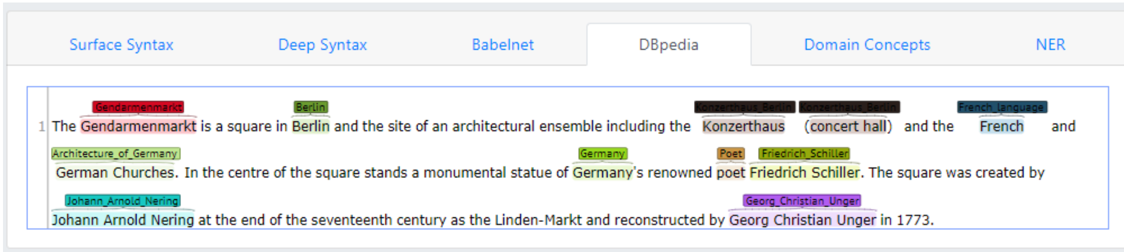


Figure 10: Entity linking configuration

Concept candidate detection (part of concept extraction)

- Current language: English
- Next language (under development): Spanish
- Current issues: slow (V0.1)

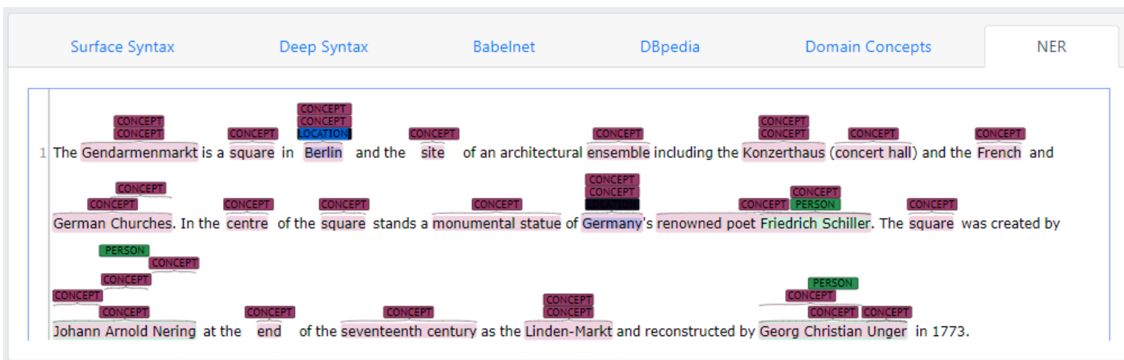


Figure 11: Concept candidate detection configuration

2.2.2 The Language Generation

The language generation module is in charge of generating textual reports, descriptions, or summaries, starting from data extracted from text, webpages, and/or visual analytics. It generates a summary of most relevant contents related to a specific keyword and/or entity.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 3: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_LG_1	Select content to be generated as texts and shown to the users.	Text Planning	Identify contents related to the queried entity, assesses their relevance relative to this entity.
TR_LG_2	Render the selected content as text.	Linguistic Generation	Generates text in target language.

The Language Generation pipeline comprises the following modules: content selection (TR_LG_1), lexicalization (TR_LG_2), sentence structuring (TR_LG_2), morphological agreements resolution (TR_LG_2), and word order resolution (TR_LG_2).

When the reasoning module is done processing new visual and/or textual contents, the resulting information is sent to the Language Generation module that performs the following actions:

- Selection of the content to be verbalized;
- Lexicalization and sentence structuring in the target language;
- Resolution of word order and morphological agreements.

The Language Generation module uses the abstract representation generated by Language Analysis to create meaningful descriptions, for example (using the output example of Language Analysis): "UNESCO recognizes Delphi as a World Heritage Site".

The roadmap is the same as described in D6.1:

V1 [M12]: Operational prototype. Generation of a few sentences from ontological representations will be supported in English. Some basic summarization techniques (e.g. extractive summarization) will be implemented for handling possible textual inputs. V1 particularly focuses on the TR_LG_2 modules

V2 [M18]: Basic summarization techniques. The generator starting from ontological structures will be adapted to one or two more languages, and its coverage will be increased (all depending on the UC requirements). A first version of the ontology-based text planning will be setup and connected with the generator.

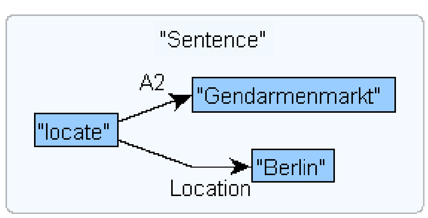
V3 [M33]: Final summarization techniques. The ontological generator will handle all V4Design languages (English, Spanish, Greek, and German) and cover all defined use cases, and will include statistical submodules when needed. The advanced version of the text planning module will be released, which will aim at optimizing the relevance and coherence of the summaries. Efforts will be dedicated to ensure the reusability of the developed tools outside of V4Design.

In the following we discuss the current status of this prototype. No online demo is available so far, so we discuss the main successive steps followed during the generation process.

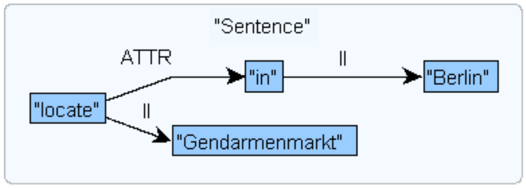
Input

Location (Berlin, Gendarmenmarkt)

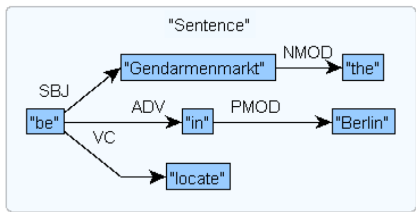
Mapping to predicate-argument structure



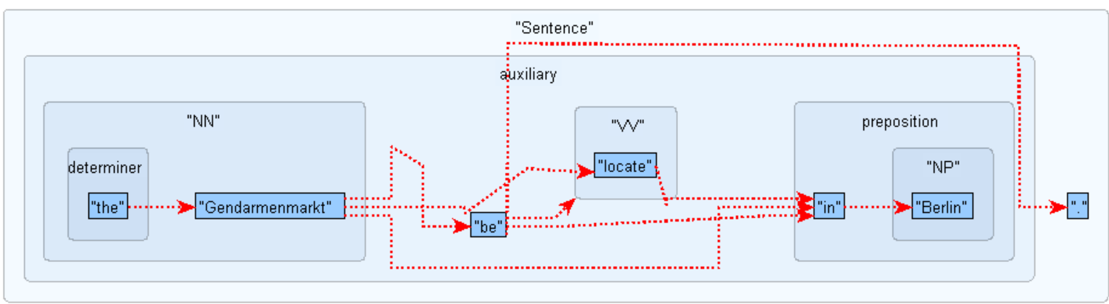
Mapping to deep-syntactic structure (sentence structuring)



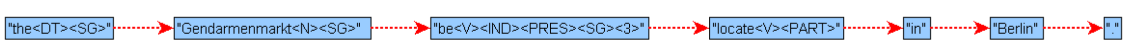
Mapping to surface-syntactic structure (introduction of functional elements and fine-grained grammatical relations)



Linearization and introduction of punctuation signs



Resolution of morphological agreements



Surface form retrieval

2.2.3 **The V4Design Crawler**

The crawler service integrates all the crawling and scraping functionalities envisioned in the project, in order to extract freely available textual and visual content from open web resources, including from social media.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 4: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_CR_1	Using a set of URLs as web entry points, collect all the hyperlinked ULRs, up to a predefined depth.	Web crawling	Discovers nodes to scrape
TR_CR_2	Add more keywords to refine the search operations.	Query expansion	Discovery of extra keywords relevant to the input query
TR_CR_3	With the help of API, search a web application (e.g. Flickr) using textual queries.	Web search	Depending on the available APIs, scraping may also be performed
TR_CR_4	Extract assets from web pages	Web scraping	Extracts content from web pages
TR_CR_5	Search and collect social media posts relevant to a keyword or a user account.	Social media crawling & scraping	Extracts content from social media
TR_CR_6	looks at the FTP server folders of a content provider to see if any new content has been added, and if so extracts it to add to data storage	FTP crawling	extracts content from the V4design FTP server
TR_CR_7	based on an EDM file or a generic JSON file, check if this JSON is SIMMO-compliant. If not, use predefined maps to make this JSON file SIMMO compliant. Send to data storage	data model mapping	maps incoming data from the incoming data model to SIMMO JSON
TR_CR_8	Application of classifiers that categorize the resources as appropriate or not for our purposes	Resource filtering	categorizing the resources as appropriate or not for our purposes

V4Design Crawler is the component that produces new data for the pipeline, based on predefined web entry points and queries. The following figure describes how it works step-by-step.

In a scenario where the crawler is searching Flickr for the query “Eiffel Tower”, the following example illustrates the array of SIMMOs produced:

Table 5: Output example of the Crawler


```
[
 { "_id" : "f240773e-949a-4439-90d9-82a43d7dc201",
 "className" : "gr.iti.mklab.simmo.core.items.Image",
 "thumbnail" : "https://farm7.static.flickr.com/6220/6371213275_6936c0378c_t.jpg",
 "source" : "Flickr",
```

```
 "type" : "IMAGE",
 "url" : "https://flickr.com/photos/14443335@N06/6371213275",
 "title" : "Eiffel Tower_Wide-1",
 "tags" : [ "paris", "eiffeltower" ],
 "crawlDate" : ISODate("2018-11-07T14:39:44.749Z"),
 "searchQuery" : "eiffel tower" },
 { "_id" : "95607186-0c30-4ad8-9de0-735186b93f54",
 "className" : "gr.iti.mklab.simmo.core.items.Image",
 "thumbnail" : "https://farm3.static.flickr.com/2711/4480893837_37215b9db3_t.jpg",
 "source" : "Flickr",
 "type" : "IMAGE",
 "url" : "https://flickr.com/photos/47077636@N07/4480893837",
 "title" : "Eiffel Tower at Night",
 "tags" : [ "paris", "eiffeltower", "france", "eiffel" ],
 "crawlDate" : ISODate("2018-11-07T14:39:44.749Z"),
 "searchQuery" : "eiffel tower" },
....
 "crawlDate" : ISODate("2018-11-07T14:39:44.749Z"),
 "searchQuery" : "eiffel tower"
}]
```


The development roadmap for this service is described in the following:

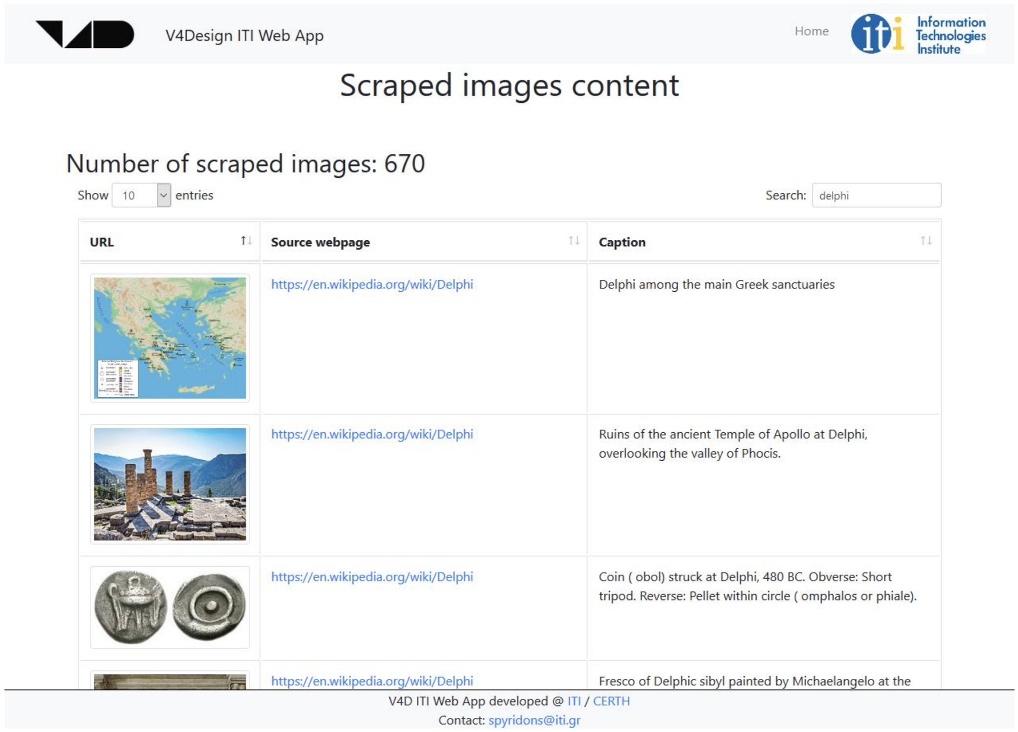
Operational Prototype [M12]: includes the implementation of the Web crawling and scraping component, the Social media search component, and the Web search component. This prototype is integrated with the platform’s message bus.

1st Prototype [M18]: Implementation and integration of Query expansion and Resource Filtering.

2nd prototype [M24]: Delivery of an advanced version of web/social media scraping and search.

Final Prototype [M30]: Update of all the components and finalization of the module.

In the following we show a screen capture of the Crawler interface.



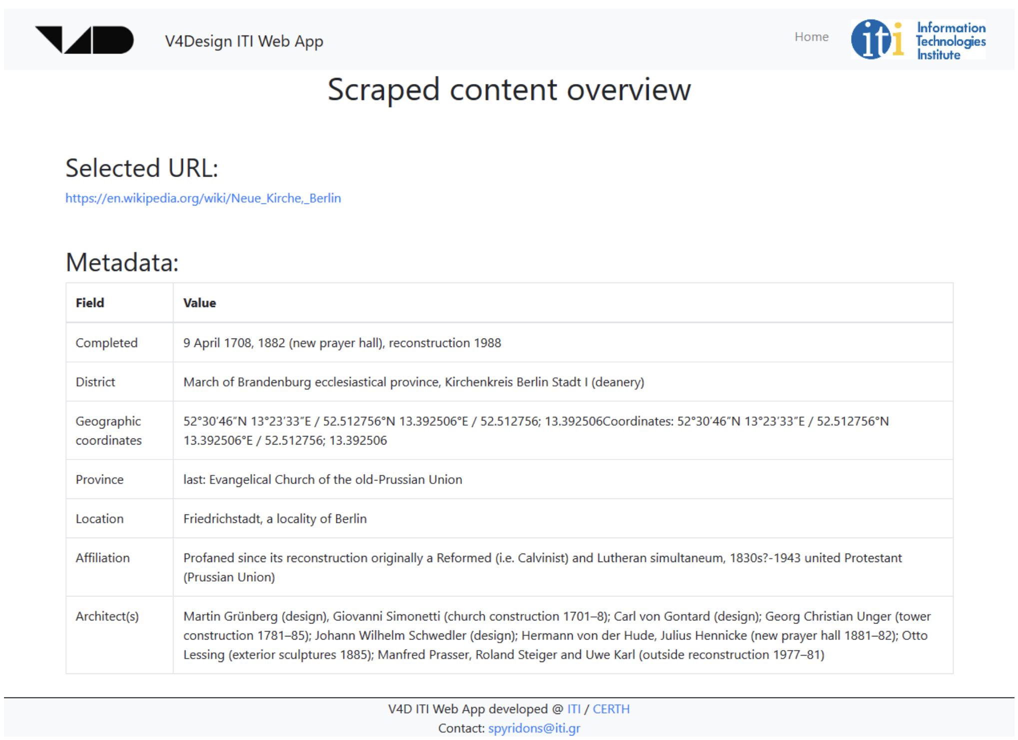


Figure 12a, b: The V4Design Crawler interface

2.2.4 **Aesthetics Extraction and Texture Proposals (AE&TP)**

The Aesthetic Extraction (AE) and Texture Proposals (TP) service extracts and categorizes the aesthetics of media assets that contain architecture objects and buildings based on their style (i.e. impressionism, cubism and expressionism), creator and emotion that they evoke to the viewer and combine them so as to produce/suggest novel textures

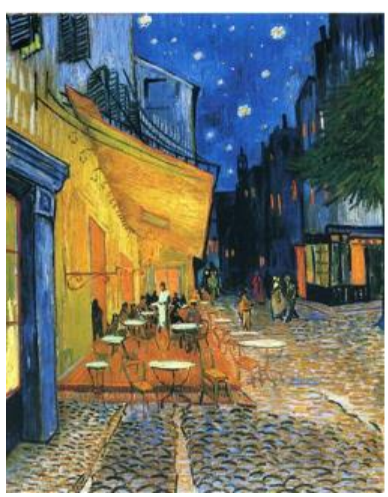
The technical requirements that this service aims to fulfil are summarized in the following table.

Table 6: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_AE_1	extract texture and style for images and videos so as to be able to retrieve patterns, textures and styles	Aesthetics extraction (AE)	Aesthetics extraction from paintings, clustering, model extraction and storing on a local file storage
TR_TP_1	combine textures and styles to propose them in the generation of a new image	Texture proposals (TP)	Transfer painting style from the desired image or aesthetic model and pass it to the goal image

Initially, an offline process will run so as to build the initial AE models. For that purposes, AE components will need to gather a great deal of annotated images that consist of renowned paintings, buildings and architecture objects. These data will be crawled from the web and/or compiling data from the content providers APIs and when enough images are compiled (>10K batch size), the AE component will be notified by the message bus, retrieve these data and build/update the aesthetics models. These models will be stored in V4Design server's file storage and will be used to define the aesthetics category of a building, object and painting that will be acquired during the online process. Furthermore, the top 50 results of each category will be depicted to the V4Design user through the V4Design interface. The user will be able to select the desired painting style that he would like to transfer to his creation (3D model) and alter its texture using the TP component, which will perform this process.

The Aesthetic module takes the following input as example: {Impressionism, Vincent Van Gogh, painting}, Storage format: h5



The corresponding output will be:

Table 7: Output example of Aesthetic Extraction

Field name: style Type: DCNN-model (.h5) Allowed values: {Baroque, Impressionism, Expressionism, Cubism, Rococo, Minimalism, Abstract Expressionism, Action painting, Analytical Cubism, Art Nouveau, Colour Field Painting, Contemporary Realism, Early Renaissance, Fauvism, High Renaissance, Mannerism Late Renaissance, Naive Art Primitivism, New Realism, Northern Renaissance, Pointillism, Pop Art, Post Impressionism, Realism, Romanticism, Symbolism, Synthetic Cubism, Ukiyo-e} Field name: creator Type: DCNN-model (.h5) Allowed values: { Salvador Dali, Vincent Van Gogh, Pablo Picasso,Albrecht Durer, Boris Kustodiev, Camille Pissarro, Childe Hassam, Claude Monet, Edgar Degas, Eugene Boudin, Gustave Dore, Ilya Repin, Ivan Aivazovsky, Ivan Shishkin, John Singer Sargent, Marc Chagall, Martiros Saryan, Nicholas Roerich, Pierre Auguste Renoir, Pyotr Konchalovsky, Raphael Kirchner, Rembrandt,Paul Cezanne,} Field name: type Type: string Allowed values: {painting, building, object}

The Texture Proposal module takes the content and style images as input and produces the following examples:

Table 8: Output example of Texture Proposal

The development roadmap for this service is described in the following:

Version 1 [M12]: 1st version of the basic aesthetics and texture proposals is released and integrated with the platform and message bus.

Version 2 [M26]: The basic version of the algorithm integrated in V4Design system

Version 3 [M33]: Advanced version deployed.

In the following we show screen capture of the Aesthetics extraction service interface.

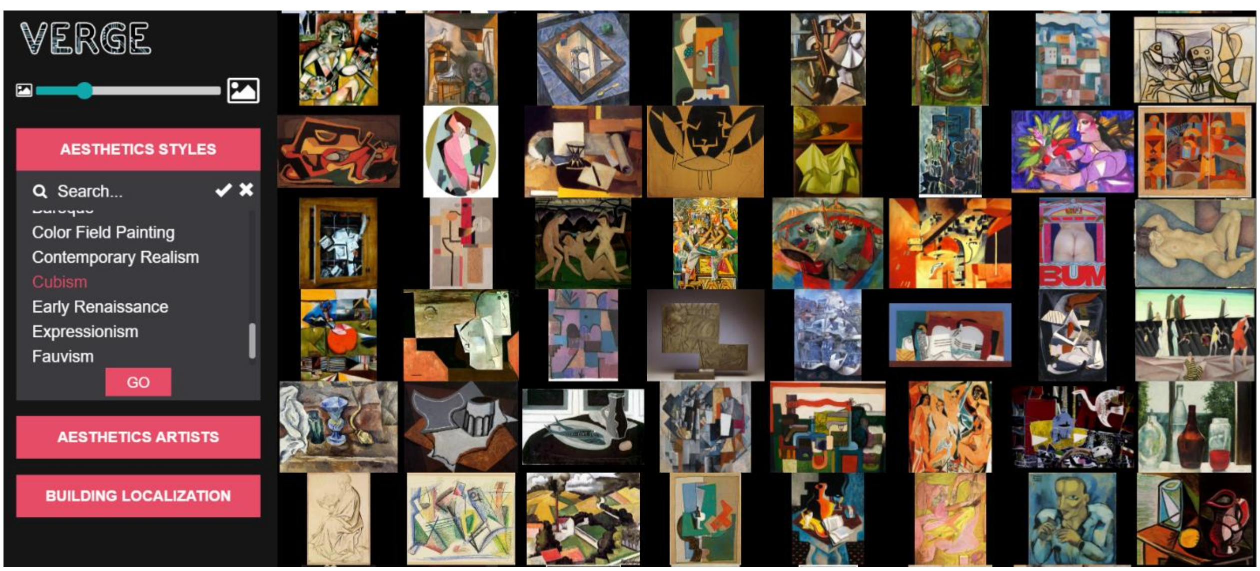


Figure 13: The Aesthetics Extraction interface

2.2.5 **KB Population**

The KB Population service is responsible for mapping the results generated by the different V4Design services to the RDF-based representation format, based on the ontologies that will be developed to provide the annotation models. This involves the development of vocabularies for capturing texture and aesthetics, semantic relations (e.g. named entities, concepts and relations), and various properties, such as artists, year etc., buildings, interior objects and other content-specific attributes (e.g. landscapes, architectural styles, etc.). The underlying knowledge structures will also provide all the necessary semantics needed to generate textual descriptions and summaries for each asset.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 9: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_KB_1	Map analysis results from other modules	Populate	RDF mapping and KB population
TR_KB_2	Provide an API over the KB for querying metadata	Populate	RDF mapping and KB population
TR_KB_3	Map metadata about texture resolution	Populate	RDF mapping and KB population
TR_KB_4	Map analysis results from building localisation	Populate	RDF mapping and KB population
TR_KB_5	Map analysis results from object localisation	Populate	RDF mapping and KB population
TR_KB_6	Map analysis results from aesthetics	Populate	RDF mapping and KB population
TR_KB_7	Map analysis results from text analysis	Populate	RDF mapping and KB population
TR_KB_8	Map analysis results from reasoning	Populate	RDF mapping and KB population
TR_KB_9	Map metadata about quality	Populate	RDF mapping and KB population
TR_KB_10	Map geo-location of assets	Populate	RDF mapping and KB population
TR_KB_11	Map date (creation date)	Populate	RDF mapping and KB population
TR_KB_12	Map author info	Populate	RDF mapping and KB population
TR_KB_13	Map copyright info	Populate	RDF mapping and KB population
TR_KB_14	Map visible colours	Populate	RDF mapping and KB population

TR_KB_15	Map metadata coming from 3D model reconstruction	Populate	RDF mapping and KB population
TR_KB_16	Map results from text generation	Populate	RDF mapping and KB population
TR_KB_17	Ability to associate assets with relevant external Web Pages	Populate	RDF mapping and KB population
TR_KB_18	Map results from text generation	Populate	RDF mapping and KB population
TR_KB_19	Associate assets with preview thumbnails	Populate	RDF mapping and KB population
TR_KB_20	Ability to map texture material metadata	Populate	RDF mapping and KB population
TR_KB_21	Support the linking of assets with relevant ones	Populate	RDF mapping and KB population
TR_KB_22	Support the annotation of assets with reuse rights and copyrights	Populate	RDF mapping and KB population
TR_KB_23	Map analysis results from text generation	Populate	RDF mapping and KB population

The service is triggered whenever some other module of the pipeline produces results. The results are stored in the Data Storage by the module that generates them and publishes a message informing other modules how to obtain the results. KB Population reads these messages, retrieve the results from the data storage, generates the mapping (RDF triples) and stores the results in the KB.

A Building localisation example output would be:

Table 10: Output example of KB Population

```
{ "simmo": "http://v4design-ds.com/simmo/<ref>",
 "assets": [ {
 "type": "image",
 "original": "http://v4design-ds.com/file/<imageId>",
 "mask": "https://v4design-ds.com/file/<maskId>",
 "tags": [ "tower" ] } ]
}
```

RDF mapping:

Table 11: RDF mapping of KB Population output

```
@prefix oa: <http://www.w3.org/ns/oa#> .
@prefix v4d: <https://v4design.eu/ontologies/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
<https://v4design.eu/ontologies/simulation_v2#BuildingLocalisationAnnotation_1>
 a <https://v4design.eu/ontologies/BuildingLocalisationAnnotation> ;
 oa:hasBody <https://v4design.eu/ontologies/simulation_v2#LocalisationBuildingView_1> ;
 oa:hasTarget <https://v4design.eu/ontologies/Mask_1> .

<https://v4design.eu/ontologies/Mask_1>
 a <https://v4design.eu/ontologies/Mask> ;
 v4d:uri "https://v4design-ds.com/file/Mask_1" .

v4d:simulation_v2#LocalisationBuildingView_1
 a v4d:LocalisationBuildingView ;
 v4d:originalImage v4d:Image_1 ;
 v4d:tag <https://babelnet.org/synset?word=bn:00077766n> .

<https://babelnet.org/synset?word=bn:00077766n> rdfs:label "tower" .
v4d:Image_1
 a v4d:Image ;
 v4d:simmoRef "http://v4design-ds.com/simmo/5ac38f1bca994aefd5f3e6be" ;
 v4d:uri "http://v4design-ds.com/file/Image_1" .
```

The development roadmap for this service is described in the following:

Operational prototype [M12]: Basic mapping functionality will be available towards v1. This involves the delivery of the mapping algorithms able to populate the KB with real results generated by the current version of the V4Design components. The interaction with the bus will be also implemented and tested, aligning the subscription mechanisms to the events published by the analysis modules.

V1 [M20]: Fully fledged mapping service, supporting the full structure and content of the outputs generated by the V4Design modules for v1. The mapping algorithms in M20 will extend the ones developed in M12, taking into account updates and refinements made in the V4Design modules to address the technical and user requirements.

V2 [M28]: Necessary updates for v2, in line with the updated structure and content provided by the analysis modules. This involves the update of the mapping algorithms to support the richer inputs that will be provided by the components, as well as to update the publishing and subscription mechanisms to the bus in order to realise the communication with the other modules of the framework. Special focus will be given on the semantic enrichment of the incoming information, e.g. by including additional references to Linked Data resources.

V3 [M36]: Necessary improvements on the final system, according to the updates made on the output (structure and content) provided by the other components. In the final version the focus will be also given on the scalability of the mapping algorithms, as well as on developing fall-back strategies when the incoming information is incomplete. The possibility of a tighter interaction with the Reasoning service will be also investigated, according to the need to incorporate some sort of reasoning in the mapping process (this depends on the semantics of the input that will be provided).

Operational prototype demo

In order to test the KB Population module, we have developed a demo page that performs on the fly transformation of various V4Design modules into the V4Design Annotation Metamodel described in D5.1. As depicted in Figure 14, all four V4Design modules are supported. When the user selects a module, an example output is shown (Figure 15) and by clicking on the Convert button, the results of the mapping is shown. Figure 16 depicts the results by selecting the Aesthetics module. At the same time, the RDF triple store is populated with the results of the mapping. In Figure 17, the RDF graph semantically annotates `image1` with the BabelNet "minimalism" resource (`https://babelnet.org/synset?word=bn:00055162n`), linking V4Design Knowledge Base with the BabelNet semantic network.

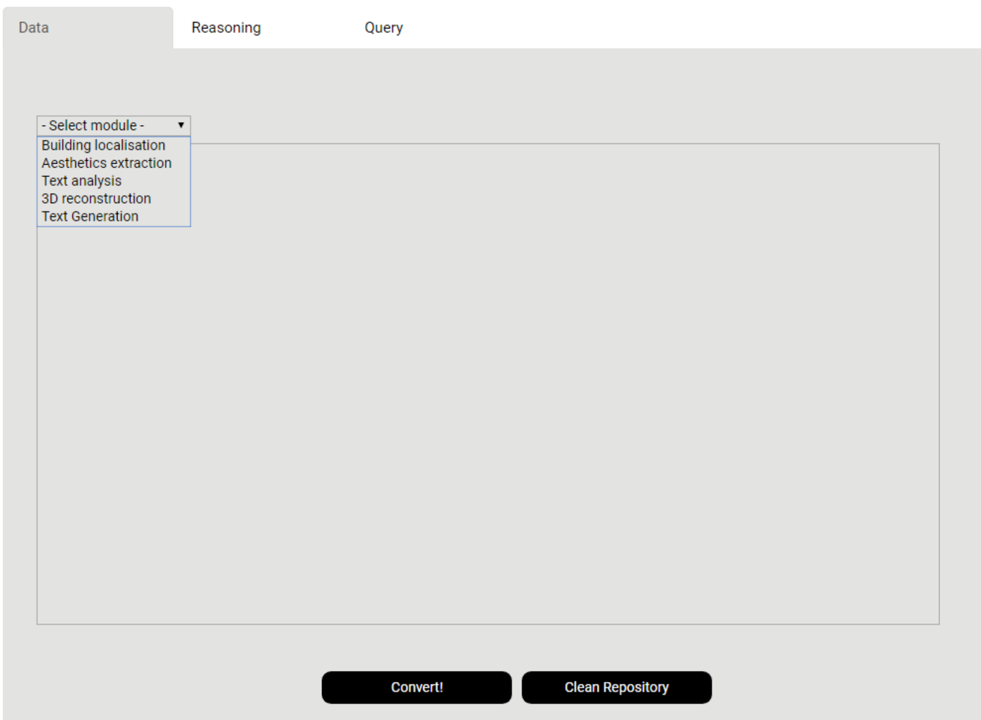


Figure 14: Home page for the KB Population demo

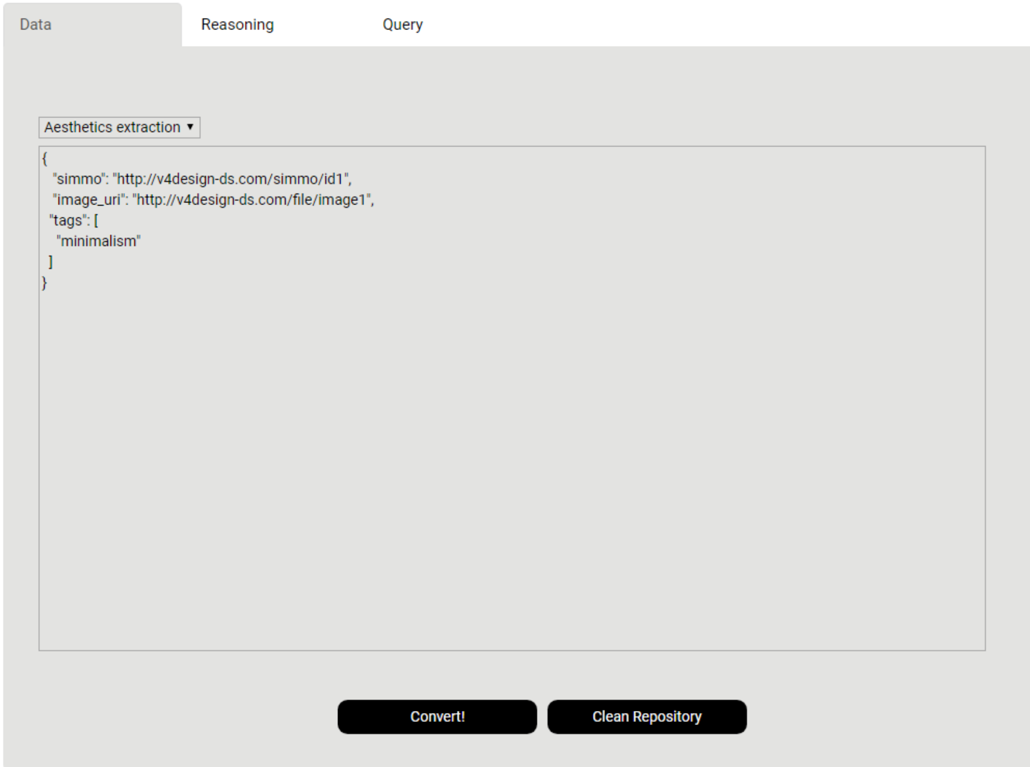


Figure 15: Example output of the Aesthetics module

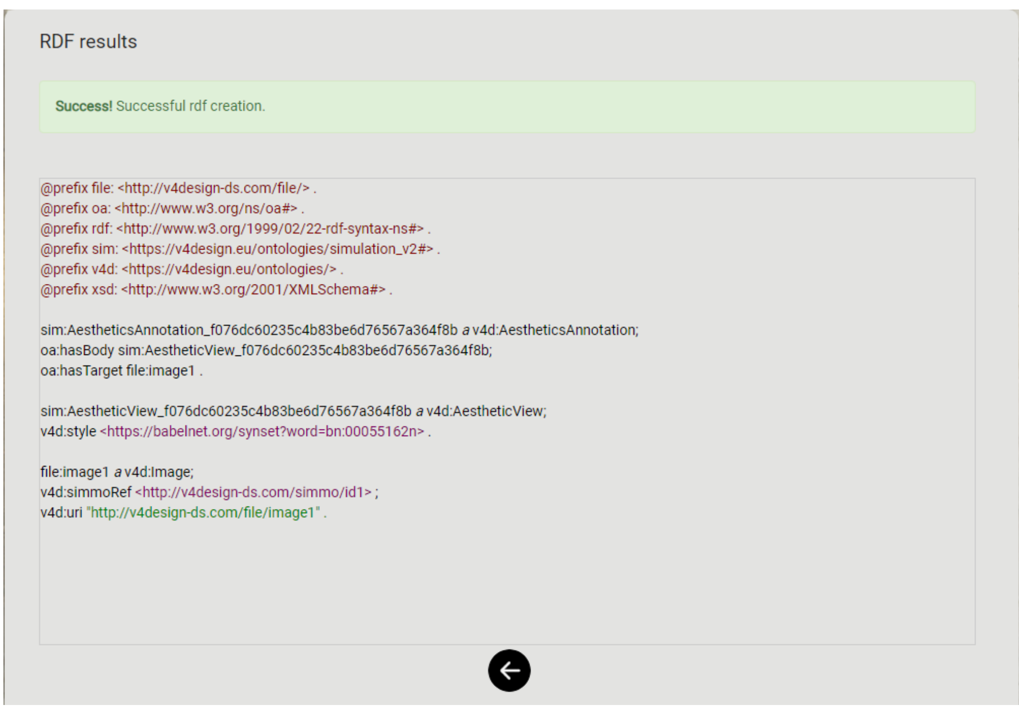


Figure 16: RDF mapping results of Aesthetics output

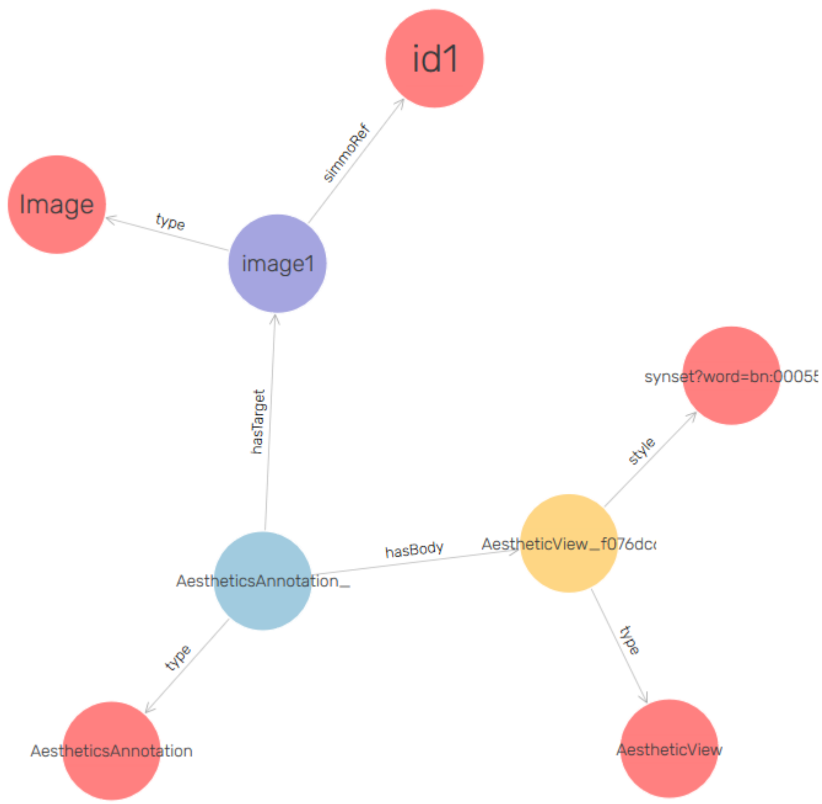


Figure 17: RDF visual graph of Aesthetics mapping

2.2.6 **Reasoning**

The reasoning service builds a unified representation of the available assets, taking into account information relevant to texture and aesthetics, named entities, concepts and relations extracted from textual analysis, as well as buildings, interior objects and other content-specific attributes. The component will be also responsible for query formulation, i.e. the translation of interface requests into one or more queries to the backend data storage infrastructure in order to retrieve and send back results.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 12: Corresponding functional requirements

TR NB	**Description**	**Function**	**Function performed**
TR_RQ_1	Support searching functionality (translation of user requests into one or more queries over the data storage	Reasoning Service	query formulation / enrichment
TR_RQ_2	Infer geolocation from location tag	Reasoning Service	Inference of implicit relations and context

TR_RQ_3	Propagate annotations from other modalities to the 3D models	Reasoning Service	Inference of implicit relations and context
TR_RQ_4	Find relevant external Web page, based on the annotation provided by other components	Reasoning Service	Inference of implicit relations and context
TR_RQ_5	couple searching functionality with text analysis on the keywords	Reasoning Service	query formulation / enrichment
TR_RQ_6	Find assets relevant to other assets	Reasoning Service	Inference of implicit relations and context

Example rule for the Reasoning service:

```
PREFIX oa: <http://www.w3.org/ns/oa#>
PREFIX : <https://v4design.eu/ontologies/>
CONSTRUCT {
    [] a :_3DModelView;
        :tag ?style.
}
WHERE {
    ?annotation1 a :AestheticsAnnotation;
                oa:hasBody ?view1;
                oa:hasTarget ?image.
    ?annotation2 a :_3DModelAnnotation;
                oa:hasBody ?view2.
    ?view2 a :_3DModelView;
           :image ?image.
    ?view1 a :AestheticView;
          :style ?style.
}
```

An example output:

Table 13: Output example of Reasoning

] a :_3DmodelView; :tag <https://babelnet.org/synset?word=bn:00013722n>.

The development roadmap for this service is described in the following:

The service is triggered whenever results are stored in the KB. Therefore, it listens to published by KB Population.

Operational [M12]: Basic reasoning functionality will be available towards V1. This involves the development of the rule-based reasoning framework able to combine existing tags and generate high-level concepts, semantically enriching the captured context.

V1 [M20]: Reasoning functionality aiming to address the V1 requirements. This involves the extension of the reasoning framework developed in M12 with advanced multimodal information fusion and content aggregation techniques to generate higher level conceptualizations for content repurposing. A hybrid reasoning scheme of Description Logics and rule-based reasoning will be investigated.

V2 [M28]: Necessary updates for V2, based on the evaluation of V1 and the new input provided by the other modules. In addition, the reasoning framework will be further enriched with non-monitoring capabilities, addressing challenges relevant to content disambiguation and handling of conflicts, e.g. in the case when conflicting information is received from different modules.

V3 [M36]: Necessary updates for the final system. Improvements on the scalability will be investigated, while similarity measures will be implemented for advanced Linked Data resource linking and approximate reasoning (e.g. to define clusters of relevant assets).

Operational prototype demo

The demo page for KB population also contains the “Reasoning” tab that can be used to run example rule on top of the V4Design Knowledge Base and get the inferred results (Figure 18). More specifically, we use SPIN rules, i.e. SPARQL construct graph patterns, to implement expressive reasoning rules, enabling property value propagation and instance generation (when needed). The core idea is to associate each reasoning task with one or more SPARQL rules that address specific reasoning requirements, e.g. to propagate aesthetics from images to the 3D models. The rule that is currently supported is the one described in D5.1 (section 6.2.2) about enriching 3D models with aesthetics annotations. In this example, we assume that the knowledge base contains the mapping result of aesthetics and the mapping result of 3D model reconstruction. The supported inference rule is used to propagate the aesthetics of images, which have been used to reconstruct a 3D model, to the 3D object itself (Figure 19). The result of the rule is stored in the RDF graph, semantically associating the output of the aesthetics module with the 3D model annotations (Figure 20).

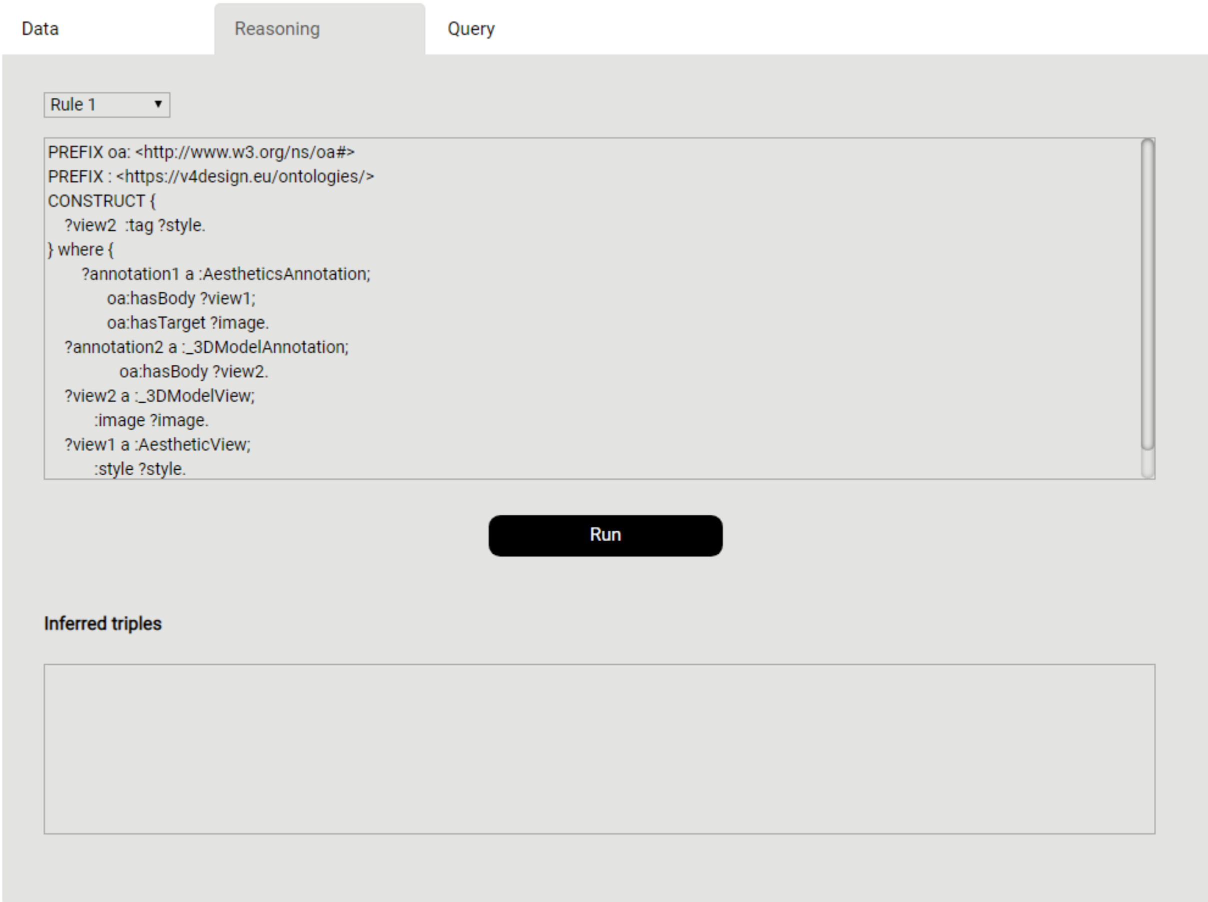


Figure 18: Example rule

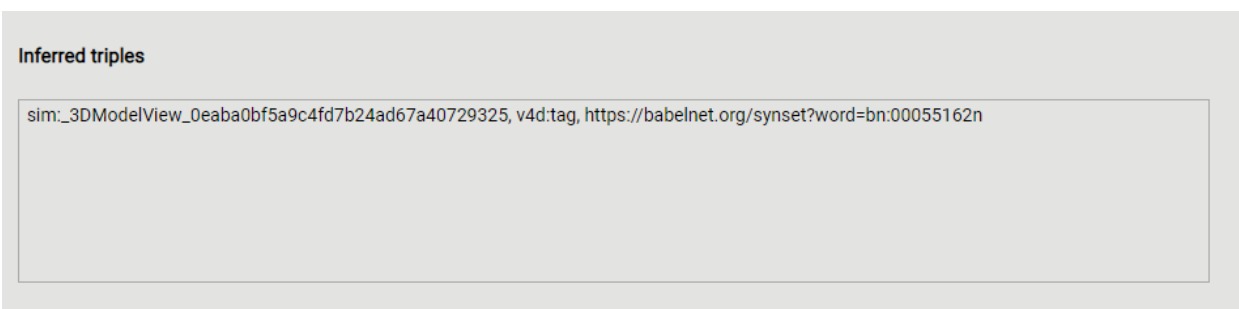


Figure 19: Inferred triples

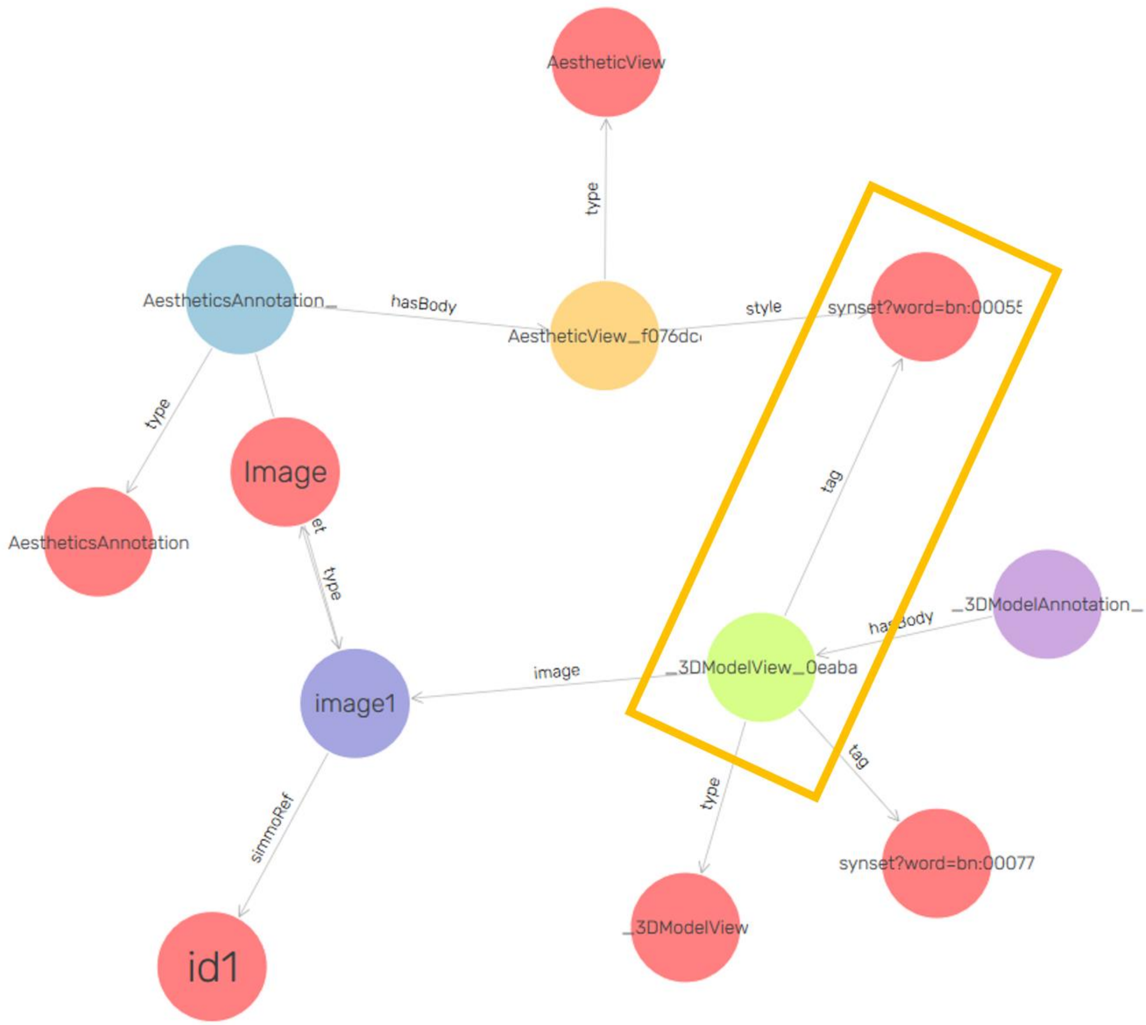


Figure 20: Materialised inferred relation in the V4Design graph

Queries can be defined to run over the V4Design Annotation Graph to get 3D models that match certain properties. For example, the SPARQL query in Figure 21 returns all the 3D models that have been annotated with the "minimalism" concept. It should be noted that by using resources instead of simple keywords allows us to support more complex queries. For example, in BabelNet the resource of "reductivism" has the same id with "minimalism". Therefore, a search from the user with the keyword "reductivism" would return the same 3D model, provided that text analysis on the user input would be able to assign the same BabelNet resource to the search parameter.

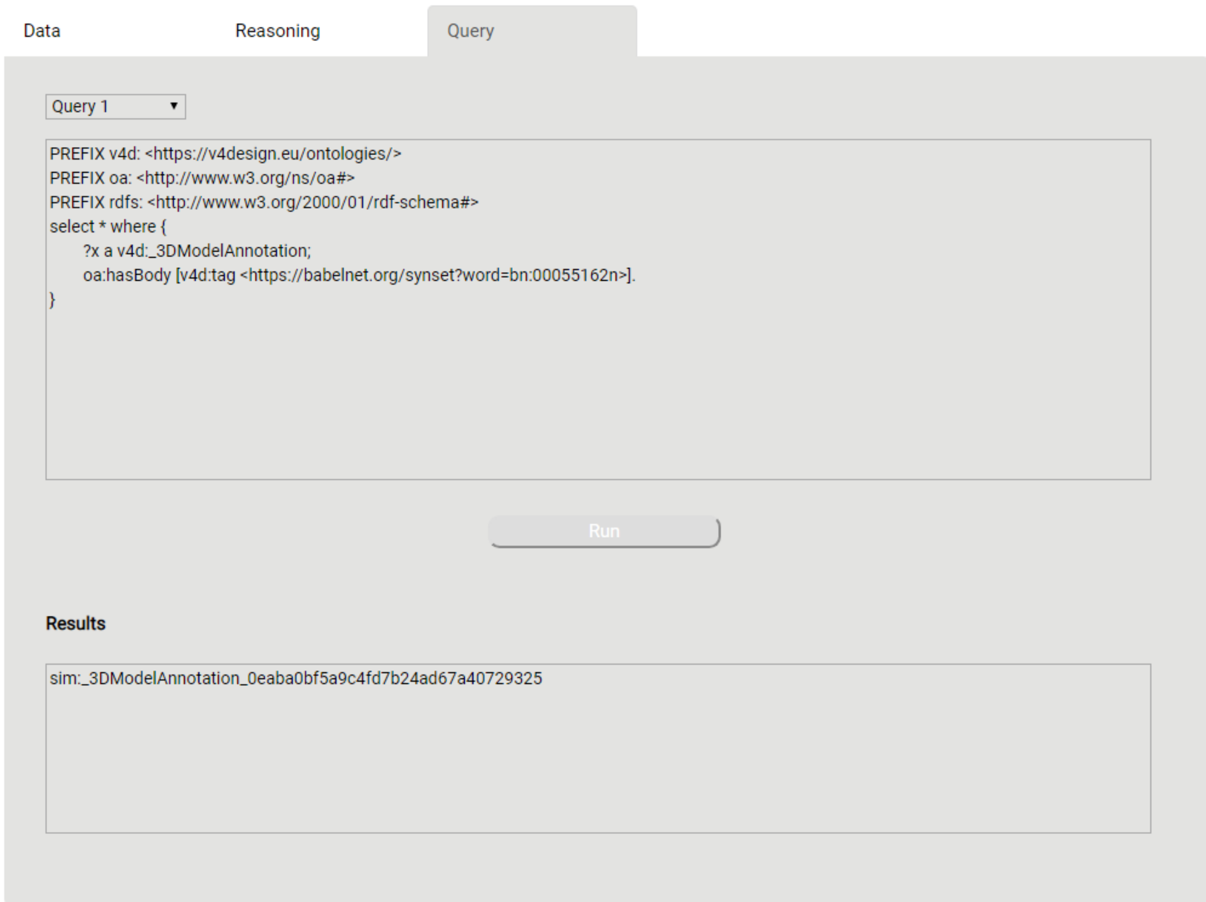


Figure 21: Example query

2.2.7 **Spatio-Temporal Building and Object Localization (STBOL)**

Spatio-Temporal building and object localization in images and video frames service detects whether an image or video contains a building, object or a painting and then semantically segments it in a spatio-temporally manner in order to localize the spatial elements of the buildings (i.e. type of window, door, roof, decoration, facade, etc.) and the surrounding area.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 14: Corresponding functional requirements

TR NB	**Description**	**Function**	**Function performed**
TR_OL_1	provide locations of buildings and objects in an image or a video	BOL	Scene recognition on the image or video frame: - Define whether a video contains data of interest (i.e. building, object) and define its location in the video - Define whether an image contains a building, object, painting

TR_OL_2	provide binary masks of the buildings and objects which are detected	STBOL	Semantic segmentation on the provided image/ video: - Segments the video in a spatio-temporal manner so as to extract masks of interest - Segments the image in a spatial manner to extract the masks of interest

The end-user will give to the system images or videos and it will get masks of video frames with tagged regions that include buildings, basic structural elements and building surroundings, which will then be given to the 3D-reconstruction module so as to incorporate the extracted tags to its 3D models. This is explained in the following diagram.

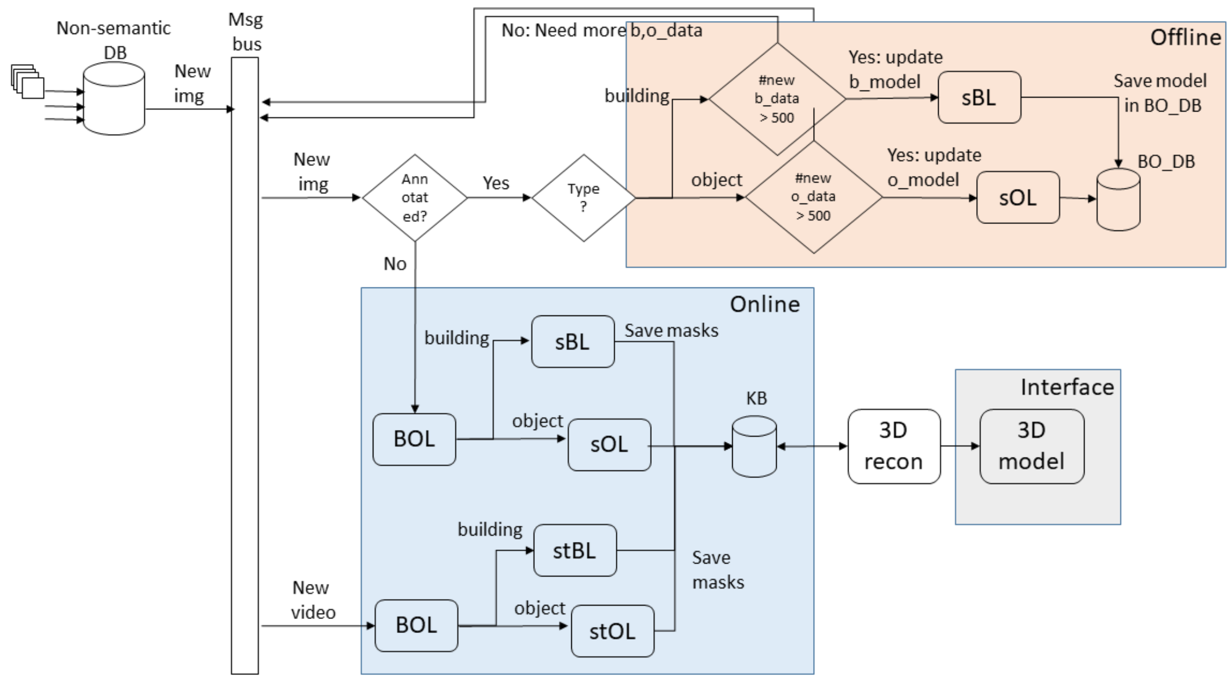


Figure 22: STBOL process diagram

An input example for the STBOL would be an image like the following:

The corresponding output:

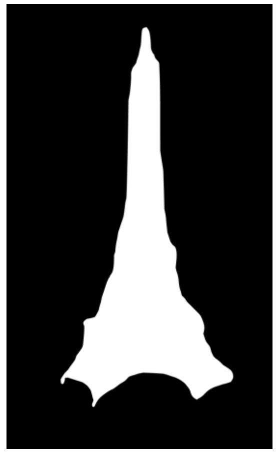


Table 15: Output example of Object Localization

{tower} Storage format: h5 Data schema: {alley;amphitheater;apartment_building;aqueduct;arcade;arch;archaelogical_excavation;archive;auditorium;balcony;barn;barndoor;bazaar;beach_house;boathouse;bridge;building_facade;bus_station;campus;castle;catacomb;cemetery;church;construction_site;corridor;dam;department_store;downtown;gas_station;general_store;gift_shop;harbor;hospital;hotel;house;industrial_area;inn;lighthouse;mansion;manufactured_home;mosque;motel;museum;natural_history_museum;oast_house;palace;parking_lot;pavilion;playground;restaurant;schoolhouse;stadium;supermarket;temple;tower;train_station;tree_house;wind_farm;windmill;yard}

Another Input example:

The corresponding output:

Table 16: Output example of Object Localization

{table, chair, etc.}
Type: DCNN-model (.h5)
Allowed values: {sofa; table; chair; lamp; mug; etc.}

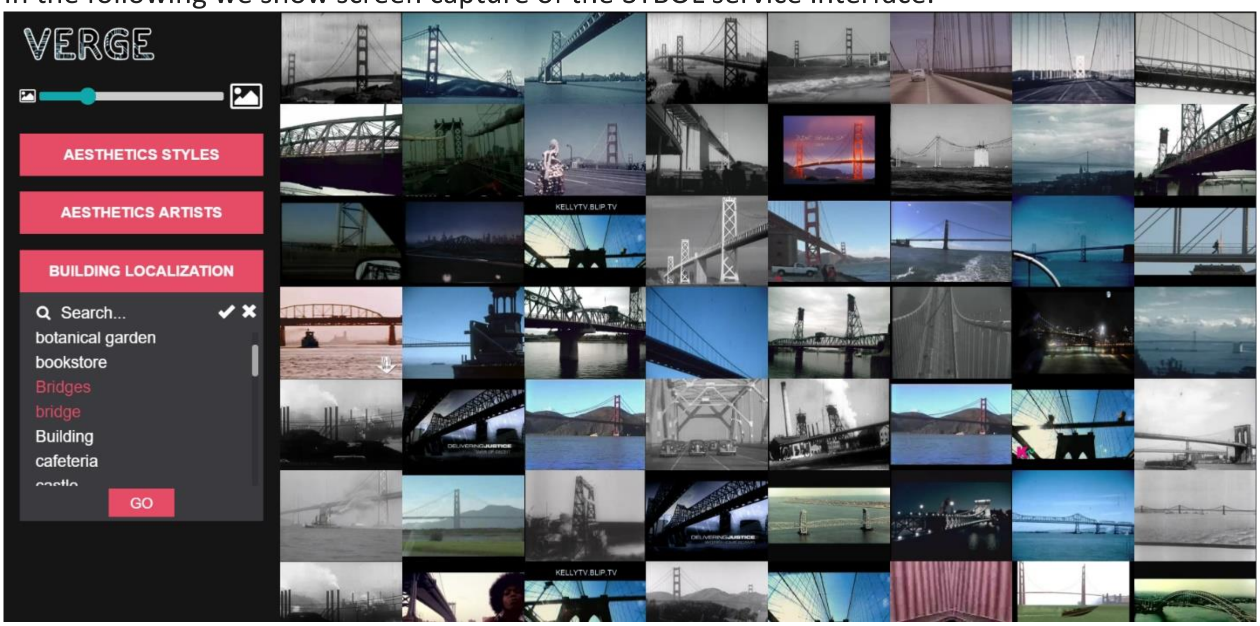
The development roadmap for this service is described in the following:

Operational prototype [M12]: Initial version of the basic STBOL component is released and integrated in the platform and message bus.

1st prototype [M20]: The basic version of the algorithm will be delivered and integrated in V4Design system.

2nd prototype [M34]: Advanced version of STBOL component will be deployed.

In the following we show screen capture of the STBOL service interface.



2.2.8 **3D Reconstruction**

The 3D Reconstruction service converts of input video and image data into 3D point clouds and meshes. Input data will be initially analysed to determine reconstruction suitability. The service will distinguish data suitable for multi multiple-view reconstruction (preferred method) and data suitable for single view reconstruction. The multiple-view reconstruction (MVR) pipeline will be providing intermediate results.

The technical requirements that this service aims to fulfil are summarized in the following table.

Table 17: Corresponding functional requirements

TR NB	Description	Function	Function performed
TR_3D_1	Extract and build a 3D model	Reconstruct	Build a 3D model from the collection of images or video frames

The 3D reconstruction module initially accepts data in the form of: 1) image batches and 2) video data.
In case where input consists of video data:

1. Initial frame extraction will begin
2. (not mandatory to start reconstruction) send extracted frames to visual analysis / localization tool for processing
3. Initiate reconstruction routine
4. If initial steps successful -> reconstruction possible. If not: further processing stops.
5. Preliminary reconstruction results may be made available in the form of a potree point-cloud.
6. Meshing processing starts

In case where input data consists of image batch:

- Check previous reconstruction if any of the images were successfully used in one of them:
- Yes? Update old reconstruction: add new images
- No? Initiate reconstruction routine
- See step 4 in the video data process above

This is explained in the following diagram.

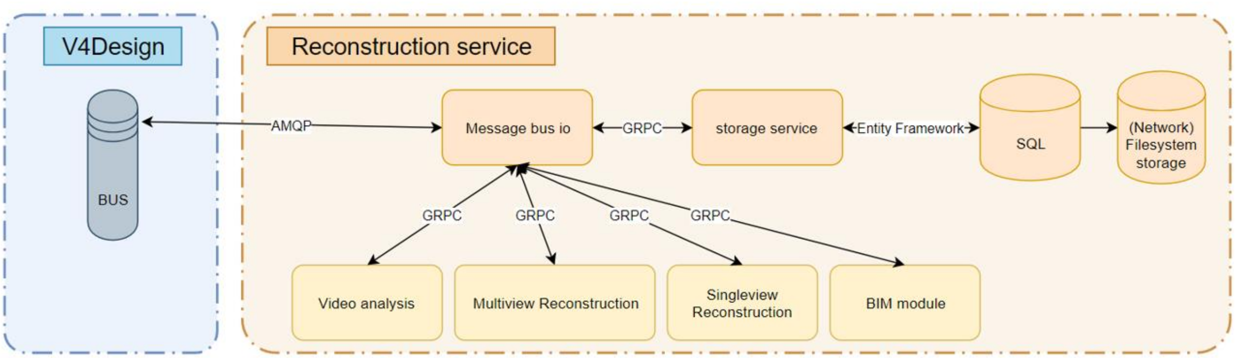


Figure 23: 3D Reconstruction process diagram

Currently for the Eiffel tower simulation example (see Crawler example), the following output layout was determined:

Table 18: Output example of 3D Reconstruction


```
{
 "reconstructions": [ {
 "reconstructionId": { "id": "string" },
 "reconstructionGroupId": { "id": "string" },
 "inputContent": [ { "sourceId": "string" } ],
 "usedContent": [ { "sourceId": "string" } ],
 "visualAnalysisTags": [ "string" ]
 } ]
}
```


Only 3 images displayed (currently using random UUIDs):


```
{
 "reconstructions": [ {
 "reconstructionId": { "id": " 984e4ec3-eadc-4483-af20-4a255e69ae0b" },
 "reconstructionGroupId": { "id": " 51c1b7f3-c66d-4adc-a922-738478f208b4" },
 "inputContent": [ { "sourceId": "f240773e-949a-4439-90d9-82a43d7dc201" },
 { "sourceId": "95607186-0c30-4ad8-9de0-735186b93f54" },
 { "sourceId": "00f77d86-3550-435f-a3c5-e8e7ab2e7eae " } ],
 "usedContent": [ { "sourceId": "f240773e-949a-4439-90d9-82a43d7dc201" },
 { "sourceId": "95607186-0c30-4ad8-9de0-735186b93f54" },
 { "sourceId": "00f77d86-3550-435f-a3c5-e8e7ab2e7eae " } ],
 "visualAnalysisTags": [ "Tower" ]
 } ]
}
```


The development roadmap for this service is described in the following:

Prototype [M12]: Initial reconstruction pipeline can be initiated (‘multiview reconstruction’ in the diagram above). Message bus component handles dummy messages (‘message bus io’).

First version [M18]: Further improvements on reconstruction pipeline: better frame extraction (‘video analysis’). Further output formats may be requested & processed (model decimation for example). Initial tests single view reconstruction on specific datasets.

Second version: [M24]: Enhancement and segmentation of reconstructions (‘BIM module’). Reconstruction feasibility test. Initial acquisition of BIM objects.

Third version [M30]: Final enhancements and updates.

2.3 Middleware modules and their development roadmaps

The V4Design platform middleware is composed of three different modules, being the message bus, the data storage and retrieval, and the API. Each performs a distinct role in supporting the services and user tools. In this section we discuss each of these modules separately.

2.3.1 The V4Design Message Bus

The message bus was initially introduced in D6.1 as a solution for implementing the selected architecture model. Accordingly, available off-the-shelf solutions for message bus were evaluated and assessed with respect to the general requirements of the architecture, and it was determined that an instance of Apache's ActiveMQ is among the most suitable solutions.

The main functionalities implemented by the message bus are the following:

A) Routing messages between components -- available in V1
B) Monitoring and control of message routing -- available in V1
C) Sequencing and queuing of messages -- available in V2
D) Resolving competition between communicating components -- Available in V2

All services and architecture modules depend on the proper functioning of the message bus. Messages are sent to the message bus through its open ports and are logged to keep track of traffic. A duplicate architecture is envisioned for the final deployment environment, where redundancy can be provided by using two message bus instances instead of one (see Figure 24).

Figure 25 shows the header structure of the messages sent through the message bus. A service can correlate messages, implicitly ask for a reply, typify messages (currently not used), delay and prioritise them, and control their scheduling and delivery. The message topics explained in section 2.1 have been implemented and tested.

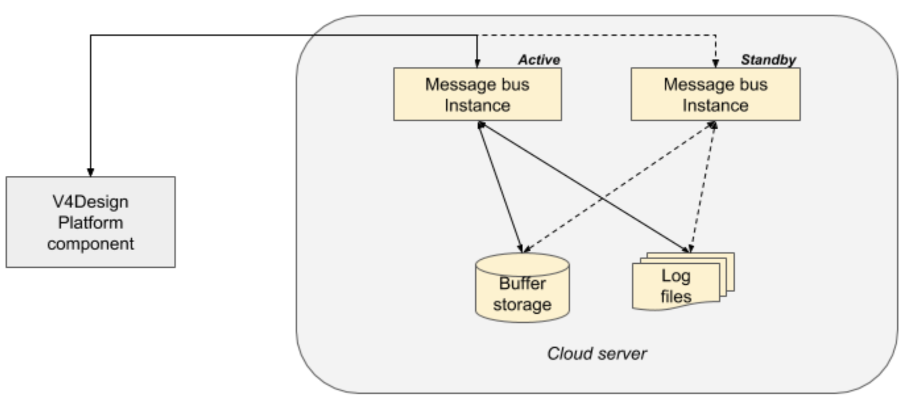


Figure 24: Redundant architecture of the message bus

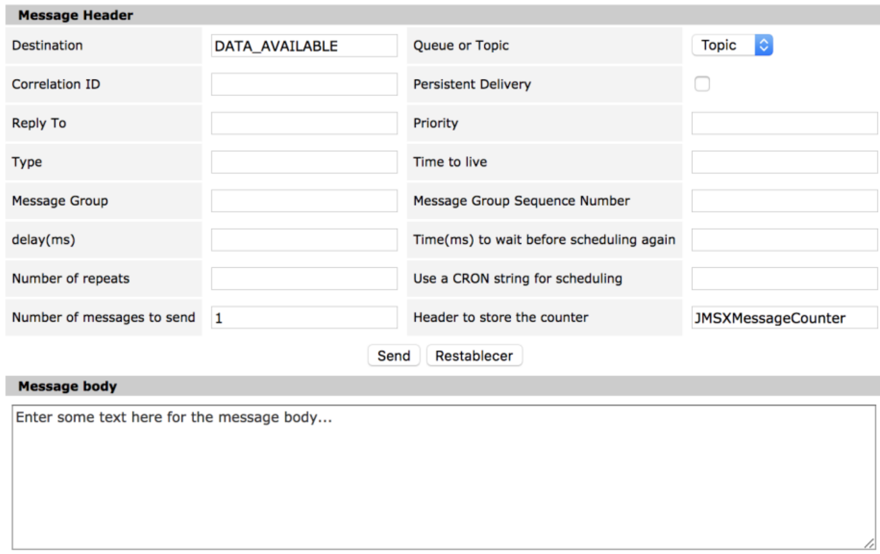


Figure 25: Implemented messages and structure of the message header

Figure 26: API Interface of the V4Design message bus

2.3.2 **V4Design REST API**

The V4Design REST API provides the functionality necessary for front-end applications to query and retrieve assets from the V4Design platform. The RESTful API provides specific calls to query through any number of metadata fields, such as asset type (3D model or image), asset date, asset quality, and any other relevant fields that would help to filter the available assets.

The front-end tools depend on this component to fulfil their usability. Basic functionality of the front-end tools are also included in the protocols of the component which can already be used by the Video Games Authoring tool and the Architecture Authoring tool.

The REST API will contain various functionalities:

1. User Authentication
2. User profiles

3. Comments, ratings by users on 3D models
4. Communication with message bus to get data

The backend of the REST API component of the platform will contain a database of the users to maintain authentication, profiles and ratings of the 3D models. Generally, the following actions may be performed:

1. User puts in an email and password for authentication using OAuth.
2. User authentication connects with the database management system sends the email/ username of the user. The Database Management replies with the password key of the user incase found or “Not Fount” incase not found.
3. The Database Management checks the database for the Queries.
4. The session management creates a session key and sends it to the database to be saved.
5. The user is also sent the Session Key.
6. With 1., the user can ask for a specific assets or list of assets, the access management first checks if the user has access to all the assets and filters them in case of a negative response (no)
7. The user is sent a proxy “URL” of internal “URI” of the asset.
8. The request Management sends the “URI”s for proxying using proxy management.
9. The request management sends messages to the Message bus to initiate commands and the Database for getting the assets.

All the action (1-9) are explained in the figure below (Figure 27).

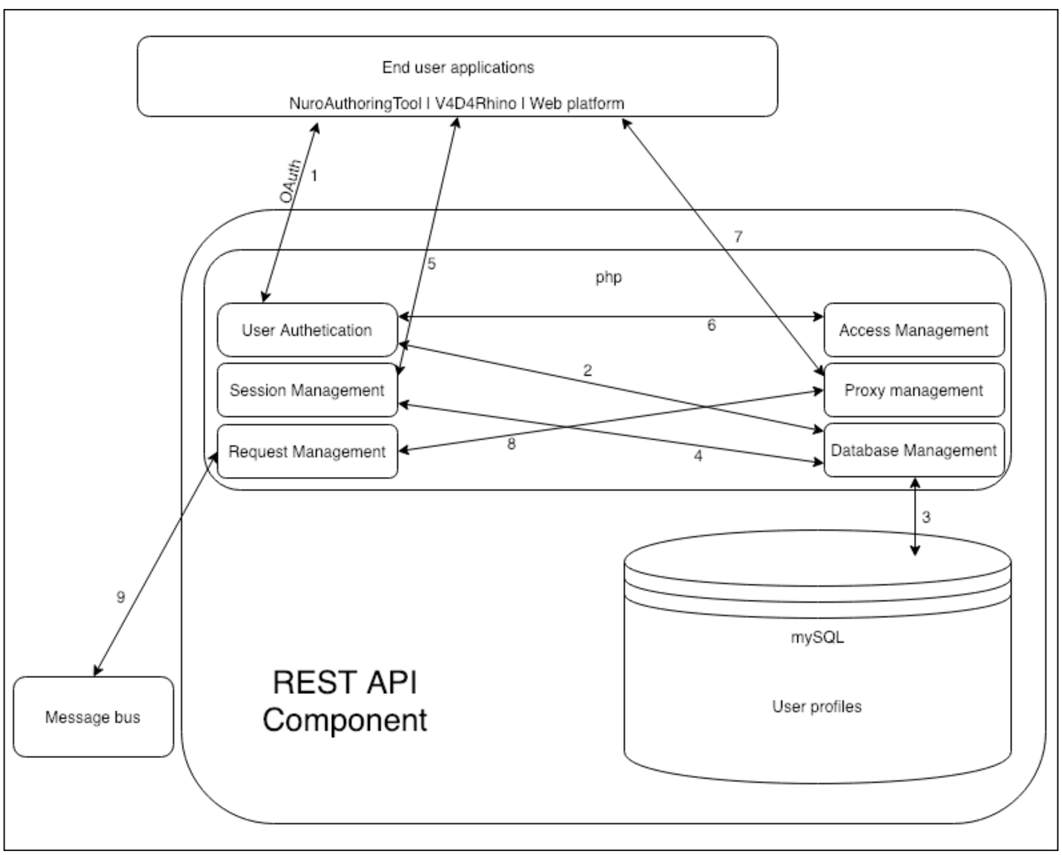


Figure 27: conceptual design of the API

2.3.3 Data Storage and Retrieval

The Data Storage and Retrieval is responsible for handling any data manipulation action that is needed for the V4Design components. It receives requests from the components and either connects to a folder containing static files or directs the request to a database API (e.g. MongoDB API). The interactions made through the Data Storage module are outlined in the following figure.

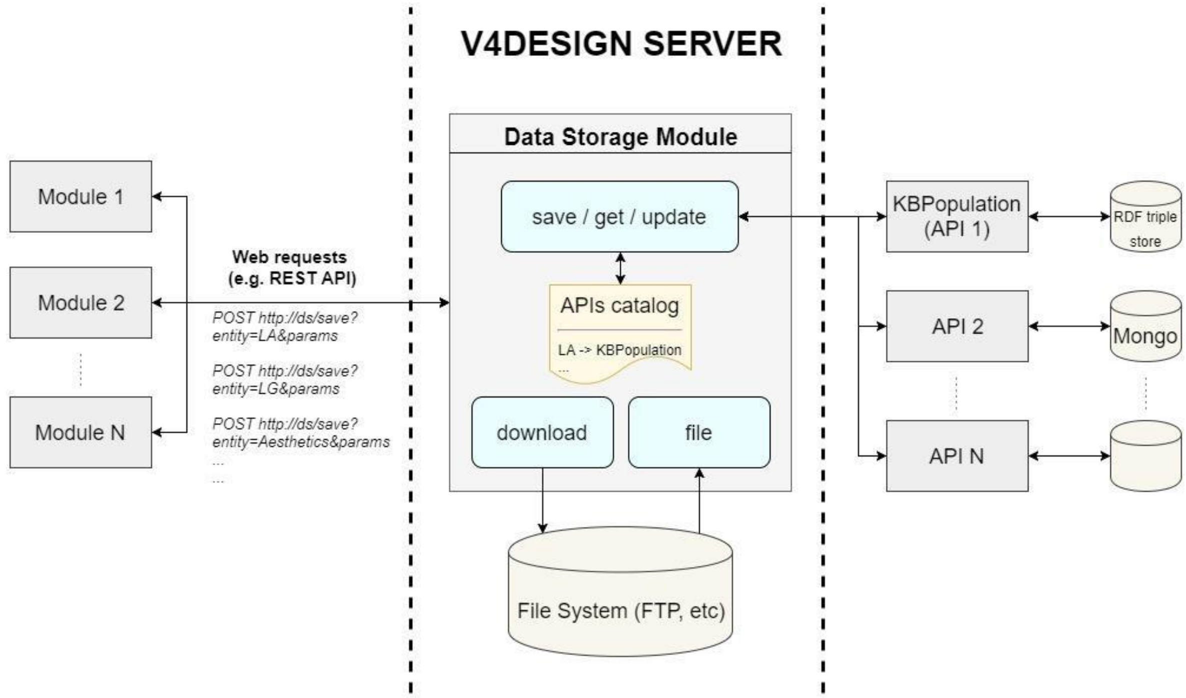


Figure 28: Interactions made through the Data Storage module

The data storage and retrieval system are actually composed of three different storages, each specialized in a specific type of data. The first storage is called the SIMMO Database and hosts the raw data or assets acquired by the crawler and the wrapper from external sources. The second storage is the Knowledge Base where all the output or results generated by the different services, except for the 3D Reconstruction service, are stored. The 3D objects generated by the platform are stored in the third storage, which is conveniently called the 3D Database. Therefore, in order to retrieve the data related to the processes that Services would like to perform, it is sometimes necessary to execute more than one GET request, each on a specific storage in the data storage and retrieval system. For instance, the Language Analysis service, which only processes raw data, will get its data from the SIMMO Database, and will store its output on the knowledge Base. Consequently, the Language Generation service will get the raw data from the SIMMO Database, and the corresponding data generated by the Language Analysis from the Knowledge Base. These three storages and the objects they host are illustrated in the following figure.

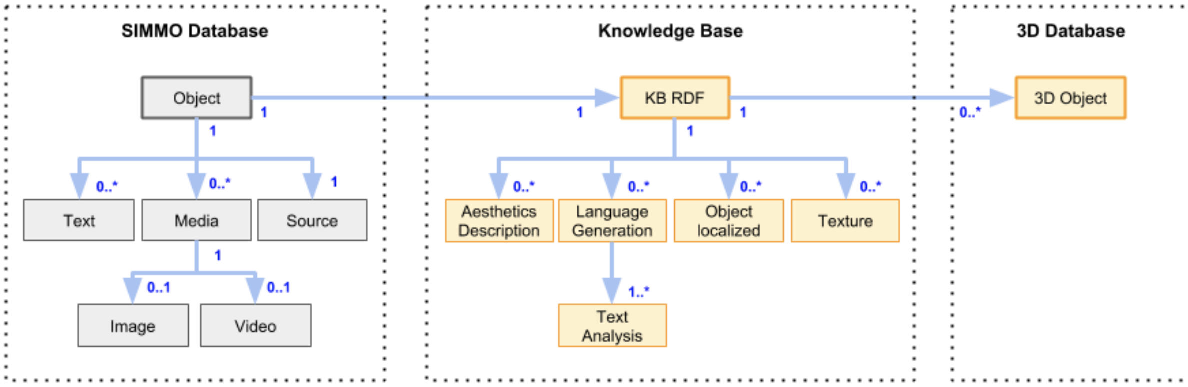


Figure 29: The Data Storage and Retrieval: three storages and the objects they host.

The basic functionalities that will be implemented by the Data Storage and Retrieval module can be summarized in the following manner:

Table 19: Functionalities implemented by the Data Storage and Retrieval

Function	Description	Function performed
Data push	Data Storing	Sending and storing of resource(s) to a target database.
Data update	Data modification	Change of an already existing resource in a target database.
Data pull	Data retrieval	Retrieval of resource(s) from a target database.

The development of the Data Storage and Retrieval module will follow the following roadmap:

- Operational prototype [M12]: Implementation of basic functionality.
- 1st prototype [M18]: Integration of all the developed database solutions into the Data Storage.
- 2nd prototype [M24]: Update of the web methods to support more data manipulation functions.
- Final prototype [M30]: Final updates and optimizations to the module.

In a scenario where we chose to retrieve a webpage from the SIMMO database using an id the returned JSON is the one below (JSON format):


```
{
 "annotations": [ {
 "metadata": { "Operator": "Cámara Municipal de Silves",
 "Type": "Castle",
 "Owner": "Portuguese Republic",
 "Built": "c. 201 BCE",
 "Coordinates": " 37°11’27.56”N 8°26’16.46”W ",
 "Materials": "Taipa, Silves Sandstone, Masonry, Wood",
 "Open to the public": "Public" }
```

}],
"id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"url": "http://en.wikipedia.org/wiki/Castle_of_Silves",
"crawlDate": 1541596601282,

"items": [{ "id": "11030bf0-4f0f-4baa-8119-8fd5b36c5764",
"type": "TEXT",
"parent": [{ "id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"type": "gr.iti.mklab.simmo.core.documents.Webpage" }],
"content": "The Castle of Silves is a castle in the civil parish of Silves in the municipality of Silves in the Portuguese Algarve ... that includes foundations in dirt, a stone staircase (with a single on one flight), a spacious living room with the remains of a vaulted ceiling, olive oil press and pesto.",
"textType": "TXT" },
{ "id": "d323cfb6-ee4c-4c60-8282-8089a4c7ff12",
"type": "TEXT",
"parent": [{ "id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"type": "gr.iti.mklab.simmo.core.documents.Webpage" }],
"content": "<!doctype html>\n<html class=\"client-nojs\" lang=\"en\" dir=\"ltr\">\n <head> \n <meta charset=\"UTF-8\"> \n <title>Castle of Silves - Wikipedia</title> ... </html>",
"textType": "HTML" },
{ "id": "fffc053b-f0f6-47aa-897b-513a71a0a956",
"url": "http://en.wikipedia.org/wiki/File:SanchoI-SilvesCastle.jpg",
"type": "IMAGE",
"parent": [{ "id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"type": "gr.iti.mklab.simmo.core.documents.Webpage" }],
"thumbnail": "//upload.wikimedia.org/wikipedia/commons/thumb/a/ac/SanchoI-SilvesCastle.jpg/235px-SanchoI-SilvesCastle.jpg",
"alternateText": "A statute of Sancho I of Portugal whose forces, supported by an even stronger Crusader army, conquered the citadel of Silves in 1189" },
{ "id": "c4363fde-c2c5-45a8-acb0-5e4d31af8e96",
"url": "http://en.wikipedia.org/wiki/File:Castelo_de_Silves_(6113330514).jpg",
"type": "IMAGE",
"parent": [{ "id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"type": "gr.iti.mklab.simmo.core.documents.Webpage" }],
"thumbnail":
"//upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Castelo_de_Silves_%286113330514%29.jpg/235px-Castelo_de_Silves_%286113330514%29.jpg",
"alternateText": "The imposing citadel as seen from below in the surrounding district of Silves" },
{ "id": "c0d3760b-b6ab-4ea3-b0b1-b60ccf6d03bb",
"url": "http://en.wikipedia.org/wiki/File:Castelo_de_Silves.26-04-18.jpg",
"type": "IMAGE",
"parent": [{ "id": "203daa5e-dbef-44a0-92ab-3b46c600ea17",
"type": "gr.iti.mklab.simmo.core.documents.Webpage" }],
"thumbnail": "//upload.wikimedia.org/wikipedia/commons/thumb/7/78/Castelo_de_Silves.26-04-18.jpg/250px-Castelo_de_Silves.26-04-18.jpg",
"alternateText": "Panoramic view of Silves castle" }]
}

2.4 Content Extraction Pipeline

The content extraction pipeline in the process by which the platform extracts raw data from data sources, and creates the assets envisioned in V4Design. It starts with the acquisition of new data objects and ends with the user consumption of newly-created assets. This pipeline is the primary process supported by the integrated architecture, and therefore the platform's backend (comprised of the services and middleware) is conceived and developed in a manner that optimizes performance from this pipeline's perspective.

In previous sections, we have discussed the platform data management policy according to which a centralized data warehouse has been conceived and created. This warehouse, referred to as Data Storage and Retrieval system, acts as a data hub for all the platform components.

First, the crawler acquires new data and stores it in the data storage and retrieval and broadcasts a "new data" message through the message bus. The message contains the IDs of the newly acquired data objects. The services can use these IDs to read the data directly from the data storage and retrieval with a "get request". Upon receiving this message, two services being the Object Localization and the Language Analysis immediately retrieve the new data objects and process them in parallel, storing their outputs in the data storage and retrieval. Subsequently, tier 2 services retrieve the raw data objects alongside with the output of tier 1 services, and process them in parallel, similarly storing their outputs in the data storage and retrieval. Then, the KB Population reads the output of Tier 2 services and update the reasoning iteratively. At this point, after the completion of all the service processes, the data is ready to be served to the user tools, which access it through query and get requests.

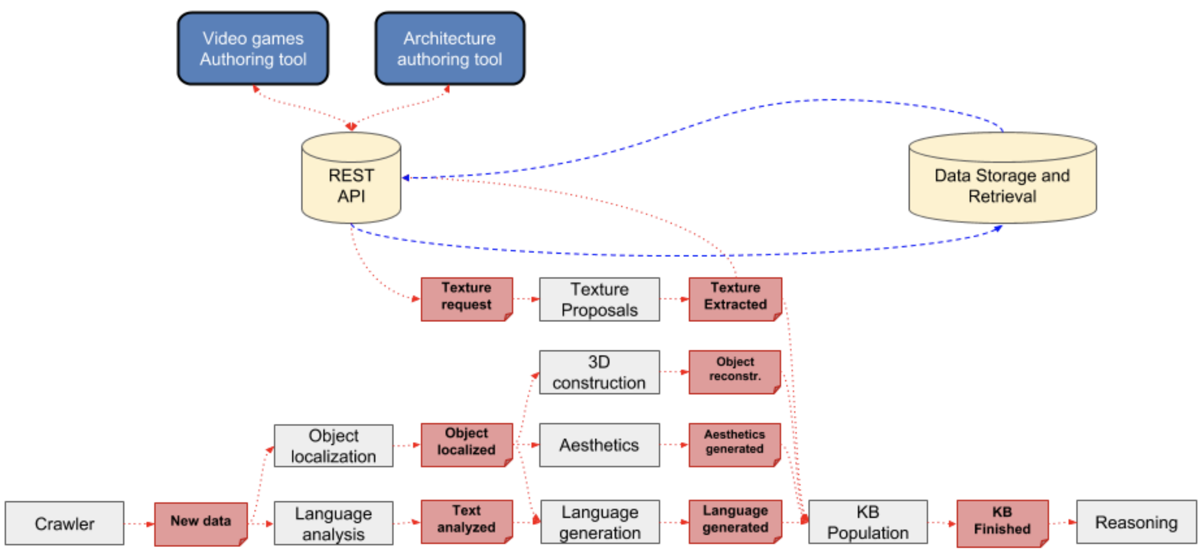


Figure 30: The platform's content extraction pipeline

This pipeline has possibly more than one ramification, which will be evident in the next development cycles and upon the user-driven evaluation of the first prototype of the platform. One ramification relates to a user-driven request for texture extraction, or 3D reconstruction. Pragmatically, such ramifications will be gradually eliminated and transferred to the tools, liberating the platform's backend from servicing user requests outside the realm of the content extraction pipeline.

One such ramification is the user request for extracting a specific texture based on user-defined parameters. During the second cycle of development, an attempt will be made to translate such functionalities, which currently are supported via petitions through the message bus, to the user tools.

3 PROTOTYPE APPLICATIONS

In this section, we introduce the visual demonstrations developed as part of the operational prototype for the platform. First, we discuss the message system visualization, a simulation that shows how the message bus implements the communication model of the platform and its cycle. Then, we discuss the authoring tool for architect describing its main functionalities and processes. Finally, we discuss the authoring tool for video games in a similar manner.

3.1 Message System Visualization

The V4Design message bus is the centre of its architecture, by which all modules connect to the each other and synchronize their execution (a model previously referred to as the platform cycle). Components connect to the message bus in an organized manner, ordered by a list of predefined topics, each addressing a specific platform concern.

These topics are presented in the following table.

Table 20: Message topics for V4Design platform components

Topic ID	Senders	Receivers
DATA_AVAILABLE	Crawler, Wrapper	Language Analysis, Object Localization
TEXT_ANALYZED	Language Analysis	Language Generation
OBJECT_LOCALIZED	Objected Localization	Aesthetics, 3D Reconstruction
OBJECT_RECONSTRUCTED	3D Reconstruction	KB Population
AESTHETICS_GENERATED	Aesthetics	KB Population
LANGUAGE_GENERATED	Language generation	KB Population
KB_FINISHED	KB Population	Reasoning
REASONING_FINISHED	Reasoning	None (at the moment)
TEXTURE_REQUESTED	API	Texture Proposals
TEXTURE_EXTRACTED	Texture Proposals	KB Population

In order to demonstrate and validate the proper functioning of the message bus, a simulation of the platform cycle has been implemented. Apart from showcasing how the message bus orchestrates the platform cycle, the simulation serves to analyse the current integration design, and improve it in order to define the architecture of the upcoming first version of the platform.

Among the subjects currently evolving is the integration with the Data Storage and Retrieval system, by which services and other components are asked to push and pull data directly onto it, and not channel data storage and retrieval requests through the message bus. In addition, the integration of Texture Proposals and 3D Reconstruction is deemed to change in the near future because both components require special input configuration

Texture Proposals requires the user to delimit areas from which texture should be extracted. One approach to follow would centre on adding intelligence to the module instead of invoking user interaction and user-cantered decision making.

3D Reconstruction needs to identify sufficiently large sets of images representing a single object in order to complete a 3D reconstruction of it. The current platform cycle and data management process do not guarantee a fruitful output for the 3D Reconstruction, which cannot immediately discern collection of images visualizing the same object. Different approaches for adapting the platform data stream to the input requirements of 3D Reconstruction are currently being considered, namely taking advantage of the semantic analysis components to identify related visual content automatically.

The following figure shows the version of the platform cycle that the simulation currently implements.

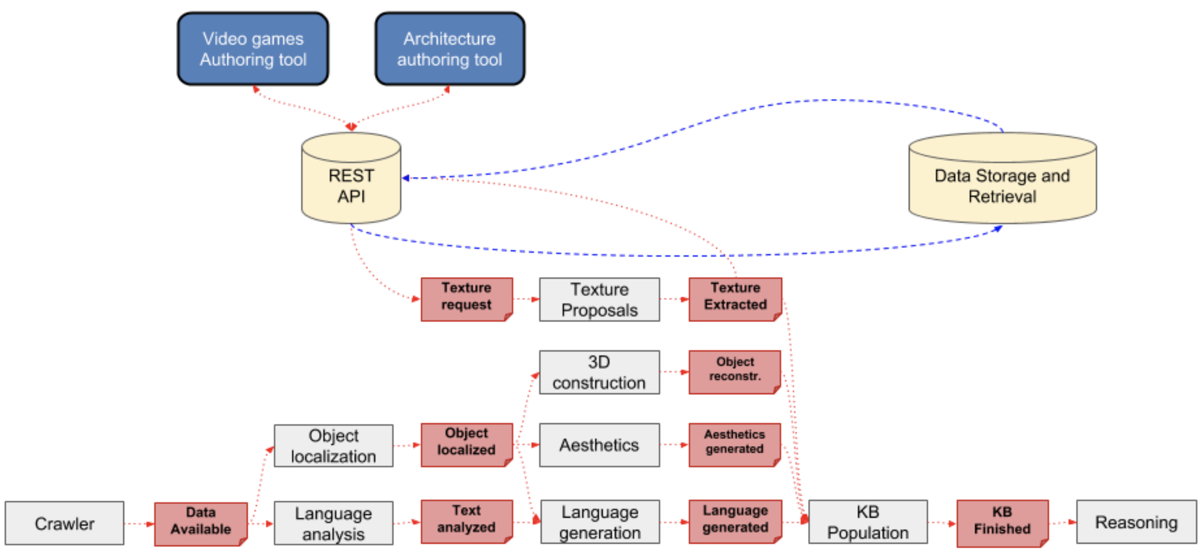


Figure 31: Platform communication model implemented by the demo

3.1.1 Description of the simulation

In order to implement the simulation, a communication client was developed in Java to connect to the message bus, create topics, and send and receive messages. The client runs as a Java servlet and can be deployed in any supporting web server. The client is composed of:

- A single producer of messages, able to send messages to any topic.
- A series of message consumers, each listening for a specific topic.

Apart from the client, a web simulation has been developed in HTML + JavaScript that launches and controls the client servlet. The web simulation implements the platform cycle as designed, starting by the arrival of new data, and ending with the generation of processed objects as previously argued.

The web simulation offers flexibility in implementing and adjusting the cycle and can be updated to reflect progress or to define a target model for the current version of the platform.

The interface of the simulation shows a diagram that visualizes how the V4Design architecture is integrated, connecting each of its services through message communication. Messages are sent and received by objects dynamically created to represent actual services. The user can interact with a single service (e.g. shutting it down, restarting, sending message), or can launch entire cycles to simulate specific cases.

The simulation actually sends and receives messages through the message bus, but currently does not connect any data or metadata to these messages, a feature that could be contemplated for its next version. Its interface visualizes the messages received upon reception, both as integrally and conceptually via arrows shown on the diagram. A small delay is programmed between the reception of a message and the emission of the corresponding response in order to simulate the processing delay caused by running a service.

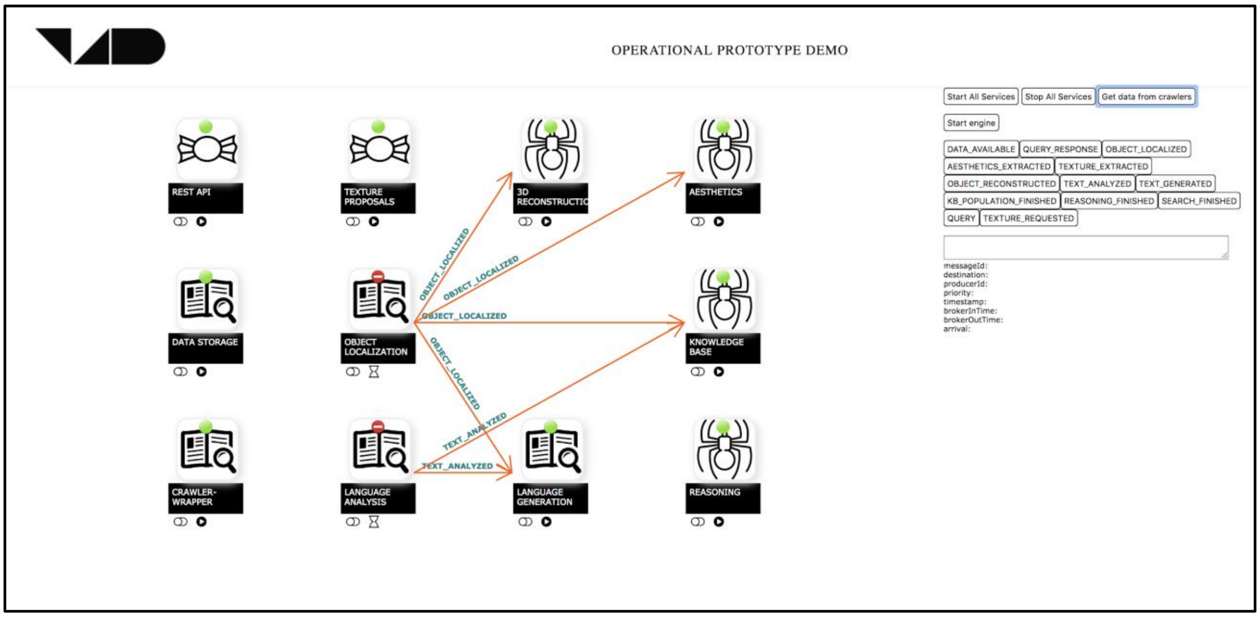


Figure 32: Interface of the simulation showing the cycle in motion.

3.1.2 **Testing the message bus implementation**

In order to proper monitor the functioning of the message bus and the exchange of messages, the execution of the platform cycle by the simulation can be observed through the message bus API interface as illustrated in the following figure. Tests show that the message bus is able to accommodate the cycle execution comfortably, and that this process can scale further. Tests did not yet address a continuous execution, which is a concern left for the next development cycle of the V4Design platform.

Name ↑	Number Of Consumers	Messages Enqueued	Messages Dequeued	Operations
'DATA_AVAILABLE'	0	0	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.MasterBroker	0	1	0	Send To Active Subscribers Active Producers Delete
AESTHETICS_EXTRACTED	0	0	0	Send To Active Subscribers Active Producers Delete
DATA_AVAILABLE	0	0	0	Send To Active Subscribers Active Producers Delete
KB_POPULATION_FINISHED	0	0	0	Send To Active Subscribers Active Producers Delete
OBJECT_LOCALIZED	0	0	0	Send To Active Subscribers Active Producers Delete
OBJECT_RECONSTRUCTED	0	0	0	Send To Active Subscribers Active Producers Delete
QUERY	0	0	0	Send To Active Subscribers Active Producers Delete
QUERY_RESPONSE	0	0	0	Send To Active Subscribers Active Producers Delete
REASONING_FINISHED	0	0	0	Send To Active Subscribers Active Producers Delete
SEARCH_FINISHED	0	0	0	Send To Active Subscribers Active Producers Delete
TEXT_ANALYZED	0	0	0	Send To Active Subscribers Active Producers Delete
TEXT_GENERATED	0	0	0	Send To Active Subscribers Active Producers Delete
TEXTURE_EXTRACTED	0	0	0	Send To Active Subscribers Active Producers Delete
TEXTURE_REQUESTED	0	0	0	Send To Active Subscribers Active Producers Delete

Figure 33: The message bus API interface for message monitoring

3.2 Authoring tool for architects (V4D4Rhino)

3.2.1 **Description**

The Authoring Tool for Architects is developed as a portal to the V4Design Asset Repository. Technically, this tool is developed as a plugin to the Rhinoceros 3D CAD and 3D modelling application.

The authoring tool developed by MCNEEL will allow the various functionalities including the following:

- The users will be able to directly import assets from the V4Design repository to the scene in Rhinoceros 3D
- The users will be able to analyse and manipulate the models imported from the V4Design Asset Repository
- The users will be able to create personalized asset libraries from the assets available in the V4Design Asset Repository

3.2.2 **User and Technical Requirements**

The current state of the authoring tool for architects attempts to fulfil some of the basic user requirements defined in deliverables D7.1 and D7.2, specifically, the requirements to be able to retrieve assets from the V4Design Asset Repository as well as to add these assets into a 3d modelling environment in order to be able to study and manipulate the 3d model. This functionality is exposed through a simple user interface. In the future, the functionality of the tool will be expanded to include search and filtering functionality and other enhancements required to fulfil the user requirements.

3.2.3 **Development Tools**

V4D4Rhino has been developed as a plug-in to the Rhinoceros 3D (Rhino) application. Beyond being a capable 3d modelling application, Rhino also comes with a host of APIs which 3rd party developers can use to develop custom functionality for Rhino. These API come in different programming languages (C++, .net C# and VB, Python, and VBScript), and are targeted to developers with potentially different objectives. While the Python and VBScript APIs are mainly for writing script extensions which are meant to be distributed as text files, the C++ and .net APIs allow for developers to compile their source code to a Dynamic Link Library (DLL) for distribution. By this DLL compilation mechanism that a developer can create a plug-in for Rhino in the format of an .rhp file. The .rhp file format is simple a DLL with a different extension. This format is understood by the Rhino application as the entry point for a Rhino plugin and contains the appropriate functions to instruct the Rhino application on what to do with the source code and any associated DLLs.

V4D4Rhino is currently being developed in the C# programming language through the .net API available for Rhino called RhinoCommon [4]. This API was chosen due to relative ease of use and widely available source code samples. The RhinoCommon API includes functionality for authoring Rhino plug-ins, functionality to interact with the current open Rhino file, as well as functionality to do geometry creation and operations, all of which are relevant to developing an authoring tool for architecture and design which meets the UR and HLUR gathered in previous deliverables D7.1 and D7.2.

The V4D4Rhino plug-in includes a user interface developed in ECMAScript (JavaScript), HTML 5, and CSS facilitated by the Vue.js framework.

3.2.4 **Development Plan**

The development of the V4D4Rhino tool features will be as follows:

- D6.4 [M18]: Connect V4D4Rhino to the V4Design Asset Repository and rework the data models to receive the actual metadata schemas generated by the V4Design back end. Additionally, the UI will be updated to accommodate all of this data. Ideally at this point the tool would address the user related data storage requirements, such as ‘liked’ assets, user comments, and model derivatives.
- D6.5 [M26]: Incorporate feedback from *D7.3 [M20] Evaluation of 1st prototype* into the tool. Address any user requirements which need to be integrated at this point.

- D6.6 [M34]: Incorporate feedback from *D7.4 [M28] Evaluation of 2nd prototype* into the tool. Address any issues of tool distribution that might arise throughout the development of the tool.

3.2.5 **UI / UX**

Currently under development is a tool which addresses the gathered user requirements for architects and designers. The objective for this tool is for it to be integrated into existing CAD applications where it can start to become part of architecture and design workflows. Technically, the tool is developed as a plug-in or add-on to the existing CAD application Rhinoceros 3D (Rhino) where it is presented to the user as a portal to the available assets in the V4Design Asset Repository (Figure 34).

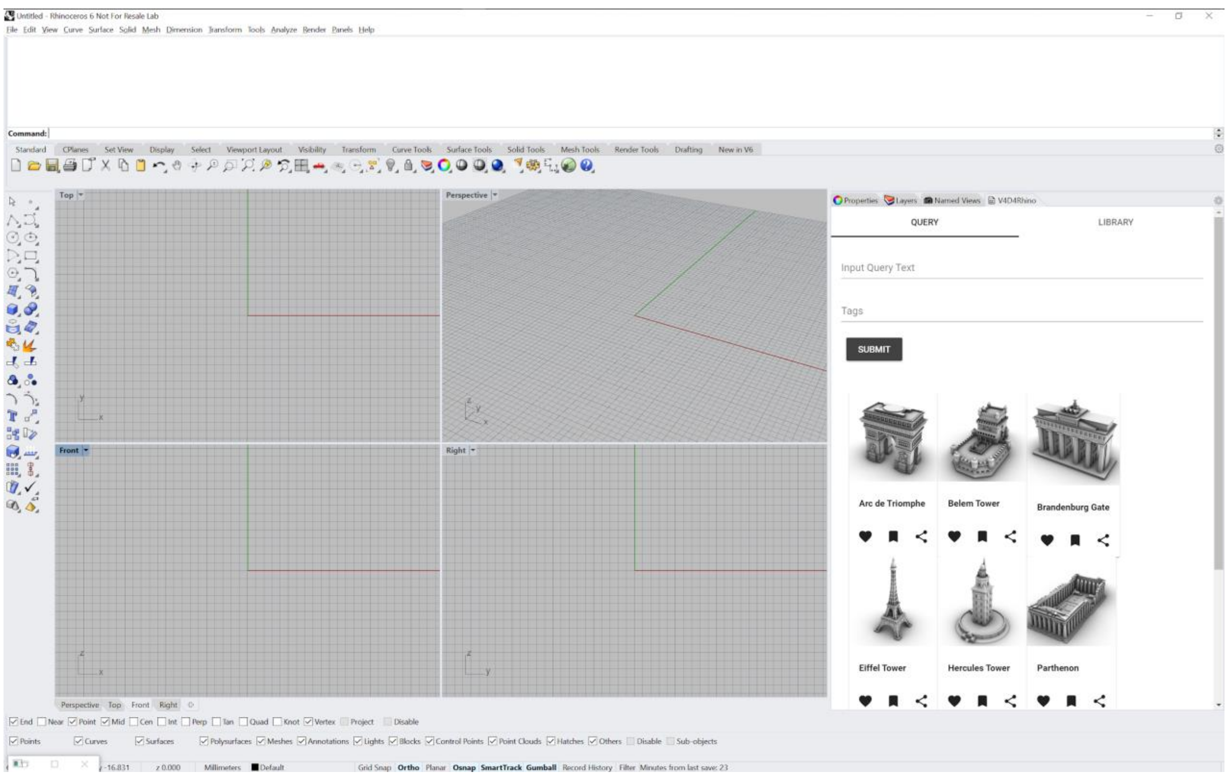


Figure 34: The V4D4Rhino plug-in Query Window within the Rhino3d application.

V4D4Rhino in its current development state exposes a query window for searching and filtering the available assets in the V4Design Asset Repository as well as a library window for organizing assets which might be of particular interest to the user. The query window currently allows for a text-based search in addition to a tag-based search. In subsequent versions, more specific search mechanisms will be implemented to further aid in filtering the results from the query.

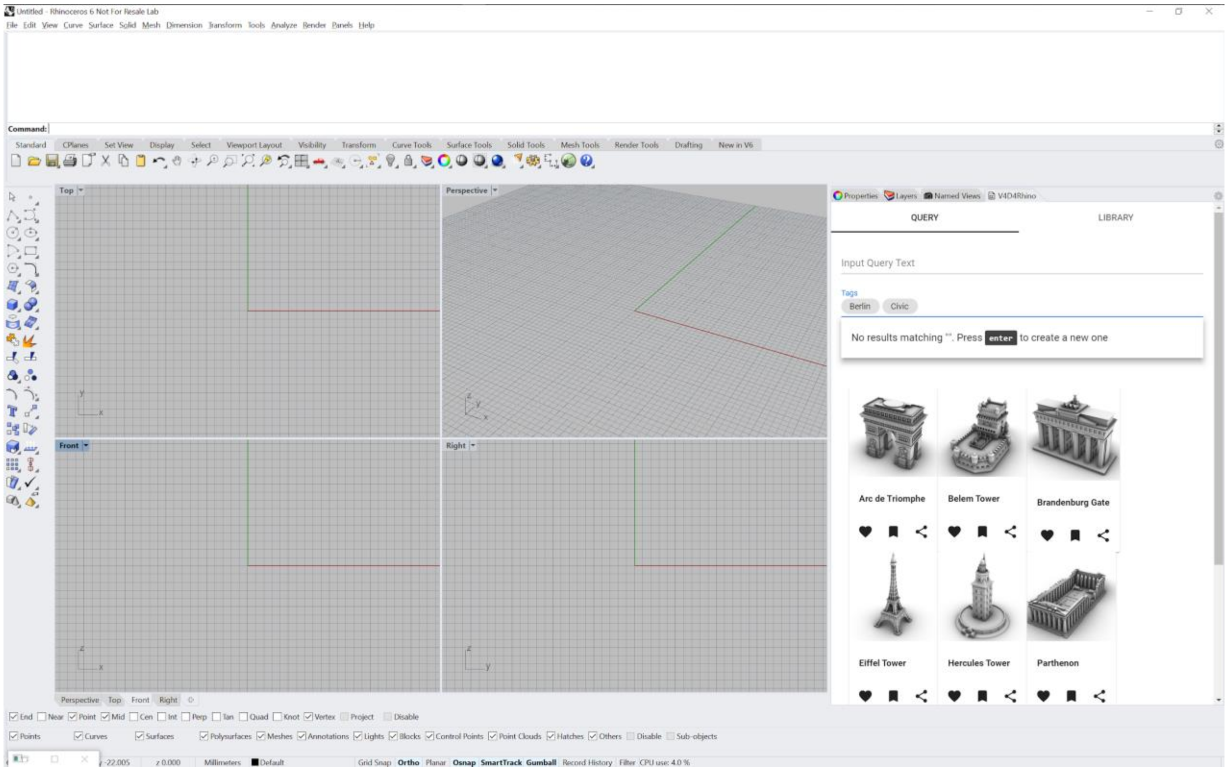


Figure 35: Text and tag-based query fields in V4D4Rhino

Once a query has been performed, the user is presented with a list of results, each showing a thumbnail of the asset, asset title, and some quick action buttons which allow the user to like, bookmark, or share the asset. The user can get further information for the asset by clicking one of the results. This opens an asset preview window which allows the user to view the details of the asset, including any relevant descriptions, user comments, asset popularity, and asset metadata (Figure 35).

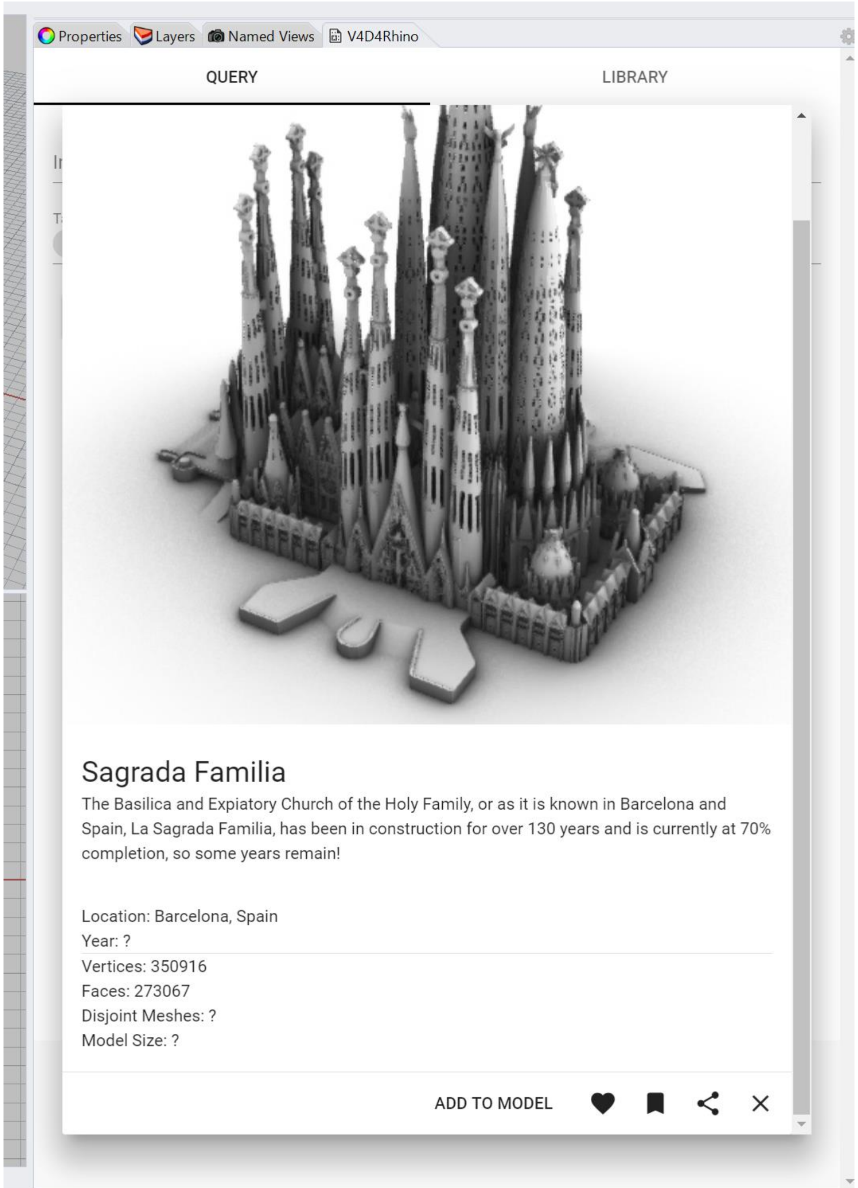


Figure 36: Asset detail window.

The library window of the V4D4Rhino plug-in shows any assets bookmarked by the user. This section could be further developed by creating user defined asset collections, where a user could bookmark a group of assets for different functions. For example, a user might be interested in retrieving assets for a specific project located in Athens, Greece. The user would create an asset collection with the asset results from a relevant query. These assets could then be differentiated from assets needed for other projects or concerns. Such collections could also be useful in sharing assets in a collaborative project environment, where several users are working together on a project.

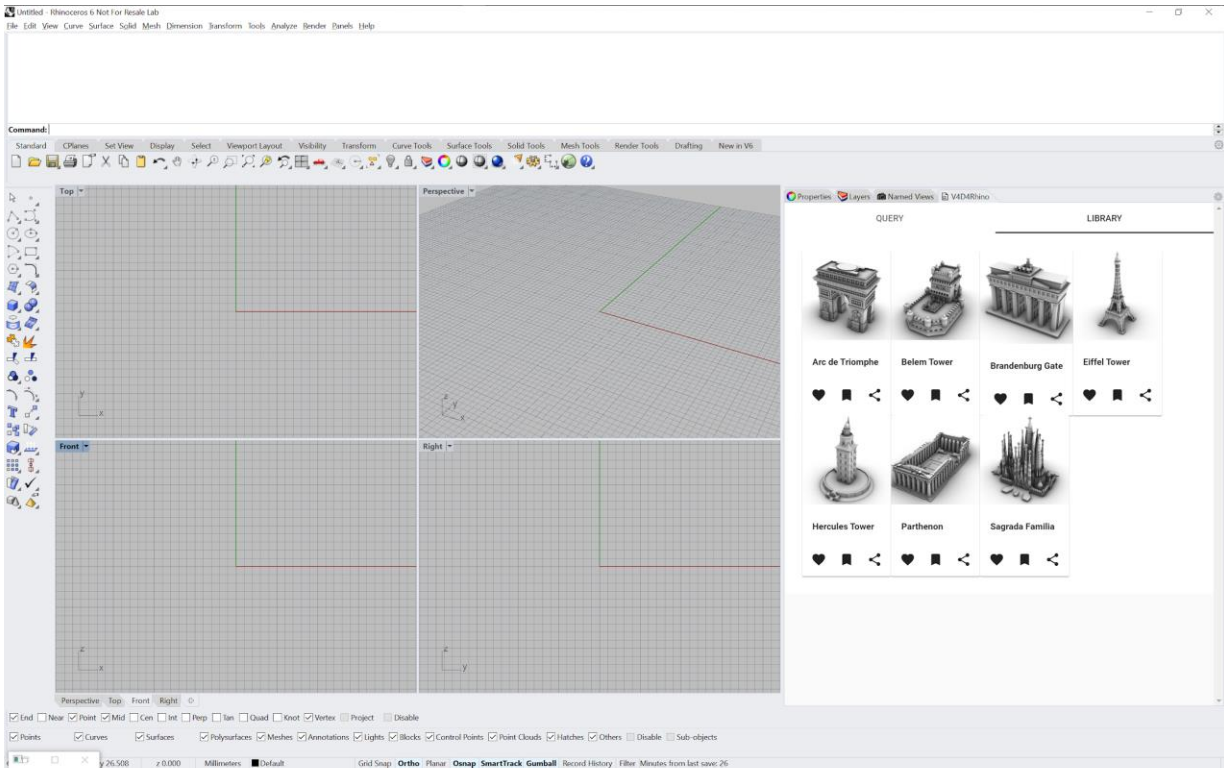


Figure 37: V4D4Rhino asset library

Besides retrieving assets from a remote repository, the V4D4Rhino plugin allows the retrieved models to be added to the Rhino modelling environment so that the user can then use the available tools to further manipulate the model. This functionality is exposed currently in the Asset Detail Window dialog via the “ADD TO MODEL” button. Once pressed, the asset 3D model is retrieved and added to the Rhino modelling environment.

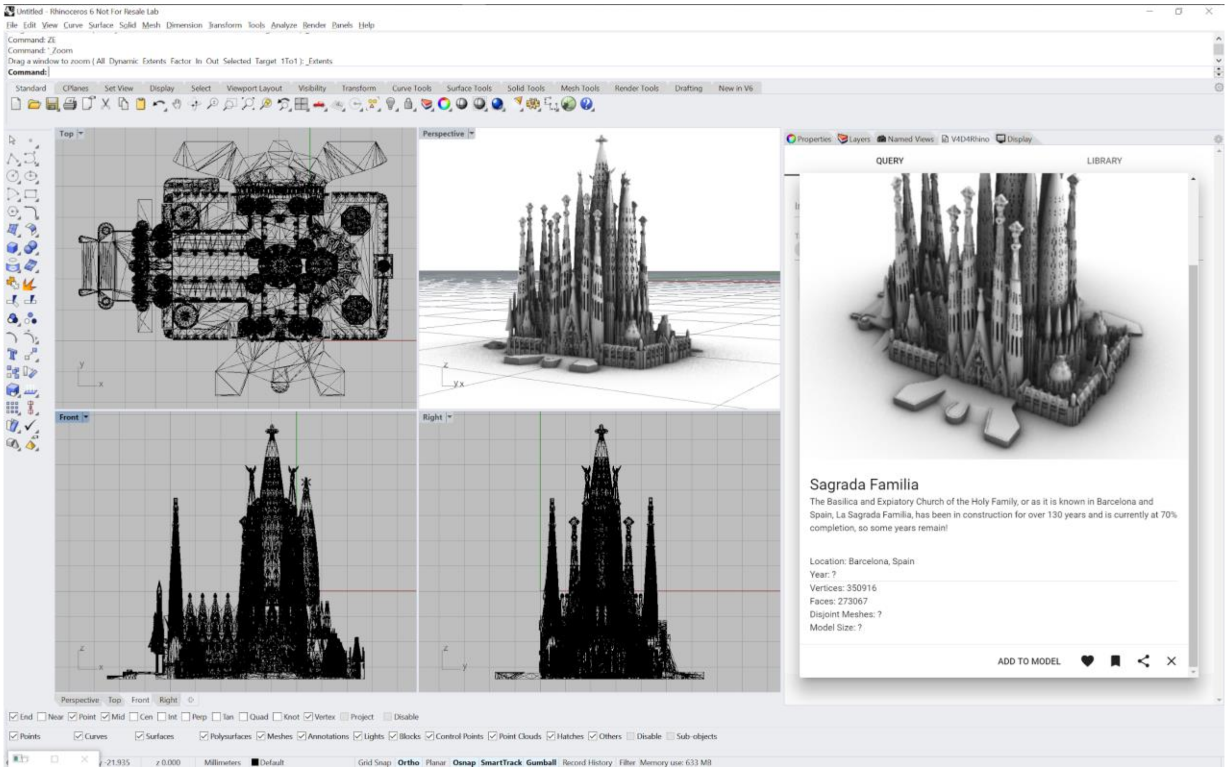


Figure 38: Model from remote repository added to Rhino.

Once the model has been retrieved and added to the Rhino modelling environment, users can start to analyse or manipulate the model with all the tools available in Rhino. For example, it might be of interest to an architect to study the interior of a retrieved asset to understand the relationships between open space and structural supports. In this case, the architect would use the "Clipping Plane" functionality in Rhino to occlude parts of the model and reveal the interior spaces. An example of this can be seen in Figure 38.

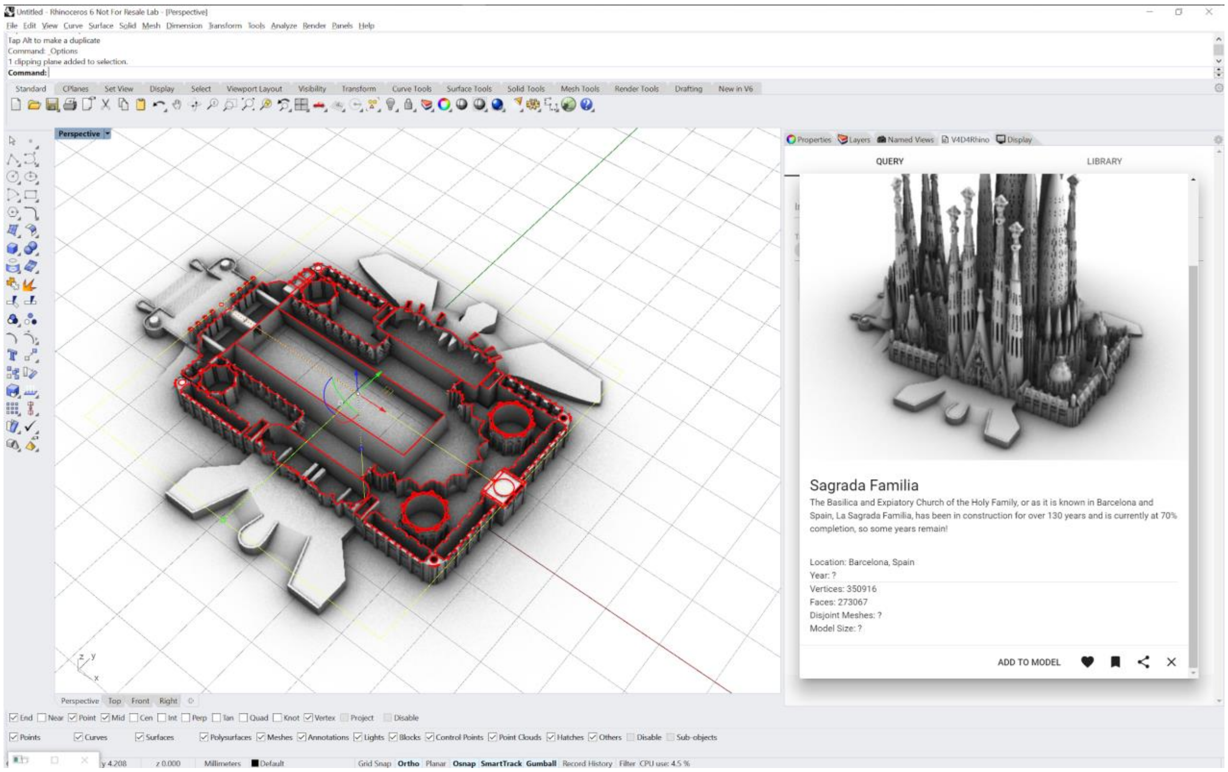


Figure 39: Showing retrieved model interior via the '"Clipping Plane" functionality in Rhino.

3.2.6 **Tool evaluation**

V4D4Rhino will be evaluated during the specific deliverables defined for tool evaluation:

- D7.3: Evaluation of the 1st prototype and updated user requirements [M20]
- D7.4: Evaluation of the 2nd prototype [M28]
- D7.5: Final system evaluation [M36]

Additionally, MCNEEL will host a user event where the tool will be given to users to try and evaluate.

Finally, MCNEEL will make the tool available via its distribution channels in order to allow to the existing Rhino3D users to give feedback on the plugin functionality.

3.2.7 **Tool exploitation (distribution, licensing, exploitation)**

The V4D4Rhino source code is currently developed under the open source MIT license. Compiled versions of the source code will also be released under the same license. These releases will be made available to Rhino users via two main distribution channels:

1. food4rhino.com: A website where McNeel and 3rd party developers publish Rhino plugins and resources.
2. Yak: A package manager (similar with Nuget.org or the Node Package Manager, npm) which ships with Rhino. Yak is currently under development, but some functionality is available in Rhino 6 (the current release). This package manager allows users to directly

search and install plugins from within Rhino, without having to go to a website like food4rhino.

Furthermore, the tool will be promoted via MCNEEL communication channels, including the company blog, domain specific email newsletters, and events.

3.3 Authoring tool for video games

3.3.1 Description

The authoring tool for video game developers is developed in order to facilitate game development procedure for people with preliminary knowledge of game development. The authoring tool plans to work on top of the Unity3D [3] game engine where users can create a basic environment using both the assets provided by V4design asset store and external assets.

The authoring tool developed by NURO will allow the various functionalities including the following:

- The users will be able to directly import assets from the V4Design repository to the scene in Unity3D.
- The users will be able to edit the environments in VR using the authoring tool
- The users will be able to add questions to assets where the player of the game will have to answer the question to pass the asset in the game
- Import textures from the assets of V4design repository

3.3.2 User and Technical Requirements

The current version of the Authoring tool fulfils a great deal of the user and technical requirements stated in D6.2. The basic requirements fulfilled includes the ability to search, browse and import 3D models from V4Design repository into a scene of Unity3D, the ability to modify the asset, see a preview of the asset, see the metadata related to the asset, import a 3D Model into an environment in real-time while in Virtual Reality and export assets.

3.3.3 Development Tools

NuroAuthoringTool is being developed as a plug-in to Unity3D game engine. A game engine provides developers ability to easily create game. It is a software development environment specifically for creating video games. Unity3D provides game developers perfect rendering engine, physics engine, collision detection, scripting and other tools for development of games.

Plugins for Unity3D can be written using C# through .NET for managed plugins and can be added to the engine as a .dll file.

The NuroAuthoringTool can be used directly in VR to create, modify environments. To get assets directly into a VR environment using an API, the 3D models have to be wrapped as a unity3D file extension for the importing on the go. Currently the authoring tool uses static assets from the system to showcase in the VR editor for adding and modifying.

3.3.4 **Development Plan**

The development of the NuroAuthoringTool features will be as follows:

- D6.4 [M18]: Connect NuroAuthoringTool to the V4Design Asset Repository and rework the data models to receive the actual metadata schemas generated by the V4Design back end. Additionally, the UI will be updated to accommodate all of this data. The tool will include version 1 of all the necessary functionalities.
- D6.5 [M26]: Incorporate feedback from *D7.3 [M20] Evaluation of 1st prototype* into the tool. Address any new or updated user requirements which need to be integrated at this point.
- D6.6 [M34]: Incorporate feedback from *D7.4 [M28] Evaluation of 2nd prototype* into the tool. Address any issues of tool distribution that might arise throughout the development of the tool.

3.3.5 **UI / UX**

The UI/UX design was designed with keeping in mind the user requirements, usability guidelines and experience of NURO in designing user interfaces. Figure 40 represents the initial mock-ups designed and presented for the partners to get an perspective on the implementation plans and comment on features they would like to be implemented or not.

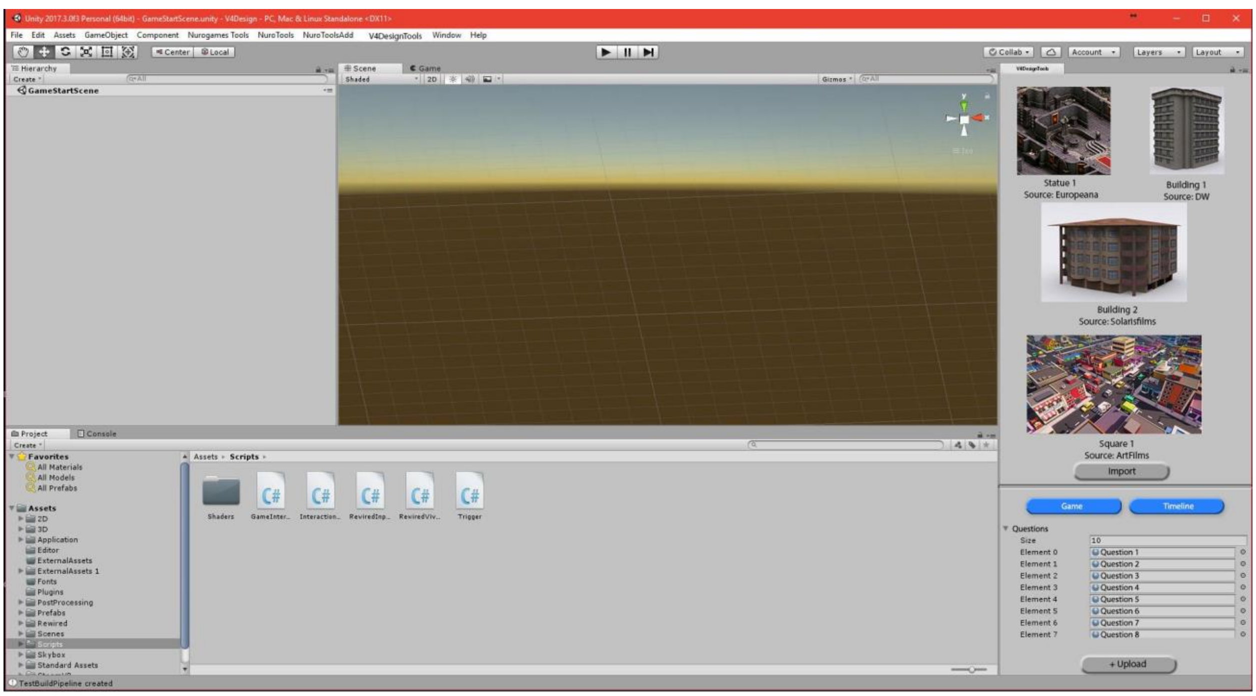


Figure 40: Initial Mock-up of the tool.

Following the initial mock-ups and the comments from the partners, an initial version was deployed with static data. The Figure 41 represents the first version of the NuroAuthoringTool with various display previews of the assets.

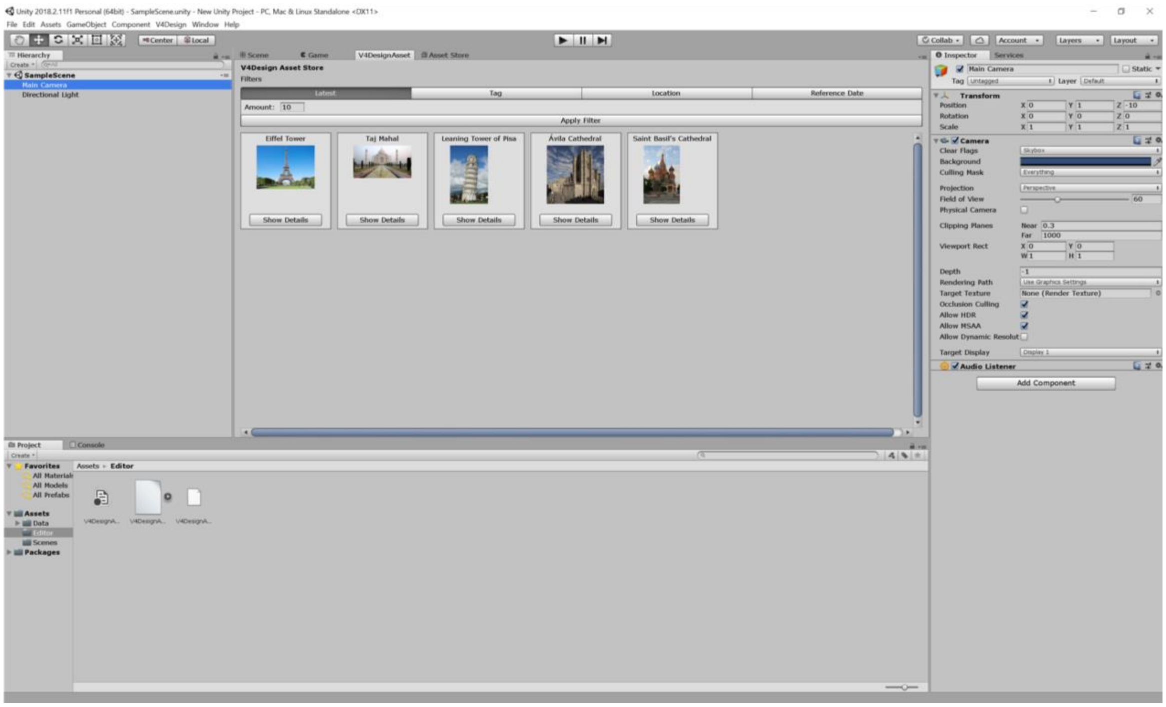


Figure 41: Initial Mock-up of the tool.

Figure 42 represents the UI of the tool when an asset is chosen. Various aspects are shown about the asset with the ability to bring it in to the scene while in the development mode of unity. The user can get more information and, in the future, would be able to trigger certain actions in the backend such as extracting textures.

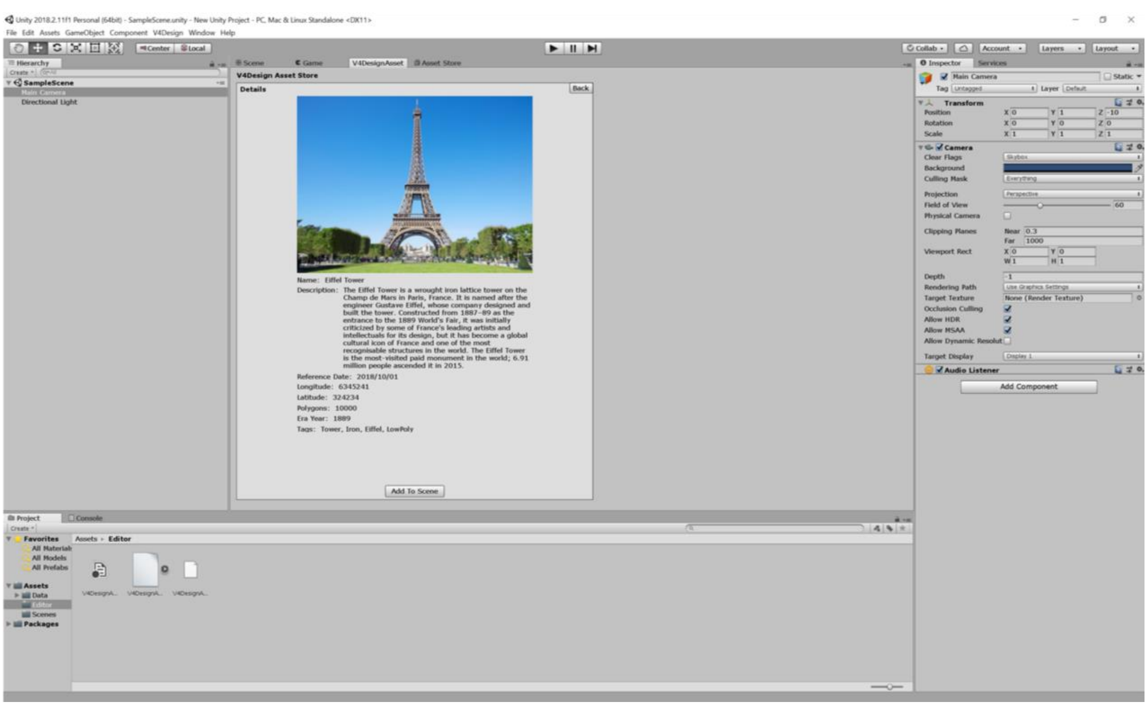


Figure 42: Initial Mock-up of the tool.

Figure 43 represents the current UI with the ability to drag and drop assets inside the VR environment where the users can change the environment in real time in VR. The users can scale the assets, change their positions and delete them as well.

Figure 43: Example of scaling

3.3.6 **Tool evaluation**

The NuroAuthoringTool will be evaluated during the specific deliverables defined for tool evaluation:

- D7.3: Evaluation of the 1st prototype and updated user requirements [M20]
- D7.4: Evaluation of the 2nd prototype [M28]
- D7.5: Final system evaluation [M36]

Apart from this, in 2019 we plan to organise a workshop for game enthusiasts and developers to show the tool, have them use it how they like and evaluate and provide feedback.

The tool will also be tested by user groups of V4design, internally at NURO and DW as the users of the tool. These evaluations will be considered continuously, and feedback will be integrated.

3.3.7 **Tool exploitation (distribution, licensing, exploitation)**

NURO plans to exploit the tool under Apache v2.0 License. The tool will be added to NURO's portfolio of tools and services for various markets. The marketing department of NURO plans to reach out to potential game development companies and other customer cohorts to sell the tool.

The exploitation will take place in multiple stages based on the progress of the tool, firstly information about the tool will be disseminated to the relevant stakeholders, version 1 of the tool would give the stakeholders a demo into its capabilities will be a teaser for exploitation.

Lastly, the full version of the tool will be given as a demo and then can be bought by stakeholders if needed. The final exploitation plan will be based on the market analysis, user requirements and pricing strategy.

3.4 Web Platform (API Interface)

The web platform will act as an asset store for the assets extracted by the V4Design platform. The user can download, rate and comment on the 3D models. This will provide the V4design REST API a web interface.

The web platform will be written using HTML, CSS, JavaScript and angular JS. The component will be available for the mass, more generalised market rather than the specialised markets the NURO authoring and the V4D4Rhino tools plan to target to explore V4Design assets and capabilities of the system.

4 CODE ORGANIZATION

The V4Design platform is a distributed system composed of heterogeneous modules, each developed under its own specifications and proper development framework. Due to its complexity, the platform cannot be easily compiled and published as a single solution. Although this is technically possible, making such an integrated product may lie outside the scope of V4Design, as it entails efforts for product development and consolidation that are generally addressed beyond TRL7 maturity level, such as enterprise integration and enterprise packaging.

In order to effectively organize the sharing of code and the publishing of solutions, both in-house and public, the following approach has been adopted.

The packaging and publishing of V4Design modules can vary according to the development technology and deployment environment of each, and no unified approach will be devised. A code repository that groups all the code components will be provided and used to share codes among the partners. Open-source modules can have their code published to the public in coordination with the consortium.

Modules (both as compiled solutions and as code) are published according to their type, and the following three different types have been identified.

Complex modules that are composed of several integrated components, often customized to befit a particular configuration, can be published as “Docker Images” [5], facilitating their deployment by third parties. Docker images encapsulate the entire module with its integrated components and underlying dependencies in a manner that allows easy redeployment in other environment, similarly to migrating virtual machines. Docker images can encapsulate databases and specific server configuration files and setup programs, making it an easy and effective option for publishing integrated modules. In addition to the docker instance, a copy of the source code is also published allowing third parties to modify and build on top of the existing modules.

Simple modules (e.g. packaged programs and algorithms) can be published as source-code alongside their compilation and execution instructions. The deployment of such simple module depends on specifications determined by their development environments (e.g. Java programs can be deployed as Java Servlets or Python programs can be executed as background algorithm).

Modules derived from open-source solutions are not published unless they have been changed or altered in a manner that diverts them from their developers’ product tree. Their configuration, deployment and integration within the V4Design platform can be published to facilitate the platform’s complete deployment and integration by third parties.

4.1 Source tree layout

In order to house the codes of the different modules and make them available for other partners in the consortium to consult, a GitLab repository account has been created for the project. This account allows partners to publish code securely, and control access to it (privately shared or public). Each service, middleware module, and tool have its code hosted on this repository. This is explained in the following table and depicted in Figure 44.

Table 21: Licensing and distribution of V4Design modules

Module	Policy	Code repository	License
Language Analysis	Public	V4Design code repository: https://gitlab.com/v4designEU/v4d-text/v4d-text-integration	Most likely license: Apache Licence v2.0. Possible different license for third-party components.
Language Generation	Public	V4Design code repository: https://gitlab.com/v4designEU/v4d-text/generation-grammars	Most likely license: Apache Licence v2.0. Possible different license for third-party components.
V4D Crawler	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4d_crawler	Apache Licence v2.0
Aesthetics Extraction	Protected	V4Design code repository https://gitlab.com/v4designEU/v4design-aesthetics	Apache Licence v2.0
Texture Proposals	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4design-tp	Apache Licence v2.0
KB Population	Protected	V4Design code repository: https://gitlab.com/v4designEU/kb-and-reasoning/demo-2018/jsontordfmapping	Apache Licence v2.0
Reasoning	Protected	V4Design code repository: https://gitlab.com/v4designEU/kb-and-reasoning/demo-2018/converttordf	Apache Licence v2.0
Object Localization	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4design-stbol	Apache Licence v2.0
3D Reconstruction	Protected	V4Design code repository:	TBD

		https://gitlab.com/v4designEU/3d-recon	
Message bus	Public	Publicly available from the original developers' website. V4Design code repository: https://gitlab.com/v4designEU/v4d-messagebus	Apache Licence v2.0
V4D REST API	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4d-rest-api	Apache Licence v2.0
Data Storage and Retrieval	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4d_data_storage	Apache Licence v2.0
Video Games Authoring tool	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4d4unity	Apache Licence v2.0
Architecture Authoring tool	Public	V4Design code repository: https://gitlab.com/v4designEU/v4d4rhino	MIT

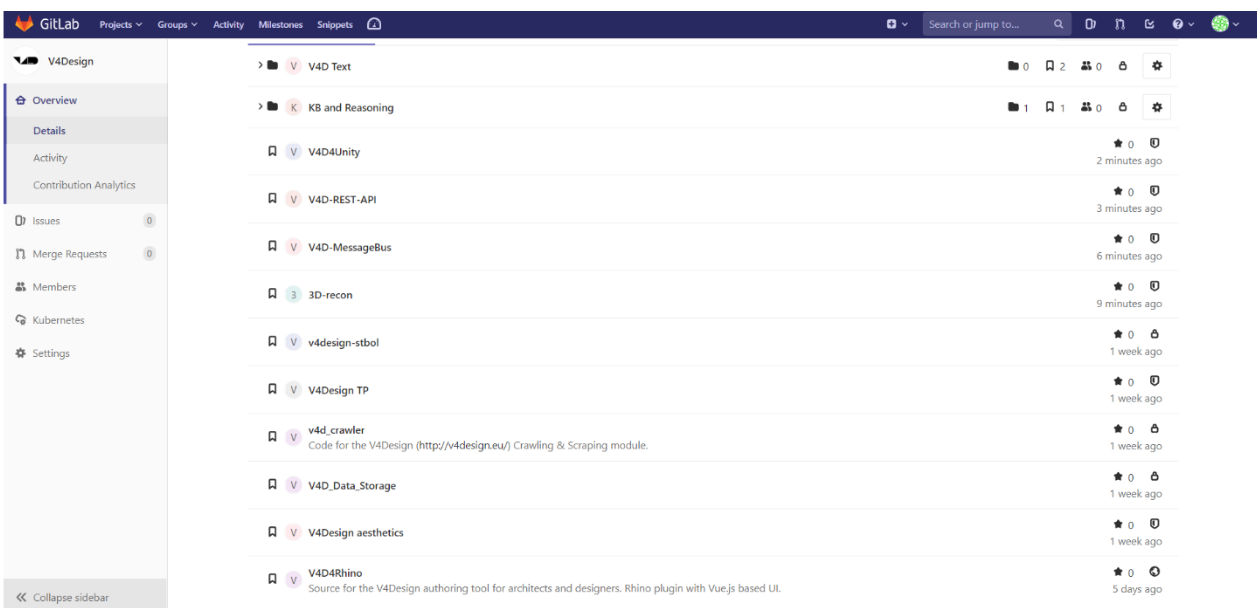


Figure 44. V4Design Gitlab repository

4.2 Packaging

Packaging describes the distribution and installation process of each module in the platform. Currently, each module has a proper packaging strategy, which corresponds to its development framework and underlying technologies. Some modules are server-side and contained in Docker images, others are installed as backend components, and some are installed on the user machine as desktop applications.

The following table specifies for each developed module, the packaging strategy that it follows.

Table 22: Packaging policies of V4Design modules

Module	**Distribution**	**Installation**
Language Analysis	Docker	Deployment using Docker
Language Generation	Docker	Deployment using Docker
V4D Crawler	Java source code (Maven project) and jar executable, along with configuration files.	Deployment using Docker (e.g. Swarm, Kubernetes, …).
Aesthetics and Texture Extraction	(Docker, exe, .app, .zip, tinfoil) python source code	Deployment using Docker
KB Population	Java application	Deployed over any java runtime environment
Reasoning	Java application	Deployed over any java runtime environment
Object Localization	(Docker, exe, .app, .zip, tinfoil) python source code	Deployment using Docker
3D Reconstruction	Compiled executable for windows 64bit	Manual installation
Message bus	Windows and Linux distributions	Installation instruction available on the developer's website.
V4D REST API	Docker	Deployment using Docker
Data Storage and Retrieval	Jar executable of a Spring boot project, along with configuration files.	launched by running the jar executable.

Video Games Authoring tool	Dll file	Import into Unity as an extension (currently works only in Windows OS)
Architecture Authoring tool	A .net plugin for Rhino 6 in the form of a .rhi (Rhino Installer Engine extension) or .yak (Rhino Package Manager extension) which are both .zip files with some specific file structure.	Double clicking on the .rhi or from Rhino's Package Manager interface.

5 DEMONSTRATOR URLS AND INFORMATION

In the following section we describe each prototypical service developed as a demonstration for the role that each architecture component performs. Some of the demonstrations are more mature than others, for instance some are completely functional while others are in early development stages.

In all cases, the demonstrations implement the following architecture requirements and technical functionalities:

- Each demonstrator is hosted on a server (demonstrators can share servers but should communicate exclusively through the message bus - no local communication is allowed).
- The demonstrators should be able to remain online for at least a single platform cycle (processing of an incoming array of raw data objects).
- Each demonstrator connects to the message bus and implements a client capable of sending and receiving messages.
- The demonstrator is capable of responding to the topics to which it should subscribe according to the platform cycle design.
- Demonstrators that generate output related to the user assets generated by the platform should be able to push data onto the Data Storage and Retrieval
- Demonstrators should be able to read data from the Data Storage and Retrieval by sending get requests.

The demonstrators should meet the expectations related to the maturity of their corresponding modules according to the development roadmap described in D6.1 and updated in section 2.2, 2.3 and 2.4 in this deliverable.

The current deployment environment of each demonstrator is explained in the following table.

Table 23: Description of the demonstrators' deployment environment

Demonstrator	**Current Deployment Environment**
Language Analysis	Docker Swarm at UPF, no public IP. Language Analysis component establishes connection to message bus.
Language Generation	Docker Swarm at UPF, no public IP. Language Generation component establishes connection to message bus.
V4D Crawler	Independent server with the following system specifications: Windows 10 Pro, Intel® Xeon® Silver 4108, 128GB RAM, 452GB SSD + 3.54TB HDD. Installed Software: Java 8, MongoDB 3.4. It is triggered offline by the system administrator.
Aesthetics and Texture Extraction	Deployed in an independent server with the following configuration: Operating System: Windows; CPU: N/A; GPU RAM: 2-3 GB; RAM:

	N/A; Disk Space: 30 GB Tensorflow -gpu 1.1.0,Python 3.5, OpenCV 3.3.1, keras-gpu 2.1.6, pandas 0.23.0, matplotlib 2.2.2, anaconda 1.6.14, h5py 2.8.0, numpy 1.12.1, pillow 5.1.0, scikit-learn 0.19.1
KB Population	The KB Population is currently deployed at a local server with an IP 160.40.50.196:7200 with the following configuration CPU: Intel® Xeon® Silver 4108, RAM: 128GB, HDD: 452GB (SSD) + 3.54TB (HDD), both Raid 1, GPU: NVIDIA GeForce GTX 1080Ti, OS: Windows 10 Pro 64-bit
Reasoning	Deployed in a local server with the following configuration: CPU: Intel® Xeon® Silver 4108, RAM: 128GB, HDD: 452GB (SSD) + 3.54TB (HDD), both Raid 1, GPU: NVIDIA GeForce GTX 1080Ti, OS: Windows 10 Pro 64-bit
Object Localization	Deployed on a server with the following configuration: Operating System: Windows, CPU: N/A, GPU RAM: 2-3 GB, RAM: N/A, Disk Space: 30 GB Tensorflow -gpu 1.1.0,Python 3.5, OpenCV 3.3.1, keras-gpu 2.1.6, pandas 0.23.0, matplotlib 2.2.2, anaconda 1.6.14, h5py 2.8.0, numpy 1.12.1, pillow 5.1.0, scikit-learn 0.19.1
3D Reconstruction	Local server with the following configuration: Windows 10 pro, Intel Xeon w2133, Nvidia 1080gtx. Network storage. Installed software: .net framework, and Docker.
Message bus	The current deployed version of the V4D Message Bus is operational and hosted on its own server. The current server is a virtual cloud server with a fixed IP 34.253.156.62, accessible through the DNS: https://bus.v4design.eu. The message bus console can be accessed through the following port: https://bus.v4design.eu:8162/. Currently, it supports the following protocols in an interoperable manner: ● OpenWire ssl: bus.v4design.eu:61617 ● AMQP amqp+ssl: bus.v4design.eu:5671 ● STOMP stomp+ssl: bus.v4design.eu:61614 ● MQTT mqtt+ss: bus.v4design.eu:8883 ● WSS wss: bus.v4design.eu:61619 The server has 1GB RAM memory and 10GB disk space, and a Ubuntu Linux operating system. Installed are: ● An instance of ActiveMQ 5.15.0 ● Apache KahaDB

	• Apache Tomcat • Java Runtime Environment (JRE) JRE 1.7 The current message bus can handle up to 1000 requests per minute. It listens to its open ports for messages to channel. Messages are addressed in First-in-first-out order, each message is assigned to one of several pre-existing topics, to which the services listen. When a message is added to a topic, it propagates instantaneously to the topic listeners.
V4D REST API	Not Yet Available
Data Storage and Retrieval	Windows 10 Pro server, Intel® Xeon® Silver 4108, 128GB RAM, 452GB SSD + 3.54TB HDD. Installed Software: Java 8
Video Games Authoring tool	Desktop application installed on the user machine.
Architecture Authoring tool	Desktop application installed on the user machine.

The URLs of the demonstrators developed for the operational prototype are listed below in the following table.

Table 24: Description of the demonstrators' deployment environment

Component	Owner	Demo URL
Language Analysis	UPF	http://taln.upf.edu/v4design
Language Generation	UPF	http://mklab.iti.gr/v4design/lib/exe/fetch.php?media=d6.3-upf_demo_langgen.zip
V4D Crawler	CERTH	http://160.40.51.32:10000/scrapingDemo
EF API Crawler	EF	https://repl.it/@calvinwuyts/combineharvester
Aesthetics Extraction	CERTH	http://160.40.49.184/vbs2018/#/
Texture Proposal	CERTH	http://mklab.iti.gr/v4design/lib/exe/fetch.php?media=texture_proposal_demo.zip
KB Population	CERTH	http://160.40.50.196:8080/ConvertToRDF/
Reasoning	CERTH	http://160.40.50.196:8080/ConvertToRDF/
Data Storage and Retrieval	CERTH	http://34.245.66.12/V4DMB
Building and Object Localization	CERTH	http://160.40.49.184/vbs2018/#/ http://mklab.iti.gr/v4design/doku.php?id=2nd_plenary_review_preparatory_meeting_leuven#demos
3D reconstruction	KUL	http://160.40.49.184/v4d_3d_demo http://136.144.207.251:8080/reconstructiondemo/

Authoring Tool: Architecture	MCNEEL	https://drive.google.com/open?id=1ftI9CxJDkR508qvQ8L2_2Y-Hl7lSX7Dc
Authoring Tool: Video Games	NURO	http://mklab.iti.gr/v4design/lib/exe/fetch.php?media=v4d4unity_demo.zip
Message bus	MCNEEL	http://34.245.66.12/V4DMB

6 SUMMARY AND CONCLUSIONS

In this document, we have described the state of the art of the operational prototypes of the V4Design platform modules. The preliminary implementation is geared to consolidating the integration model of the platform, and establish its processing cycle, as well as to provide a proof-of-concept for each envisioned technology.

Section 2 presented a detailed description of the V4Design architecture, including its conceptual design, a generic definition of a V4Design service, its communication model, its processing cycle, and the input/output model to illustrate how data is processed and created. In addition, services were introduced individually, including concepts, technical requirements and development plans, and sample output examples to illustrate the added value of each service. Then, middleware components and authoring tools were discussed in a similar manner.

Section 3 presented the visual demonstrations developed as part of the operational prototype for the platform, being the message system visualization, the authoring tool for architect, and the authoring tool for video games.

Section 4 described how the code of the different modules and components is organized and shared in repositories, discussing the protection of the code by item. A list of repository addresses was provided to access the shared codes. In addition, the section presented the packaging model of each item, discussing how it is deployed by third parties.

Overall, the development of the operational prototypes has met its goals in accordance with the platform development roadmap detailed in D6.1, with no notable deviations in the plans accorded for each module. The integration of the platform has been completed successfully, but no complete processing cycles have been launched yet, which is planned to commence in the following development cycle.

Most of the operational prototypes show basic functionalities that meet the most basic of its associated requirements. However, each prototype has validated the capacity of its related component to connect to the platform, receive data and process it, and post the results back to other components that come next in the pipeline.

The user tools illustrate how the envisioned user profiles can access and manipulate the data created by the V4Design platform. They represent a placeholder for the envisioned user-experience and will be refined accordingly in the next development cycle.

Finally, with each of the platform modules, including services, middleware, and tools, prototyped and integrated, the work can now focus on building the processing pipeline that would convert raw data collected by the platform to valuable assets for the user, and developing the user experience that shows how these assets add value in the user ecosystem. This will be the focus of the coming development cycle, designated as the first version of the platform.

REFERENCES

[1] https://developer.rhino3d.com/api/

[2] https://developer.rhino3d.com/api/RhinoCommon/html/R_Project_RhinoCommon.html

[3] Unity 3D https://unity3d.com/

[4] What is RhinoCommon? https://developer.rhino3d.com/guides/rhinocommon/what-is-rhinocommon/

[5] Why Docker? https://www.docker.com/why-docker