V4Design

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

D6.4 - 1st prototype and applications for architecture and video game design platforms

Dissemination level:	Public
Contractual date of delivery:	Month 18, 30 June 2019
Actual date of delivery:	Month 19, 05 July 2019
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:	Demonstrator
Approval Status:	Final
Version:	1.0
Number of pages:	53
Filename:	D6.4-V4Design_first_prototype_v1.0

Abstract

This deliverable documents the first prototype of the V4Design platform, i.e. the first version of the implementation of the front-end applications for the architecture and video game design platforms that will be considered in V4Design, as well as the backend development, which will deal with the personalisation of the applications and their support.

co-funded by the European Union

History

Version	Date	Reason	Revised by
V0.1	14/05/2019	ToC and First inputs	Yash Shekhawat (NURO) Ayman Moghnieh (McNeel)
V0.4	10/06/2019	main chapters developed, input from partners solicited	Yash Shekhawat (NURO) Ayman Moghnieh (McNeel)
V0.8	20/06/2019	Partners' contributions integrated	Yash Shekhawat (NURO) Ayman Moghnieh (McNeel)
V0.9	26/06/2019	Version ready for internal review	Yash Shekhawat (NURO) Ayman Moghnieh (McNeel)
v1.0	05/07/2019	Internal review comments addressed and final version is ready	Yash Shekhawat (NURO)

Author list

Organization	Name	Contact Information
McNeel	Ayman Moghnieh	aymanmoghnieh@gmail.com
McNeel	Luis Fraguada	luis@mcneel.com
McNeel	Verena Vogler	verena@mcneel.com
CERTH	Konstantinos Avgerinakis	koafgeri@iti.gr
CERTH	George Meditskos	gmeditsk@iti.gr
CERTH	Elissavet Batziou	batziou.el@iti.gr
NURO	Yash Shekhawat	yash.shekhawat@nurogames.com
NURO	Christian Mueller	christian.mueller@nurogames.com
UPF	Simon Mille	simon.mille@upf.edu
UPF	Jens Grivolla	jens.grivolla@upf.edu
KUL	Jens Derdaele	jens.derdaele@kuleuven.be

ABBREVIATIONS AND ACRONYMS

3DR	3D Reconstruction
AE	Aesthetics Extraction
AMQP	Advanced Message Queuing Protocol
API	Application program interface
CR	Crawler / Wrapper
DB	Database
DoS	Denial Of Service
DSRS	Data Storage and Retrieval System
GUI	Graphic User Interface
HLURs	High-level user requirements
JMS	Java Message Service
JSON	JavaScript Object Notation
KB	Knowledge Base
LG	Language Generation
OL	Object Localization
RDF	Resource Description Framework
SQL	Structured Query Language
SR	Scene Recognition
STBOL	Spatio-temporal building and object localization
TA	Text Analysis
TE	Texture Extraction
TRs	Technical Requirements
UIMA	Unstructured Information Management Architecture
URI	Unique Resource Identifier
URs	User Requirements
VR	Virtual Reality

LIST OF FIGURES

LIST OF TABLES

Executive Summary

This deliverable discusses the implementation of the V4Design V1 platform prototype, which is preliminarily delivered at M18 of the project to conduct user evaluations, and closed by M20 when the third technical milestone of the project is reached.

The V1 platform implements the V4Design process, which is designed to reconstruct valuable assets for game design and architecture applications from media resources extracted from online repositories, social media, and other sources. It extends the functional prototype of the platform, which was delivered at M12 as a proof of concept of the integration model that standardizes and connects the platform services.

The components that make up the V1 platform are discussed, namely the middleware (the platform API, messaging service, and data storage and retrieval system), its three pipelines (3D Reconstruction, Semantics, and Aesthetics), and the authoring tools (architecture and game design). Advances in the implementation of each of these components are discussed and results shown.

In order to validate the V1 platform from a functional perspective, a battery of functional tests have been conducted, processing 64 raw data items selected by the content providers. The results show that the V4Design process has been correctly implemented, and indicate how the implementation can be improved to streamline this process further and expand user control over it.

The deliverable is concluded by summarizing the major achievements and discussing future works, namely the actions contemplated for the next iteration of development under which the second version of the platform will be developed.

TABLE OF CONTENTS

1 INTRODUCTION

The V4Design platform is designed to implement the process of finding and extracting assets from media collections acquired from repositories and web sources, otherwise referred to as the V4Design process in this document. In order to extract the assets, the platform has to identify them, reconstruct models, extract textures and information on the aesthetics of each identified element, and develop elaborate descriptions, among other tasks. The platform is conceived for game designers and architects alike, providing intuitive interfacing and control techniques for asset reconstruction. The streamlining of the V4Design process, including the integration and chaining of different technological modules in a manner that does not bound the flexibility inherent in this process.

At the first iteration of development of the V4Design platform, the conducted analysis on the technical requirements and specifications of the technologies that the platform integrates and the functions it is trying to support, has allowed to establish and implement a standardization of the platform modules as interconnected services, and to create a communication model capable of synchronizing the services under specific processes and sub-processes. This prototypical integration was consequently evaluated and different inconsistencies regarding the data flow and management paradigms, as well as precedence among services have been identified.

In the second iteration, which is mainly documented in this deliverable, these inconsistencies were largely addressed, leading to a more grounded arrangement of components geared to the processing of data and the reconstruction of composite assets (3D models with textures, descriptions, and rich metadata). On top of this arrangement, basic versions of the authoring tools conceived for supporting the user interaction have been developed and integrated. This version of the integrated platform has been deployed in the cloud, connecting a series of servers, each housing one or more services. It is referred to as the V1 platform prototype.

Essentially, and as previously stated, the objectives of the V1 platform is to implement the V4Design process, centering on data processing as a primary concern.

The V1 platform lifecycle is expected to last 4 to 6 months, from the moment it is first delivered at M18. During this period (M18-22/24) it is primarily employed as a testing framework to conduct functional tests on the data processing mechanisms implemented, and discover and remedy shortcomings in terms of compatibility with the data input quality and types, performance, error avoidance and error handling, among other technical concerns. Apart from the functional tests intended, the V1 platform will allow user groups to access the assets reconstructed from raw data, and evaluate their quality, usability, compatibility, added-value, and other concerns. The upcoming development cycle will introduce solutions to the detected aspects and analysed shortcomings. In addition, it will also gear the development efforts towards providing more elaborate support for users, and the processes they are ought to control through the authoring tools.

2 1ST PROTOTYPE ARCHITECTURE

In the following chapter, the first prototype of the V4Design platform is described and delimited from the experimental proof-of-concept developed in the previous iteration. The construction of pipelines geared towards data processing is first discussed, then the characteristics of the resulting platform model are detailed, namely the communication model, the data management model, and the synchronization model.

2.1 Moving from chained services to pipelines

In the previous development iteration, which lasted from the beginning of the project till M12, a thorough analysis of technical requirements was conducted, and an architecture model designed to facilitate the integration of the different components and modules that make up the V4Design platform. In order to validate such a model (see Figure 1), a prototypical integration was implemented by which prototypes of the envisioned services and components were integrated into a single workflow. This workflow implements the generic process of V4Design by which composed and valuable assets are extracted and reconstructed from media acquired from different sources, including online crawling and online repositories. The chaining among the components, which the communication and data exchange models implement, was conceived in accordance with the high-level requirements of each component. For instance, services that process raw data were directly connected to the Crawler service that is responsible for acquiring and distributing new data. Also, services that process more advanced or intermediary data objects were chained accordingly and launched subsequently to the generation of this required intermediary data.

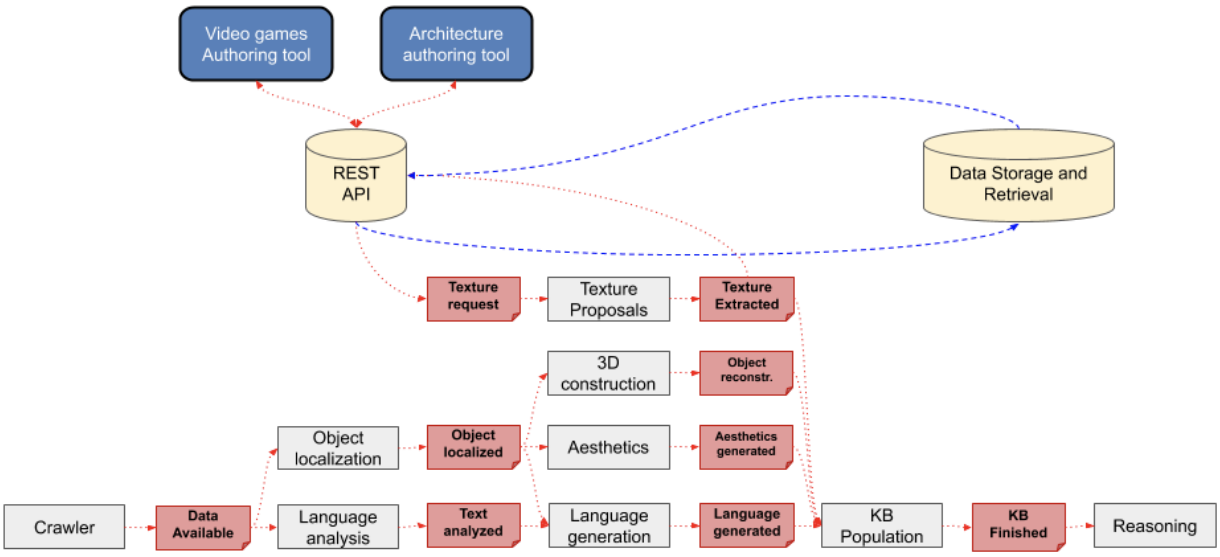


Figure 1 Original communication model of the functional prototype

In this iteration, which spans M12 to M18, the focus of development and integration activities centred on addressing the platform as a data processing system, moving beyond chaining and into data management and synchronization. Under this perspective, the integration model

was revisited to align it better with low-level or more precise requirements, and some alterations were introduced to the original model. By taking this data-processing viewpoint, the architectural concerns of the V4Design platform centred on establishing the aforementioned generic process of V4Design as a pipeline capable of driving raw data objects through the transformation process that creates valuable assets for the architecture and game-design industries.

Consequently, the platform back-end architecture was rebranded as an integration of three specialized pipelines, one in charge of reconstructing 3D models and generating relevant metadata, another in charge of the extraction of aesthetic information, including masks and textures, and a third pipeline in charge of extracting semantic information and generation rich descriptions. These pipelines abstract and integrate the V4Design services in larger back-end components, and implement the V4Design processing cycle according to the following diagram.

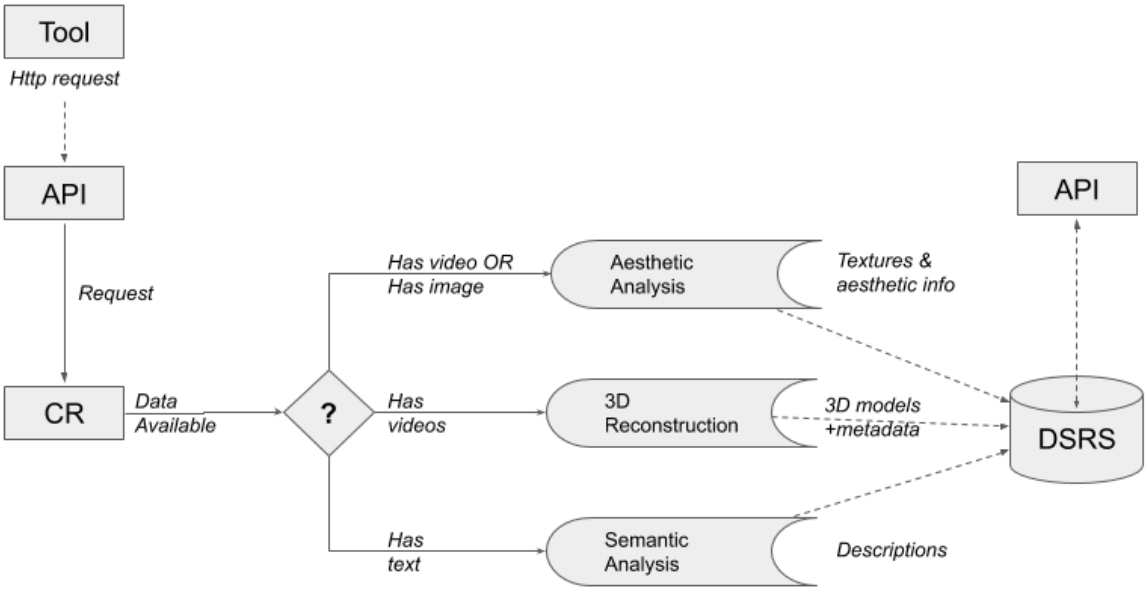


Figure 2 Conceptual design of the current V1 architecture

The envisioned cycle starts by a processing request sent by the authoring tool to the API. The request details the information required to extract the relevant data to process, for instance a URL and a media type (e.g. www.youtube.com/myvideo, webpage). The API translates the request to the Crawler (CR) through the message bus, which crawls and extracts the relevant data and stores it in the platform's Data Storage and Retrieval System (DSRS), and then announces the arrival of new data by a Data Available message (prototyped in V1, but will be functional in V2). The message indicates the nature, type, and content of the data, encoded in parameters such as "has_videos", "has_images", and "has_text". If the data includes a video, then the 3D Reconstruction pipeline works to extract a 3D model from the video. If it includes a video and/or an image, the Aesthetic pipeline starts extracting masks and textures from the data. If it contains text, the Semantic pipeline starts extracting the semantics and conducting a relevant generation of descriptions. The output of each pipeline is stored back in the DSRS, and consequently can be queried and retrieved through the API. It is worth mentioning that the API keeps a buffer of available data in the DSRS for performance-related concerns.

The following table details the main functionalities implemented having direct impact on the architecture and integration of the platform. Other important implemented functionalities pertain to the development roadmaps of individual services and components, and are discussed in their respective upcoming sections of this document.

Table 1 Main functionalities implemented

Functionality	Description	Related Requirements
API implements basic authentication.	The API creates and maintains basic user profiles, and supports authentication as a basic security mechanism.	Personalizing the content, supporting the management of rights and permissions (basic support).
API retrieves index of existing data.	The API is able to retrieve for the tools an index of available data by interfacing with the DSRS.	Users are able to explore the assets available in the platform.
API queries for DSRS.	The API sends queries to retrieve assets based on different criteria, include tags and dates.	Users are able to search for assets relevant to their projects.
API sends a request for crawling.	The API sends a request message to the Crawler to retrieve raw data from a source.	Users can retrieve assets from interesting media they discover online.
Crawler enables re-processing of existing assets.	Upon receiving the related request, the Crawler can re-broadcast a Data Available message to trigger re-processing.	Facilitates the use of the current integrated platform as a testbed for improvements in different services.
DSRS stores objects according to their type and role in the process.	DSRS currently wraps three distinct storage mechanisms, one for raw data, one for metadata and analytical intelligence, and another for 3D models.	Flexibility and performance in data management are key concerns related to the platform back-end.
Object Localization extracts relevant masks to orient 3D reconstruction.	The 3D Reconstruction service solicits masks for specific video frames, allowing it to clearcut the area corresponding to the object been reconstructed.	Reduce clutter and noise in the reconstructed 3D models.
Language Generation reprocesses descriptions upon changes in asset metadata	As different services add to the metadata related to a specific asset, the Language Generation uses this newly available information and reprocesses the descriptions.	Updating descriptions upon newly available metadata.
Message parameterization is now supported.	Components now are able to communicate more effectively with parameterized messages that qualify different aspects of events they describe.	Support services in deciding when to run and how to process data.

These functionalities will be extended and consolidated in the next development iterations in order to support more user control over the V4Design processes, and towards the creation of a more consolidated and integrated service platform.

This coordinated process has different ramifications and interdependencies. For a start, it takes into consideration the current limitations of several main V4Design services and components, as these have not yet reached the ultimate maturity and complexity level intended under the project plan. Instead, while in development and progressing satisfactorily according to the project development and implementation roadmap, these services currently manifest several constraints. For example, the 3D Reconstruction service is not yet geared to extract 3D models from collections of images, although early tests on the subject show promising results. This limitation is not exclusively related to the status of this service, but also to the manner by which collections of images are ought to be created by other modules of the architecture, a concern that is scheduled to be addressed in the next iteration. Another example is the Texture Extraction service, which would require a user intervention at some point in the process of extracting textures from media content, and applying them onto generated 3D models. This is also scheduled for the next iteration, when the V4Design processes become more user-driven.

The following table shows the main technical limitations of the current architecture, and the contemplated resolution plan of each.

Table 2 Technical Limitations of the Architecture

Limitation	Description	Resolution plan
Data from content providers is extracted in a manual fashion.	The content providers' platforms and the V4Design platform are not yet integrated. Content is currently selected in a curated manner by the providers and stored in a FTP server for retrieval by the V4Design platform.	The plan to integrate the content providers' APIs has been planned finalized, will be implemented in M19-M20.
Textures are manually selected.	Although the Texture Extraction is capable of extracting several textures from media, this process was limited to a selection before it becomes user-driven in the next iteration.	Devise feasible approaches, including user-driven ones, to select relevant textures from a set of candidates.
3D Reconstruction works exclusively on videos.	The current configuration of the platform only allows 3D models to be reconstructed from videos.	Alternatives are being considered to support extraction of 3D models from images in the next iteration.
3D Reconstruction needs to be optimized or scaled.	The current performance indicators of the 3D Reconstruction pipeline needs to be improved notably in order to allow this process to scale.	Besides specific optimizations contemplated on the level of 3D Reconstruction service, plans will be drafted to support vertical and/or horizontal scaling of this pipeline.

2.2 Characteristics of the current architecture model

In the following, we summarize the characteristics of the current architecture model associated with the V1 platform prototype. Name, we discuss three architectural concerns: the communication model, the data management model, and synchronization.

2.2.1 The platform's communication model

The communication model, consisting of predefined message topics and their respective parameters, has been updated in order to 1) allow ramifications and exceptions in the manner by which services chain (e.g. a single service can perform several roles), and 2) to pass instructions about the data that is being processed, its status, and type.

For instance, the 3D Reconstruction service now sends a "Frames Requested" message to the Object Localization service, asking for the extraction of specific masks from selected frames, so as to refine the reconstruction and eliminate noise to the extent possible.

In the following table, we show the current configuration of the communication model for the V1 platform prototype.

Table 3 Services and Messages

Service	Message consumed	Message params	DSRS PULL Call	DSRS PUSH Call	Message produced
Crawler	CRAWL_IT	urls	N/A	Push array of simmos	DATA_ AVAILABLE
Language Analysis	DATA_ AVAILABLE	has_text = True	/retrieve (Task=CR, Id=Id, Entity=Texts)	/save analyzed text (JSON)	TEXT_ ANALYZED
Language Generation	REASONING_FINISHED	ID = id	/retrieve (Task=LA,id=id, Entity=textAnalyzed)	/save generated text (JSON)	TEXT_ GENERATED
Scene Recognition	DATA_ AVAILABLE	has_video = True	get(Task=CR, Id=Id, Entity=Images)	Push scene per frame	SCENE_ RECOGNIZED
Object Localization	DATA_ AVAILABLE	has_image = True	get(Task=CR, Id=Id, Entity=Images)	Push mask_generated	SCENE_ RECOGNIZED
Object Localization	FRAMES_AVAILABLE	ID = id; array frame ids	get(Task=CR, Id=Id, Entity=Videos)	Push mask_generated	OBJECT_ LOCALIZED
Texture Proposal	FRAMES_AVAILABLE	has_masks = True	get(Task=CR, Id=Id, Entity=Images)	Push textures_ generated	TEXTURE_ EXTRACTED
Aesthetic Extraction	DATA_ AVAILABLE	has_textures = True	get(Task=TP, Id=Id)	Push Aesthetics_ generated	AESTHETICS_ EXTRACTED
3D Reconstruct.	SCENE_ RECOGNIZED	Has_video = True	get(Task=CR, Id=Id, Entity=Videos)	Push 3D model	FRAMES_ REQUESTED
API	ON_UPDATE	ID = id, others	/retrieve (d=id)	N/A	N/A

2.2.2 The platform's data management model

The data management model rests on communication between the platform's modules and the Data Storage and Retrieval System (DSRS). Each module is enabled to push and pull data from the DSRS according to the generic design of a V4Design service. This is done through

HTTP protocol calls to the DSRS's API, which wraps three distinct database systems: the SIMMO DB, the Knowledge Base (KB), and the 3D model metadata DB.

The SIMMO DB stores raw data in SIMMO [4] format, which standardizes how media is organized into raw data objects. The database technology used for the SIMMO database is MongoDB. The 3D model metadata DB is a PostgreSQL database designed to store textual and numerical content associated with the created 3D models and their variations (format, textures applied, etc.). It also stores data used for the reconstruction process and the appropriate relations, such as videos, frames, masks and images. The KB structures the knowledge and metadata generated about each created asset in the form of RDF triples in a GraphDB repository. It interfaces with the authoring tools through the platform's API. Each one of the three databases expose their required functionalities through an API that is only accessible by the DSRS module. Apart from that, the module hosts a static file folder that includes the multimedia files used by the project. Web functions are developed that enable the V4Design modules to upload and download content into that folder. The DSRS module and its interactions with the databases and the V4Design modules is illustrated in Figure 3.

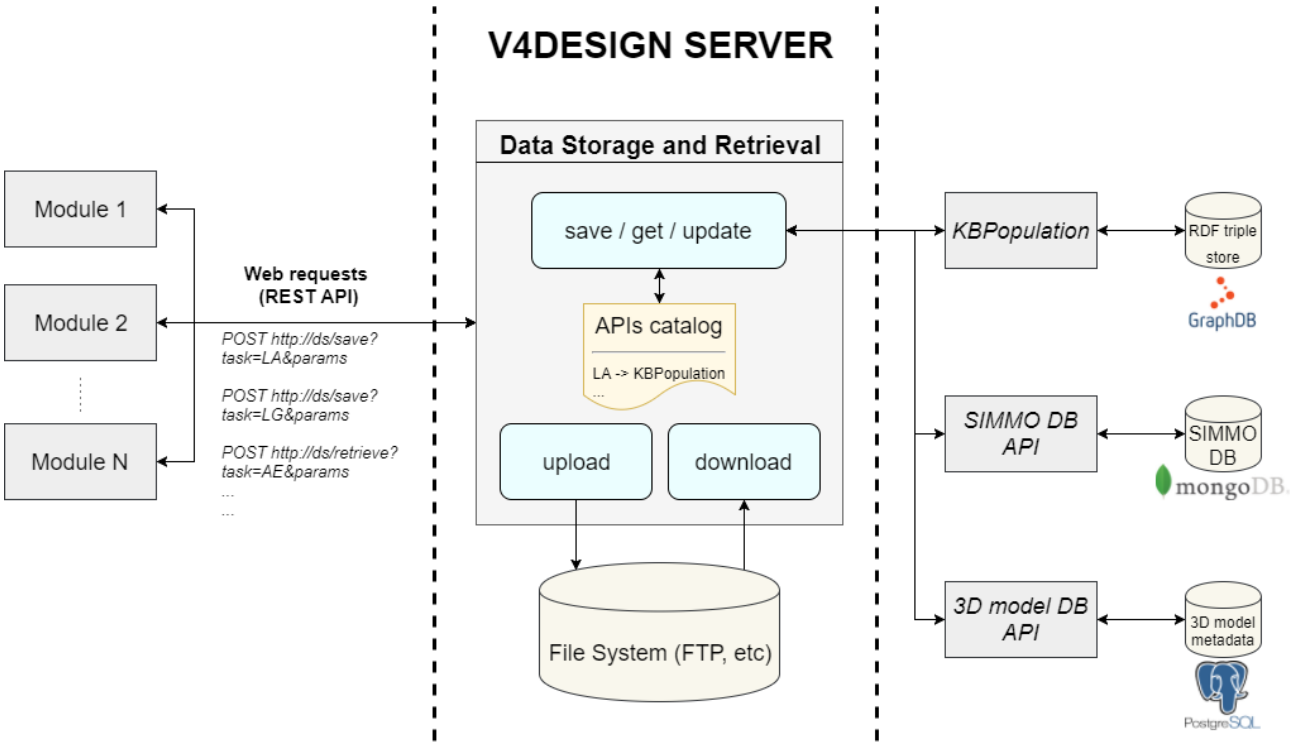


Figure 3 Overview of the V4Design data storage and retrieval design

2.2.3 Security management

As part of the implementation of the 1st version of the V4Design platform, a preliminary security management model was deployed.

There are different security threats to account for in distributed systems, and countermeasures need to be accounted for in order for the system to remain operational and reliable. We identify the following three major threats as pertinent to the V4Design platform.

Unauthorized access

Unauthorized access may be considered as the highest threat to distributed systems, especially if local node configurations are not well set. There is a strong case for implementing an access control system early-on in the development process in order to govern all requests and communication among components with a secure policy.

At this stage in the platform development, an access control policy has been deployed, consisting of an authentication-through-middleware approach, by which using any middleware requires authentication and session management, be it so with the system's API, Message Bus, or Data Storage and Retrieval. This policy can be expanded in the future to cover aspects related to permissions, and usage quotas.

As regards the three database APIs that act on the backend of the Data Storage and Retrieval module and are not intended to be available for public usage, stricter security measures have been considered. More specifically, an IP address filter is applied so as to accept connections only from the machine where the Data Storage and retrieval module is installed.

Denial of service

Involves attacks that affect the availability of information from the system to the user resulting to paralysation of the entire operation of an organization or part of activities depending on the attack. The use of resource control mechanism can help in solving the above problem by applying timing responses, sizing responses, and connection control. Also problem detection by timing latency in system can easily be done if there is a dramatic increase of latency then denial of service (DoS) can be detected as well as addressed.

At this stage in the platform development, connection control mechanisms have been enforced (access control credentials), mainly through the middleware (message bus and data storage and retrieval system). In the case of V4Design, timing responses of different services may not be applicable or accurate, therefore we adopt a more centralized approach to security to face possible DoS attacks. In particular the deployment of a redundant copy of the message bus to which traffic can be transferred seamlessly is contemplated in the next phase of development this helps in preventing DoS attacks to shut down the system, since if one copy if overloaded the other copies will start. Beyond this, and given that services only respond to messages from the message bus, the risk of DoS attacks could be considered as mitigated.

Information leakage

This is one of the threats of computer systems, specifically in the case of distributed systems like the V4Design platform, where sensitive data can easily be revealed to unauthorized users that results to lack of confidentiality, privacy violation, copyright infringement, and other concerns.

At this stage in the platform development, only curated content provided by the consortium partners has been used to perform functional tests on the platform and prepare a demo dataset for evaluation and for training potential users. The risks associated with this dataset are minimal, nonetheless several steps have been taken to mitigate the risk of information leakage. Data storage is centralized in the platform below a wrapper that forces authentication before pulling or pushing data. This guarantees that only certified V4Design services can access the data, including the platform's API responsible for passing the data to

the users (through the tools). An asset ownership paradigm has been designed to allow the platform to control ownership of created asset, so as the assets of one users are not shown to another if they are private. Also, an asset origins paradigm is contemplated to describe the copyrights associated with the raw data that was used to create the asset. This will not only allow services to take a decision to process this data or not, but also allows to cascade these rights throughout the transformation process by which V4Design assets are created.

In addition, other measures related to security have been taken, such as adding security certificates to the different machines used, including those that host the middleware.

3 THE PIPELINES

The new architecture model introduces three pipelines, each as a technical model encapsulated in the platform's backend. These pipelines in turn are composed of the services that implement the technologies defined and designed in the project. Furthermore, each pipeline centres on a specific high-level concern of asset reconstruction. The 3D Reconstruction pipeline streamlines the process of extracting 3D models from media; the Aesthetic pipeline streamlines the aesthetic analysis and extraction from raw data; and the semantic pipeline streamlines the generation of descriptive rich content using semantic analysis.

3.1 The 3D Reconstruction Pipeline

The 3D reconstruction service architecture is structured as a collection of coupled services, each with its own task(s). This heavily improves modularity of the system and subsequently makes for easier understanding, development, testing and future re-use. Figure 4 illustrates the deployed system architecture for the first prototype.

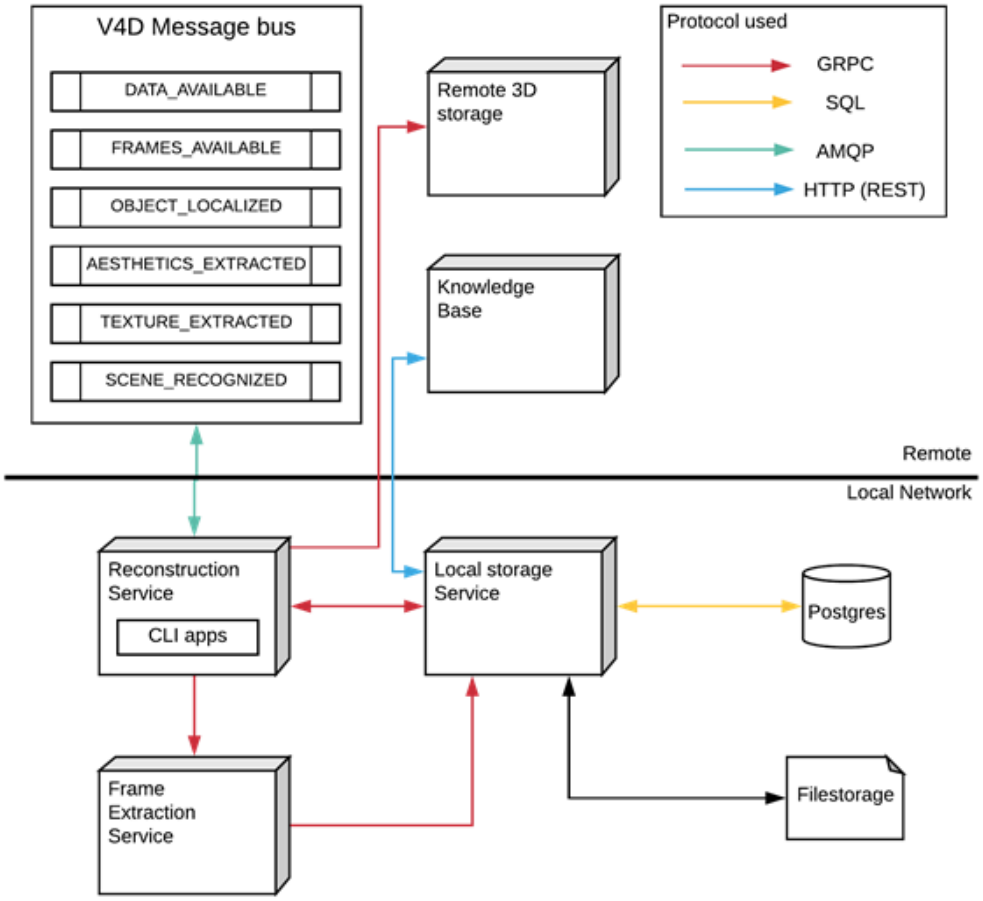


Figure 4 Overview of the currently deployed reconstruction service architecture

Communication between internal services consists of a lightweight protocol using Remote Procedure Call (RPC). The open source gRPC library, which allows automatic generation of cross-platform client and server bindings for many languages (C#, C, C++, JS, Java, Python, Clojure and others) was used. For this reason, individual service nodes such as the 'frame extraction service' may be reused in nearly all development environments without the need to change the source code or rebuild the application.

Figure 5 shows an example message that is pushed to the KB and the message bus notifying other services that a new model was generated. A more thorough discussion concerning the meshes exported from the reconstruction pipeline is presented in D4.3 First iteration of 3D reconstruction and scientific Report, section 5.5.


```
{
  "Id": "AH7X940UUE",
  "visualAnalysisMaskCollisions": [ "trees", "building/inn" ],
  "meshes": [
    {
      "Id": "KLSH4D",
      "format": 0,
      "faceCount": 147416,
      "textureCount": 4,
      "reconstructionId": "AH7X940UUE",
      "modelurl": "https://160.40.49.184/reconstructions/KLSH4D/texturedMesh.fbx"
    }
  ],
  "pointclouds": [
    {
      "Id": "",
      "reconstructionId": "AH7X940UUE",
      "pointCount": "124863",
      "file": "",
      "potreeLink": ""
    }
  ],
  "Simmos": [  // id of used content (1 video in this case)
    "11988275-7604-4121-b3bf-6e5d848f6261"
  ],
  "modelurl": "https://160.40.49.184/reconstructions/KLSH4D/texturedMesh.fbx",
  "Idmodel": "KLSH4D",
  "thumbnailurl": "https://160.40.49.184/reconstructions/thumbnail.jpg"
}
```


Figure 5 Example JSON-formatted message pushed to KB and message bus

3.1.1 Dependencies

This paragraph discusses the dependencies across the different V4Design services. An overview of internal source code dependencies (used software/libraries) is discussed in D4.3 section 5.5. The reconstruction service of the first prototype has dependencies for the following message bus topics:

Table 4 3D Reconstruction dependencies

Topic	Provided by	Purpose
DATA_AVAILABLE	Any service providing raw data (e.g. Crawler)	initiates processing if suitable raw data is present in the message
SCENE_RECOGNIZED	Scene recognition service	Provides tags on a per-frame basis.
OBJECT_LOCALIZED	STBOL service	Contains masks on a selection of used imagery. Useful for additional tagging & clutter removal
TEXTURE_EXTRACTED	Texture proposal service	Contains data in order to retexture meshes in different styles

Following messages are distributed by the service:

Table 5 Messages by the 3D Reconstruction service

topic name	Serves following services	Purpose
FRAMES_AVAILABLE	Texture proposal	Forwards the actual images/frames **used in the reconstruction**.
OBJECT_ RECONSTRUCTED	Segmentation (Not implemented in 1st prototype)	Notifies services on new reconstruction

3.1.2 Limitations

The deployed system in Figure 4 has some limitations in terms of architectural design.

Stability

By having both computational-heavy and important system management code (e.g. message bus connection) in the same node (‘Reconstruction Service’) stability issues are a concern. We aim to tackle this limitation by splitting the “Reconstruction service” seen in Figure 4 into a “manager service” connected to the MB and compute-heavy “reconstruction service” as seen in the updated schema in Figure 5.

Scalability

Currently, there is no way to scale the system in a horizontal matter that is across multiple servers / computers. For this reason the system’s capacity is limited to 1 reconstruction processing at a given time. By implementing the proposed “manager service” it would allow us to run many reconstructions across different servers and even different networks.

Exposed metadata

The “Local storage service” module shown in Figure 4 exposes an API for the management of 3D model data. Part of this API should be exposed to other partners should they need more elaborate metadata for the models than is currently send to the KB and MB. This system is part of the “3D Model DB API” shown in Figure 4. For the 1st prototype no specialized 3d model metadata was required. For this reason we opted to not expose the current grpc API concerning safety reasons. In the 2nd prototype we plan to expose a subset (metadata retrieval-only) of the current local API in either GRPC or REST.

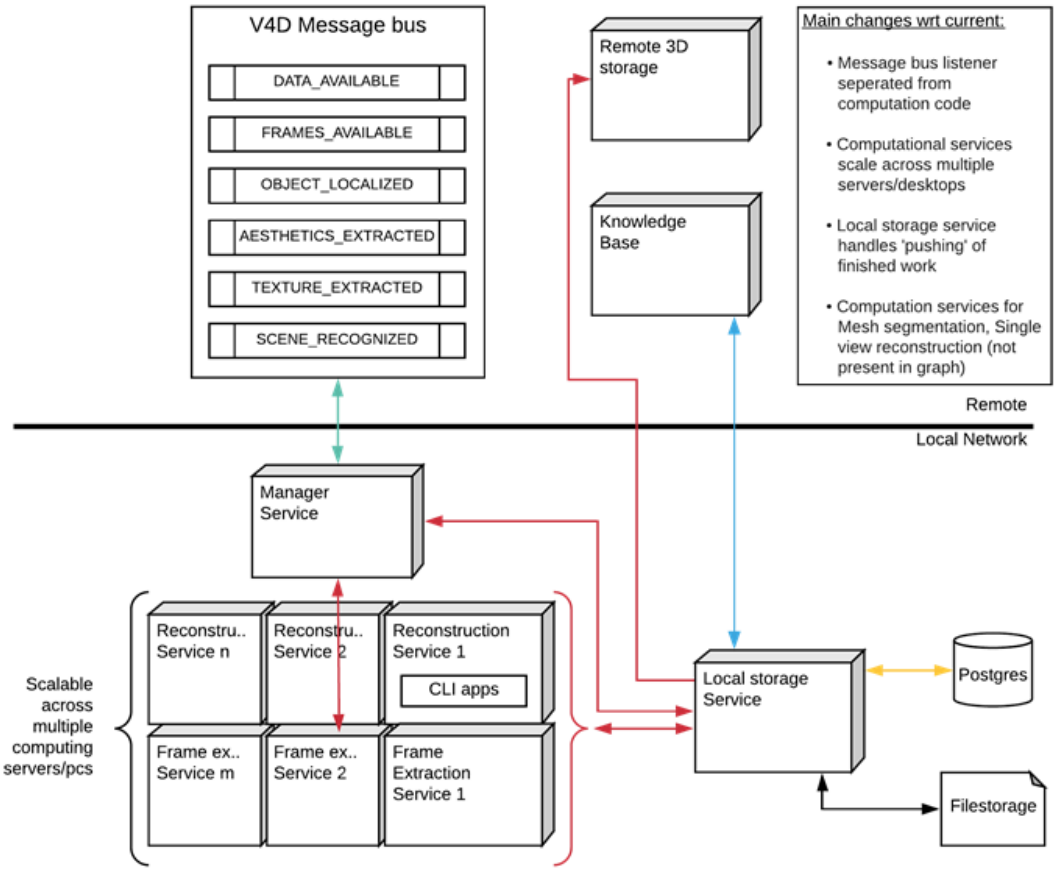


Figure 6 Overview of future architecture with support for horizontal scaling

Figure 6 shows a draft of an updated architecture tackling these limitations. It may be implemented with relatively minor system architecture change and supports a more stable and horizontally scaling design.

3.1.3 Examples of generated assets

Figure 8 shows some output generated during and at the end of the 3D reconstruction process. For each video appropriate frames were automatically extracted (Figure 7), followed by image alignment (camera position estimation). Sparse and dense pointcloud models are computed based on these aligned keyframes. Lastly, based on the dense pointcloud, a textured mesh is extracted. A more detailed step-by-step elaboration of the reconstruction module is presented in D4.3 section 5.4 (different steps according to the schema in Figure 4).

Figure 7 Frame extraction results (5 of 102)

Figure 8 Some models generated using the 1st prototype pipeline

3.2 The Aesthetic Analysis Pipeline

The AE pipeline contains two parts: Aesthetics extraction (AE) and Texture proposals (TP). The input to AE is images or video frames provided by the V4Design Crawler component (CR). The outputs of AE are tags for style and creator of given input that are sent to the Knowledge base (KB). The input to TP is a set of selected images and video frames of 3DR and the output is a collection of new images which have the content of given visual input and the style of famous paintings images that are sent to the Knowledge base (KB).

The following diagram illustrates the AE pipeline.

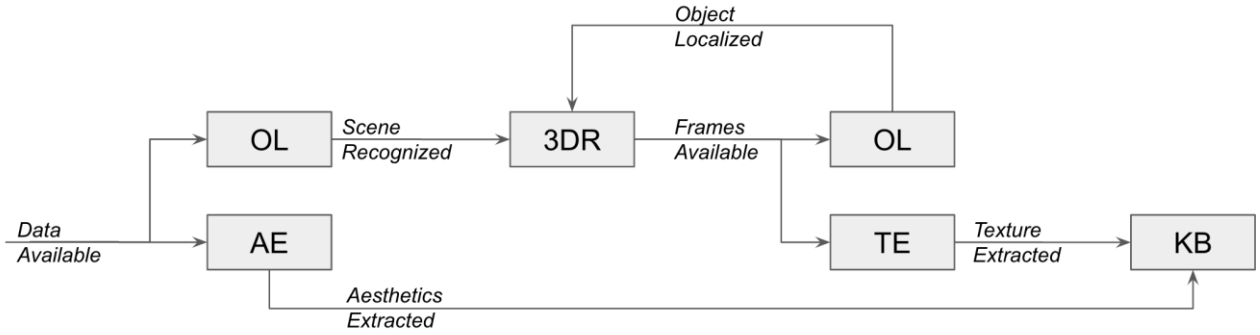


Figure 9 The AE pipeline

Dependencies

The AE service listens to the DATA_AVAILABLE topic which is provided by Crawler and sends the output to AESTHETICS_EXTRACTED topic. The TP service listens to the FRAMES_AVAILABLE topic which is provided by the 3D Reconstruction module and send the output to TEXTURE_EXTRACTED topic. In the following Table the communication between services of AE pipeline is presented.

Table 6 Dependencies of Aesthetic Analysis Pipeline

Topic (listen)	Provided by	Provided to	Topic (send)
DATA_AVAILABLE	Any service providing raw data (eg Crawler)	Aesthetics Extraction (AE)	AESTHETICS_EXTRACTED
FRAMES_AVAILABLE	3D Reconstruction	Texture Proposals (TP)	TEXTURE_EXTRACTED

3.2.1 Examples of generated assets

Following we present output examples generated by AE and TP services based on specific inputs.

Aesthetics extraction input:

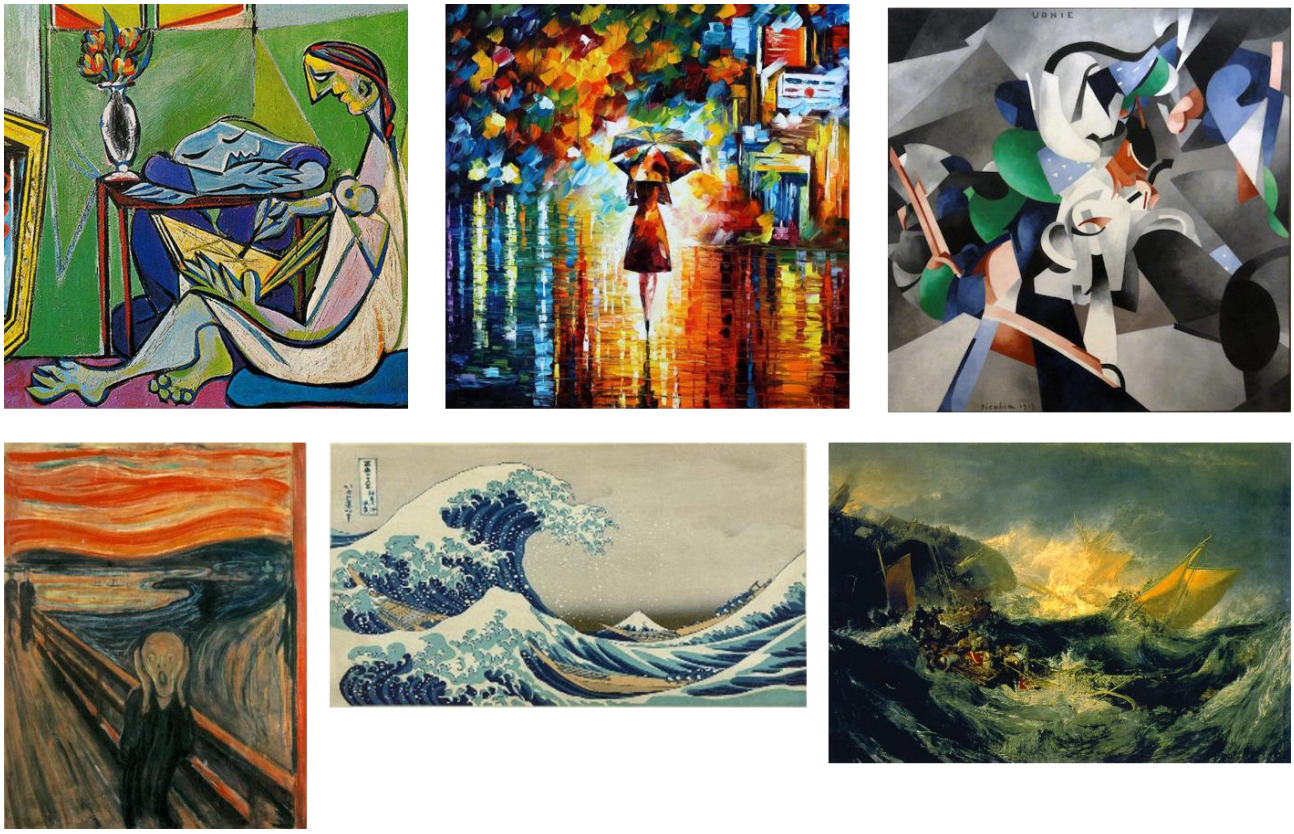


Figure 10 Input images used for aesthetic extraction

Aesthetics extraction output:

{"probability_style": "0.73710732", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “0", "style": "Northern Renaissance", "probability_creator": "0.55724233", "creator": "Pablo Picasso"}

{"probability_style": "0.79951937", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “1", "style": "Fauvism", "probability_creator": "0.48697265", "creator": "Childe Hassam"}

{"probability_style": "0.85213825", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “2", "style": "Cubism", "probability_creator": "0.46858338", "creator": "Salvador Dali"}

{"probability_style": "0.87316581", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “3", "style": "Expressionism", "probability_creator": "0.34179238", "creator": "Vincent Van Gogh"}

{"probability_style": "0.78018423", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “4", "style": "Ukiyo-e", "probability_creator": "0.47149060", "creator": "Martigos Saryan"}

{"probability_style": "0.75621650", "simmo": "115eh746-5gj5-391d-4ag6-64ker46lpa78", “frame”: “5", "style": "Romanticism", "probability_creator": "0.56946965", "creator": "Vincent Van Gogh"}

Texture Proposals input:

Figure 11 Input image used for texture proposals

Texture Proposals outputs:

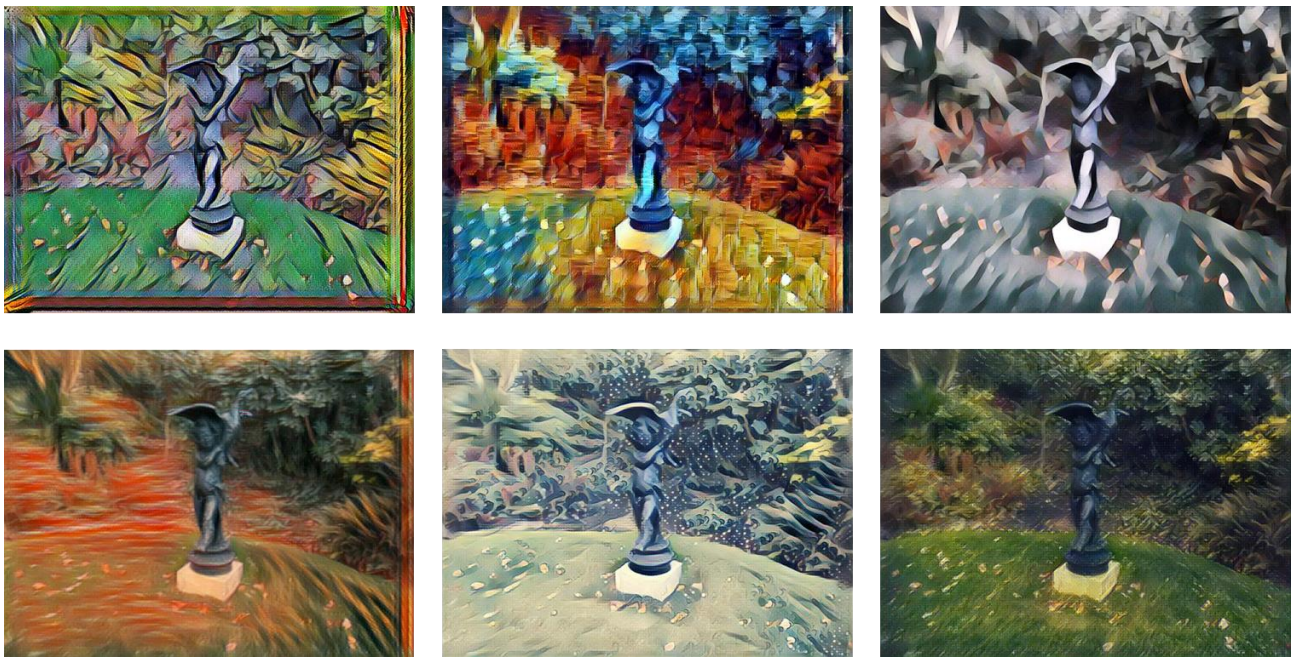


Figure 12 Output generated by aesthetic extraction

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": "0", "artist": "Pablo_Picasso", "style": "Cubism", "style_image": "La_muse", "output_resource_url": "http://160.40.51.32:10010/download/test/12962.jpg"}

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": "0", "artist": "Leonid_Afremov", "style": "Oil_Painting", "style_image": "Rain_princess", "output_resource_url": "http://160.40.51.32:10010/download/test/12965.jpg"}

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": “0", "artist": "Edward_Munch", "style": "Modern_Art", "style_image": "The_scream", "output_resource_url": "http://160.40.51.32:10010/download/test/12969.jpg"}

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": “0", "artist": "Frances_Picabia", "style": "Abstract_Art", "style_image": "Udnie", "output_resource_url": "http://160.40.51.32:10010/download/test/12972.jpg"}

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": “0", "artist": "Hokusai", "style": "Ukiyo-e", "style_image": "The_Great_Wave_off_Kanagawa", "output_resource_url": "http://160.40.51.32:10010/download/test/12975.jpg"}

{"simmo": "40450c43-8db9-416f-9ab9-32ced91cbf40", "frame": “0", "artist": "Joseph_Mallord_and_William_Turner", "style": "Romanticism", "style_image": "The_Shipwreck_of_the_Minotaur", "output_resource_url": "http://160.40.51.32:10010/download/test/12978.jpg"}

3.2.2 Capacity and performance

AE service is running in V4Design server in CERTH, consumes approximately 9% of NVIDIA GTX 1080Ti and typically takes around 15.77 seconds for each image or video frame. In reference to TP service, it consumes approximately 29% of NVIDIA GTX 1080Ti and the execution time is around 18.17 seconds for each image or video frame.

3.3 The Semantic Pipeline

The Semantic pipeline contains two distinct NLP-related parts: Language Analysis (LA) and Language Generation (LG). The input to LA is texts from the captions, descriptions, Wikipedia pages, online articles and reviews, etc. that are provided by the crawler (CR). It outputs semantic graphs that are sent and mapped to the Knowledge Base (KB), which performs additional reasoning and fuses the results with the output of other components (e.g. image analysis). The input to LG is KB substructures with the contents to be verbalized, and outputs textual reports. The following diagram shows the conceptual design of the Semantic pipeline.

Figure 13 The semantic pipeline

3.3.1 Internal architecture and interdependencies

The LA service contains 4 main components that apply sequentially

1. Concept candidate detection
2. Entity Linking, Word Sense Disambiguation
3. Surface dependency parsing
4. Semantic analysis (deep parsing + conceptual relation extraction)

The LG service contains 2 components that apply sequentially, the first one being only used in case of summarization (optional)

1. Content selection
2. Surface realization

3.3.2 Dependencies from other pipelines and interoperability

Both the LA and LG services communicate with the KB: LA feeds the KB, which in its turn feeds LG. Incoming data originates from the crawler (CR). Information from other pipelines (e.g. aesthetics) may be incorporated by the KB.

All communication with other components is performed by sending and receiving notifications through the message bus, and exchanging data through the data store, using its API. Mapping between the JSON structures used by the components and the internal data structures inside the data store (e.g. the KB) are performed at the data store API layer. The following table shows the communications involving the services of the semantic pipeline:

Table 7 Communication for semantic pipeline

Topic (listen)	Provided by	Provided to	Topic (send)
DATA_AVAILABLE	Crawler	Language Analysis (LA)	TEXT_ANALYZED
REASONING_ FINISHED	KB	Language Generation (LG)	TEXT_GENERATED

3.3.3 Examples of generated assets

The following are the Input/output examples of LA shown per specific concern.

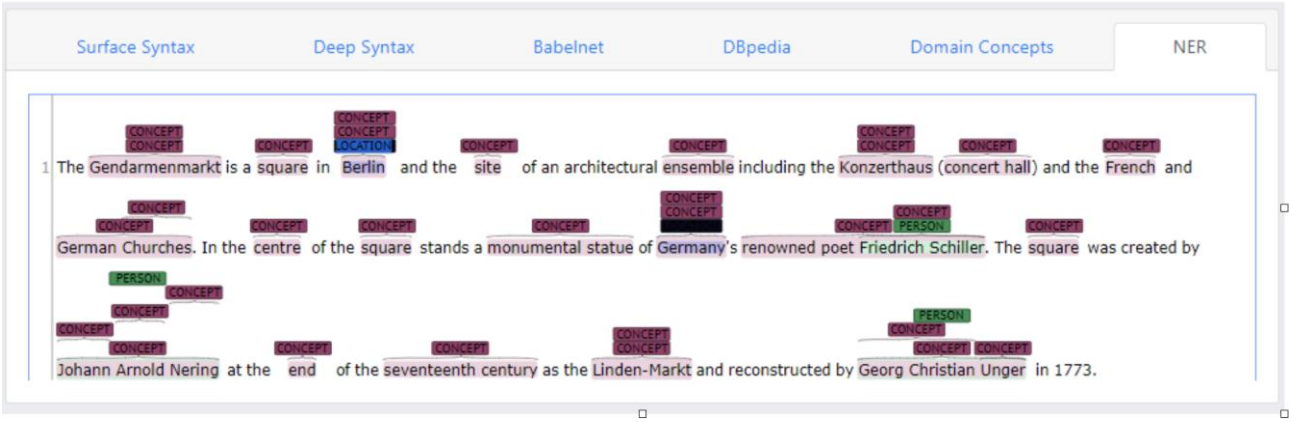


Figure 14 Concept candidate detection

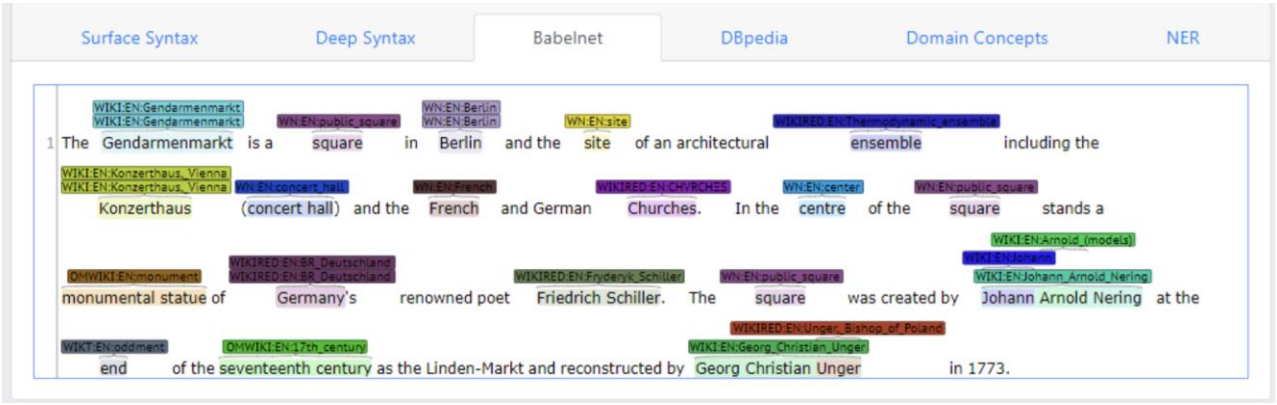


Figure 15 Word Sense Disambiguation

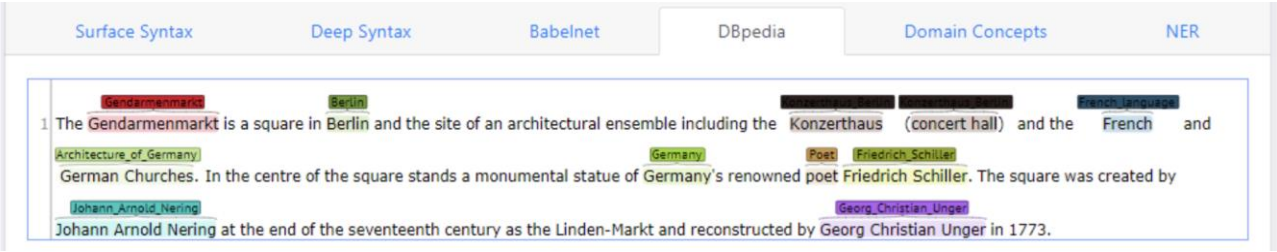


Figure 16 Entity linking

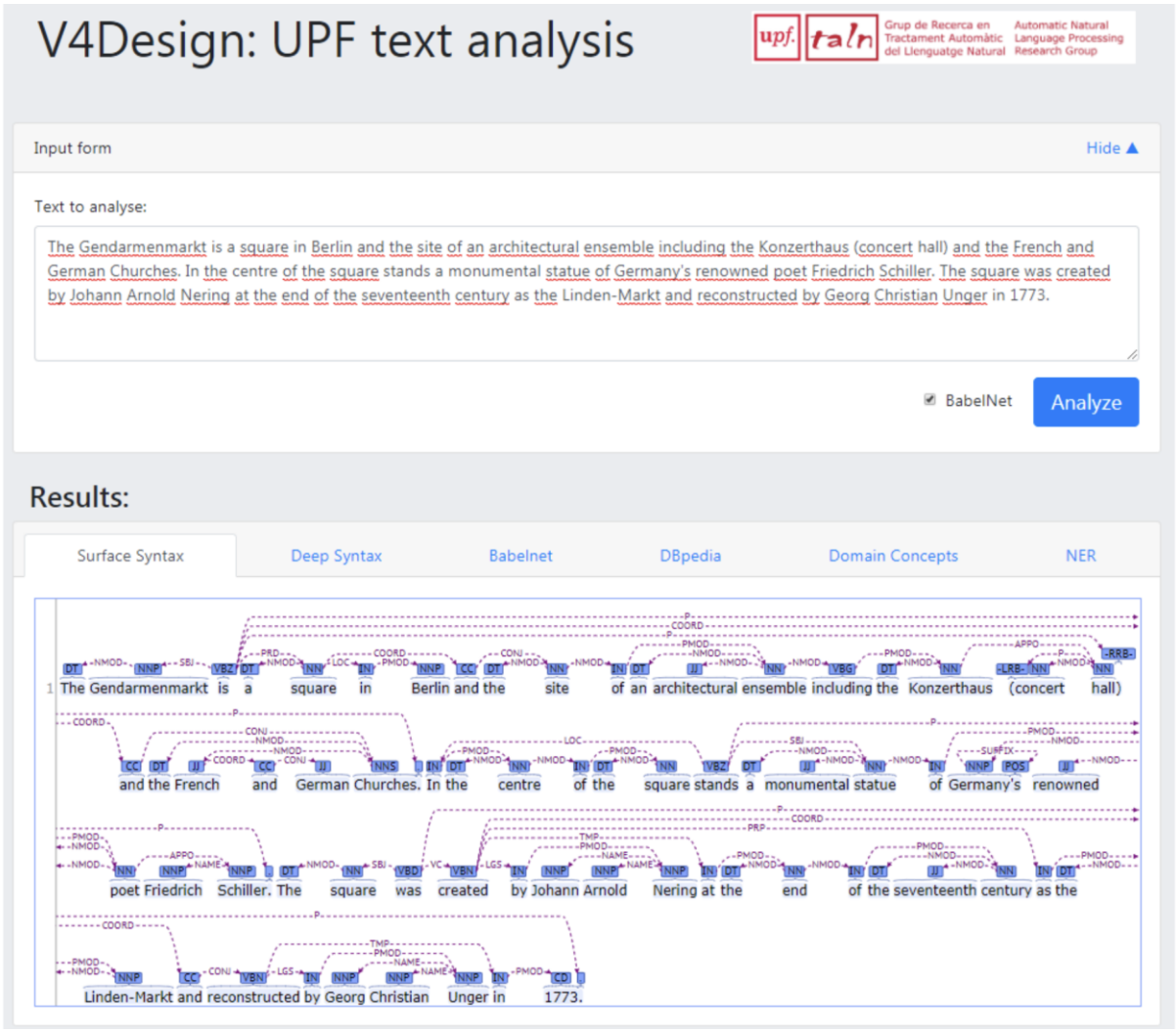


Figure 17 Surface-syntactic analysis

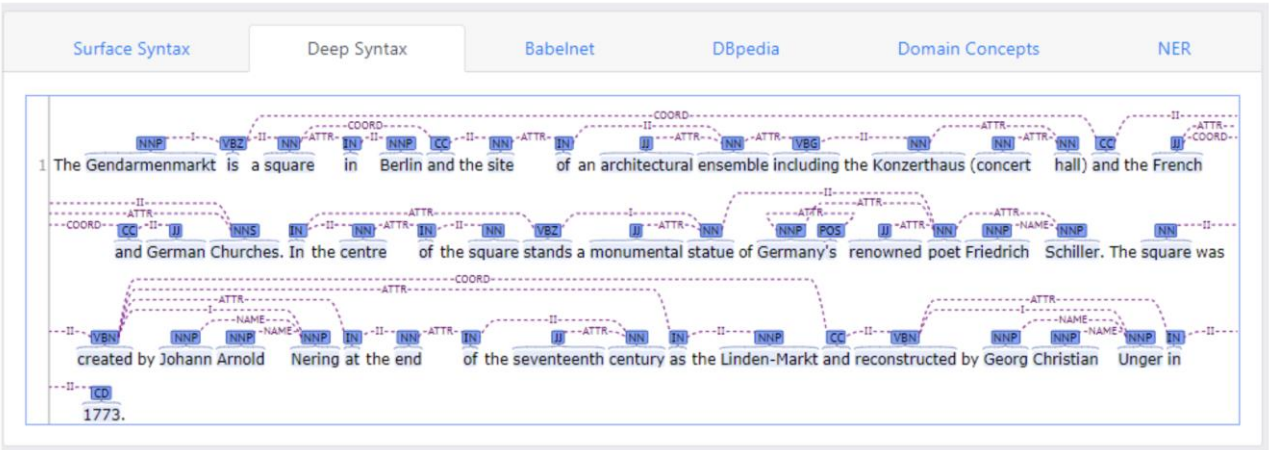


Figure 18 Semantic analysis

The following are output examples of LG for the input: Location (Berlin, Gendarmenmarkt)

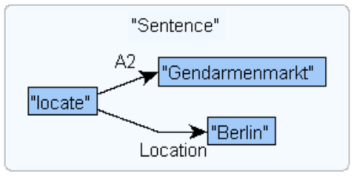


Figure 19 Mapping to predicate-argument structure

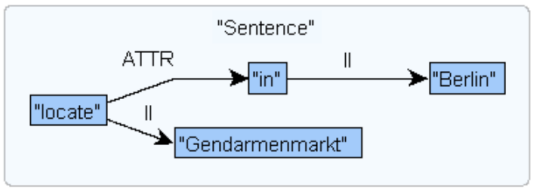


Figure 20 Mapping to deep-syntactic structure (sentence structuring)

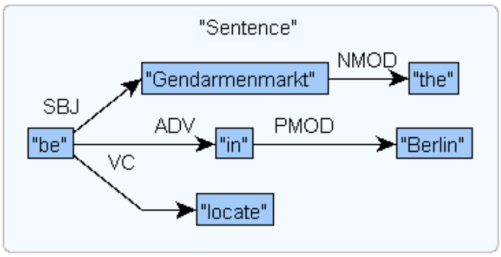


Figure 21 Mapping to surface-syntactic structure (introduction of functional elements and fine-grained grammatical relations)

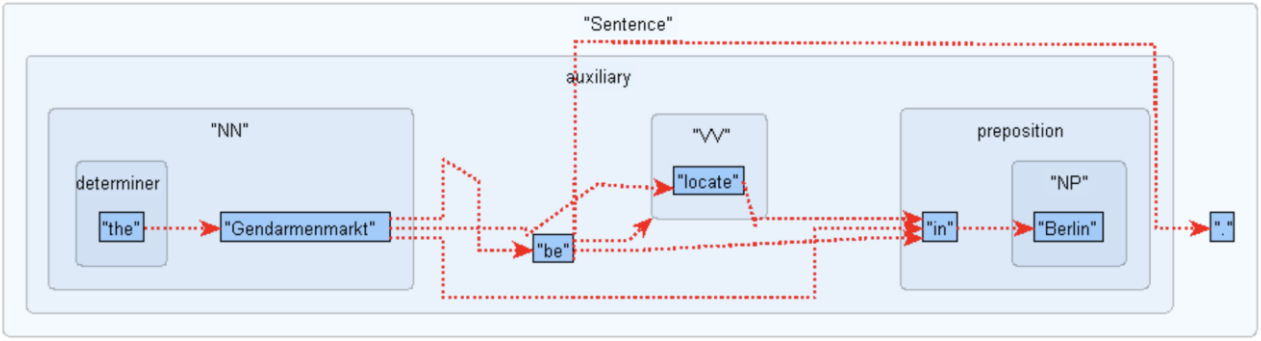


Figure 22 Linearization and introduction of punctuation signs

Figure 23 Resolution of morphological agreements

Figure 24 Surface form retrieval (output)

3.3.4 Capacity and performance

On a single machine TA typically takes around 10 seconds to a minute per document (depending on length), e.g.:

- 1000 characters: 12 seconds (full pipeline), 6 seconds (lightweight pipeline)
- 2000 characters: 17 seconds (full), 9 seconds (light)
- 6000 characters: 55 seconds (full), 15 seconds (light)

TG typically takes a few seconds to verbalize semantic structures (a dozen to a few dozen triples).

Both services can be easily scaled by increasing available hardware resources, both vertically and horizontally. They support concurrent requests within the limits of available RAM.

The following are the current technical limitations of the semantic pipeline:

- Candidate detection not operational in Greek.
- BabelNet disambiguation is slow on unseen data.
- Semantic analysis is not fully language-independent yet.

4 FRONTEND MIDDLEWARE AND END-USER TOOLS

In this section, the front-end middleware of the platform and its tools, both representing the part of the platform dedicated to supporting user interaction, are discussed. Namely, the REST API developed to tie the front-end components with the back-end services, the Virtual Reality tool, and the Architecture Authoring tool are described according to their status at M18. Demo videos for the tools can be found online[1].

4.1 The V4Design REST API (NURO)

The REST API tool developed is handling the connection of the backend services with the frontend tools. This provides a layer of security to the entire platform as the REST API by itself requires authentication to be accessed. The tool is connected to the backend DSRS and the message bus to relay messages to the other components of the system.

4.1.1 Supported functionalities

The REST API component can be easily integrated with the end-user applications as it is documented on Swagger [5]. The tool provides the following functionalities to the end user tools:

- User Functionalities related endpoints:
  - Create a new user
  - Logs the user in and creates a session
  - Gets information of an existing user
  - Updates the information of an existing user
  - Refreshes a running session to extend its expiration time
  - Ends a running session before it is expired
- Asset related endpoints
  - Creates a new asset in the database
  - Uploads the model file to the server
  - Updates the information of an asset in the database
  - Gets a specific asset
  - Gets the change history for an asset
  - Gets a list of the latest uploaded assets
  - Gets a list of assets with the matching tags
  - Gets a list of assets with a nearby location to a given address (e. g. a City)
  - Gets a list of assets with the matching reference dates
  - Adds a rating to an asset
  - Gets the ratings for an asset
- System functionality related endpoints:
  - Instructs the crawler to extract data from the specified url

[1] https://drive.google.com/open?id=1I8-gXfoGVWKWwmBRASRbfw3KtyAX12as

The documentation of the REST API on swagger helps in easy integration, as it can provide JSON file which can be integrated in the End-User tools. Figure 25, 26 and 27 show the users, assets and other functionalities documented on SWAGGER.

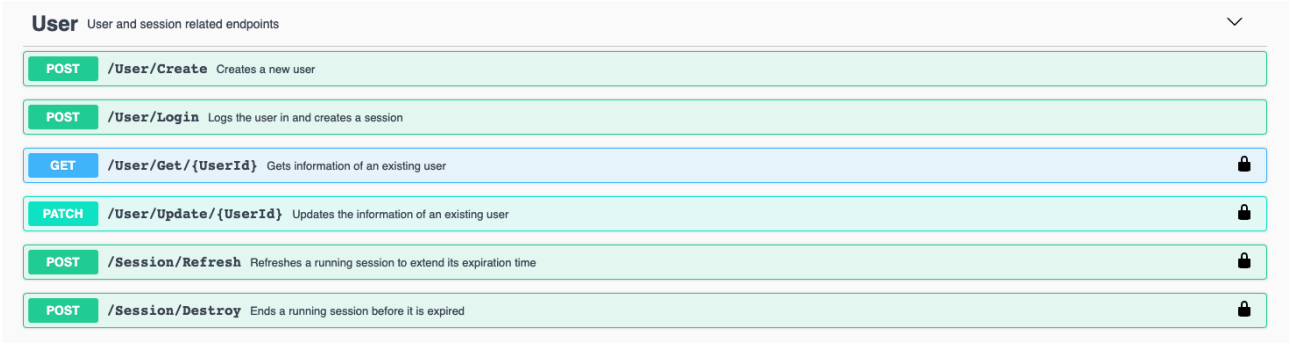


Figure 25 REST API - User functionalities

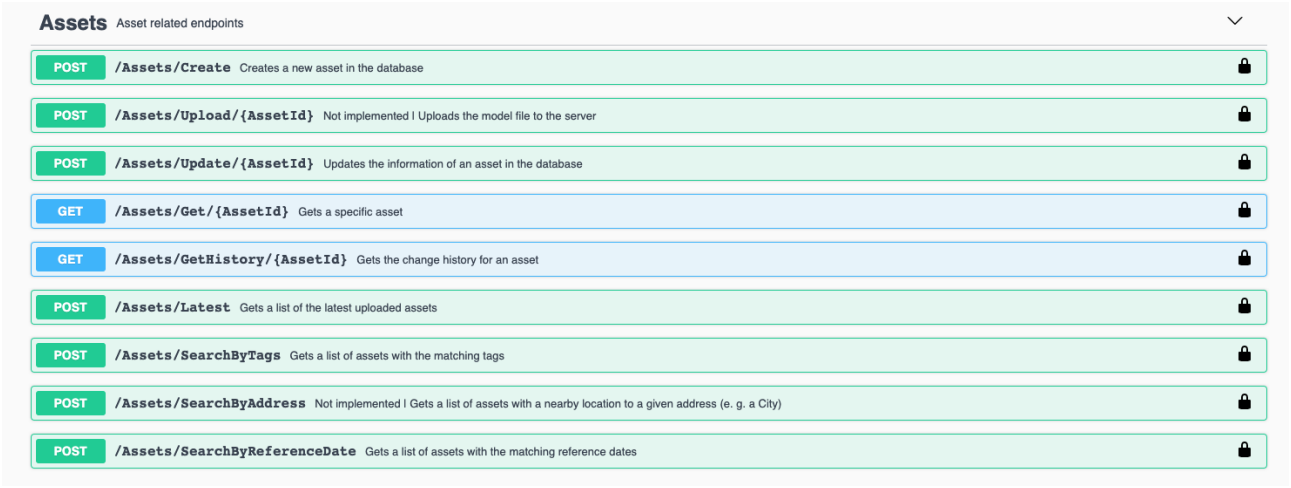


Figure 26 REST API - Asset functionalities

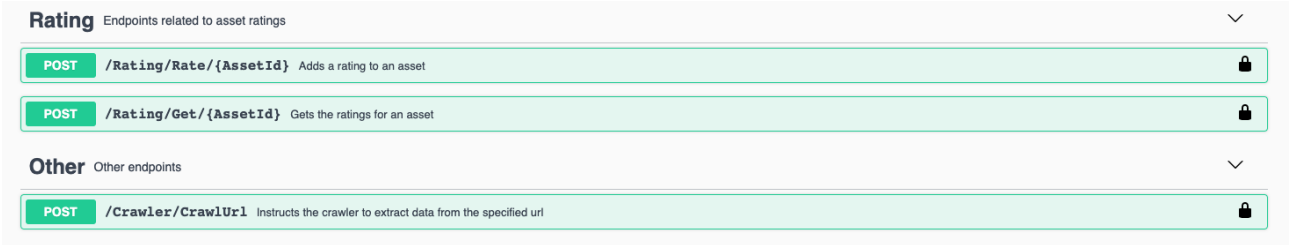


Figure 27 REST API - Other functionalities

4.1.2 Security concerns addressed

The REST API tool uses secure HTTP requests to handle the transfer of information from the end user tools to the system, therefore the system is secured by TLS (Transport Layer Security). Following that, the REST API tool uses an OAuth [6] token system to create a session ID that is needed to request any information from the server or send any information to the server.

4.1.3 Aspects contemplated for the next iteration of development

In the next version of the REST API tool, we envision the following:

1. User profile management

2. To be able to send messages to all the system functions as an admin
3. Proxy management
4. Add comments to 3D models
5. Version management for 3D assets

4.2 Virtual Reality Tool

The Virtual Reality tool consists of 2 tools, one the editor tool and another as a VR tool which can be used in the project. The tools allows game developers to import, view and use the assets generated by the V4Design backend. The assets can be imported just in the Unity3D project (using the Editor tool) or directly in a VR environment (Using the VR Tool).

Figure 28 and 29, represents the workflow of the Editor and VR Tool respectively.

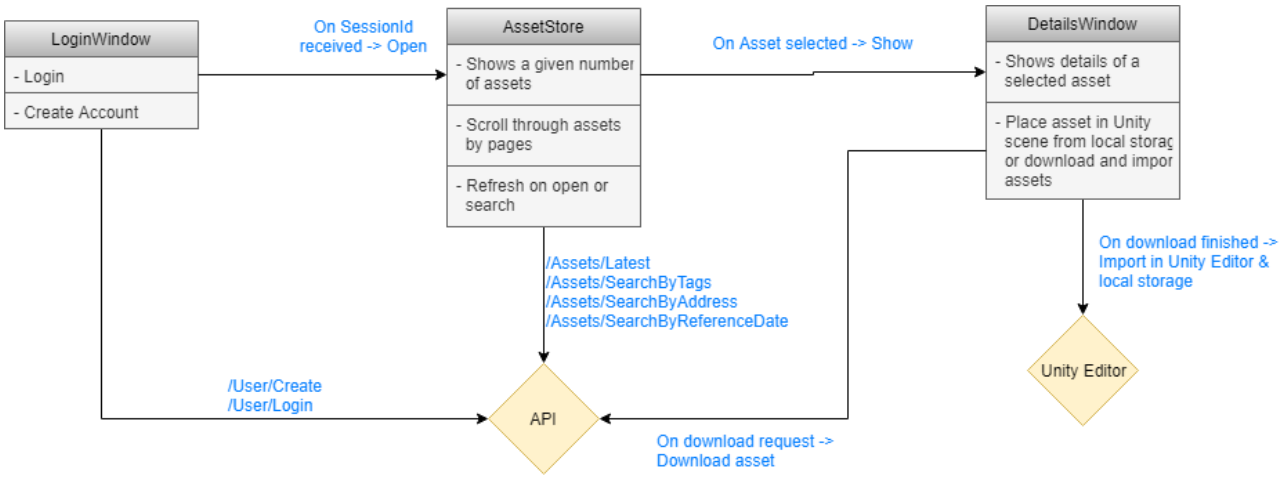


Figure 28 Workflow of the Editor App

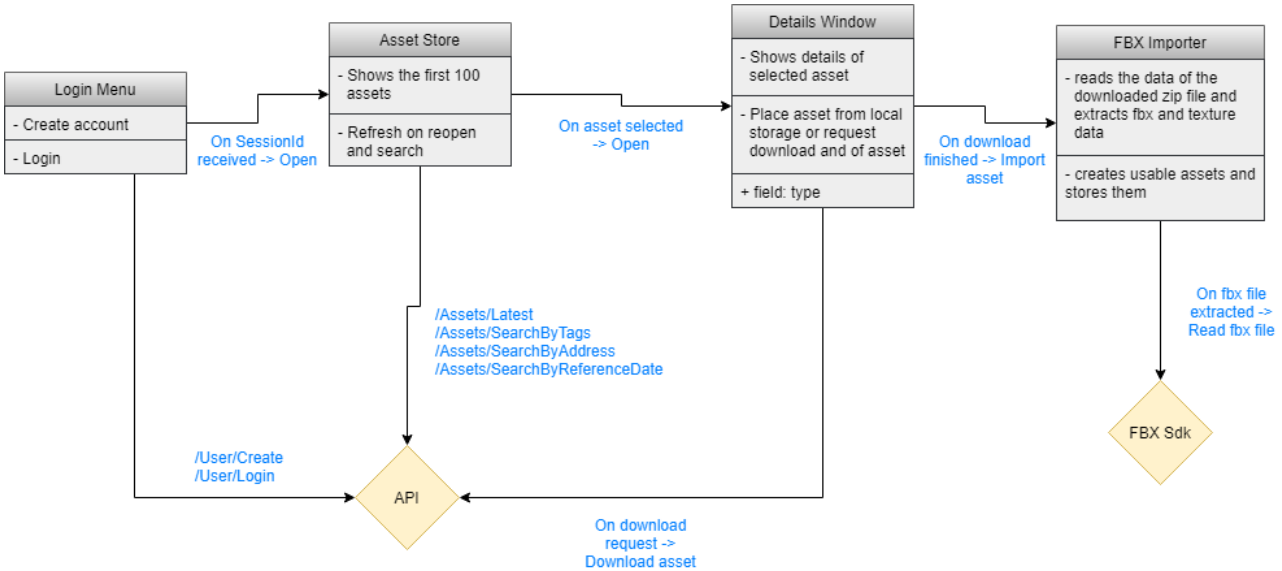


Figure 29 Workflow of VR Tool

Figure 28 shows the 3 major interfaces of the Unity Editor tool. This tool is used when the developer wants to use the V4Design application without being inside VR. This helps in the integration of Unity tools in the work environment apart from just V4Design features. The tool consists of the first interface for login and account creation, this is connected to the REST API tool using RESTful APIs and takes a username and password as input. The Asset Store interface, gets a specific number of assets from the backend and displays the assets. It creates pages for the assets to be shown in and includes the search functionalities. The third interface in this tool displays the assets and is linked to the Unity Libraries to download the asset from the backend and place it inside a Unity scene.

The second tool, as shown in Figure 29, is the VR tool which consists of 4 interfaces/modules. The first module of this is for login and account creation; this is connected to the REST API tool using RESTful APIs and takes a username and password as input by the Virtual keyboard inside the VR app. The second module gets the assets from backend, it gets 100 assets at once and displays them and the user can request for more, this component also allows for search of assets. The third module displays the information of the asset from the backend and connects with the FBX importer, the fourth module, to import and unzip assets directly from the V4Design servers to the VR environments in real time. The FbX importer also connects to the FBX SDK for importing functions. The tool also consists of a questions importer that allows the user to add questions to these assets, that when answered correctly, deletes the asset from the VR scene. The questions are imported form the Questions database created for V4Design and hosted by Nurogames.

4.2.1 Demonstration scenario for the Virtual Reality Tool

A demonstration scenario was designed and implemented to show how the tool's concept addresses the requirements of the users, namely in asset search and retrieval, and their insertion into a Unity3D project.

The scenario starts by a user creating an account on Unity3D with a username and password. Then the user is able to login to the system on Unity. This would start a session for the user and the tool is responsible to manage the session.

The user is then able to see the assets in the Editor tool and view their information. The user also has the ability to search through the assets and get the relevant asset from the backend. Searching by tags allows for a lot of flexibility in the searching scenario. Once the users find the correct asset, they can view all the information about the asset and read the description coming from the Text analysis/generation modules (WP3).

Once the user wants to download the asset, they can just press download and the asset will be placed in the current Unity3D scene they have open, just like other unity assets. This will allow the user to quickly create an environment that can run in VR and without VR.

Following this, the demonstration presents a similar approach in VR, where the user is also able to login through the VR application to get the assets from the V4Design asset store. The user can see all the information about the assets and press on place to directly place them in the VR environment. Once "Place" is pressed, the tool automatically uses the unzipper and the FBX importer to import the asset from the V4Design servers to the Unity VR environment in real-time. The user can then add questions to these assets, that when answered correctly, delete the asset from the VR scene. The questions are imported from the Questions database.

The completion of the entire scenario will help us identify the possible usability and functionality issues.

Figure 30 Screenshots of the demo scenario

4.2.2 Description of the envisioned second version

We plan to integrate more functionalities in terms of possible commands that the end user can send to the backend including the ability to send a link of a video to crawl data from. Moreover, we plan to integrate more formats and metadata to be shown in realtime in the VR application. Lastly, more APIs will be included in the upcoming iterations.

4.3 Architecture Design Tool (McNeel)

The Architecture Authoring tool is designed primarily as a content and process management environment through which users can access the V4Design platform and retrieve assets from its storage. The tool also allows users to send requests to the platform to process specific sources for assets. When the assets sought after are retrieved by the user, they can be imported into a CAD design environment developed on top of Rhinoceros 3D.

The prototypical version of the tool implemented and delivered at M12 has showed how assets are displayed and then imported to the CAD environment. An example 3D model with fake data was used in order to demo this simple process.

Technically, the current version of the Architecture Authoring tool, named version 1, is fully integrated with the platform's REST API, and uses it to communicate with the V4Design platform backend. This integration not only allows the tool to access data from the platform, but also to use the authentication and session management functionalities developed in the API. This integration is explained in the following diagram.

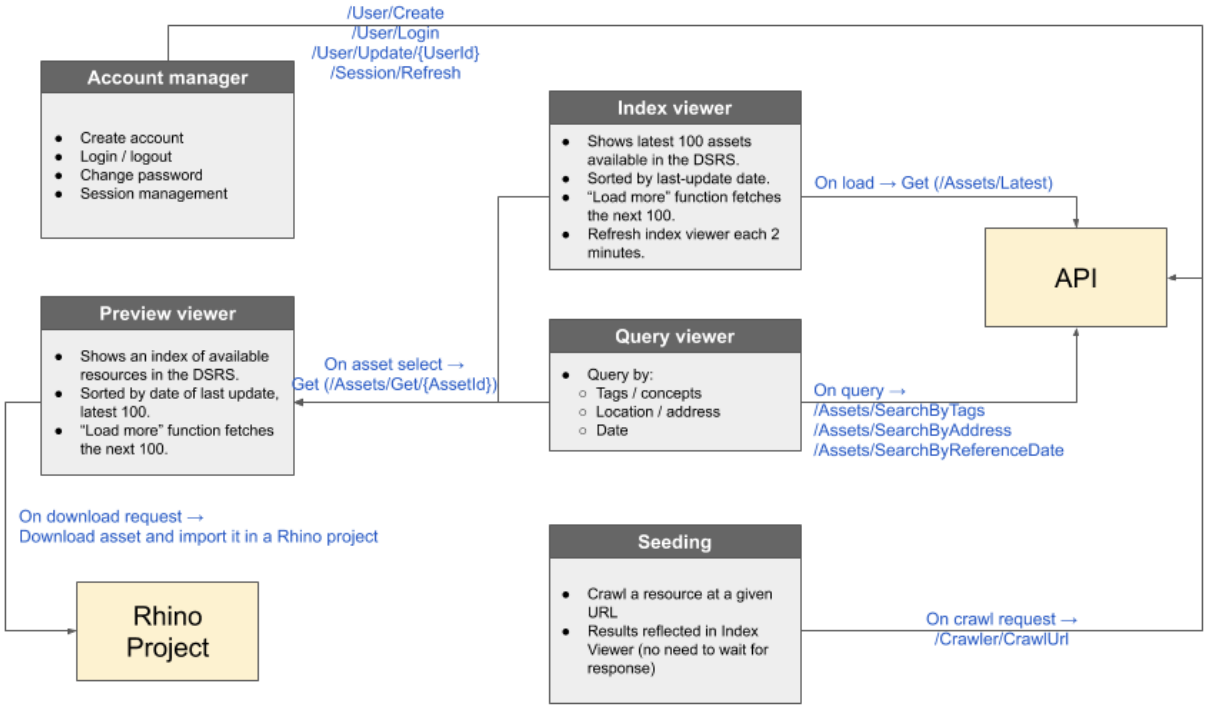


Figure 31 Authentication dialogue integrated

The diagram also reveals five new interface components implemented to support the user processes currently included in the tool. The account manager component takes care of managing the user authentication and user session, allowing the tool to distinguish between asset owners and store feedback from each user effectively. The index viewer initially shows the latest assets available in the V4Design platform, and items can be previewed directly from this component, and then imported into any Rhino project. Similarly to the index viewer, the query viewer allows the user to query assets by tags, location, and date, and then preview them before importing them. The preview viewer component shows the thumbnail image of an asset and its related metadata descriptions, which include high-affinity descriptions. Finally, a seeding component was developed to allow users to send requests to the platform.

This preliminary module is the basis for empowering the user to direct and apply the functionalities of the platform onto a selected set of media items, identified or retrieved by the user. The current implementation sends a URL to be crawled for potential assets.

4.3.1 Demonstration scenario for the Authoring Tool

A demonstration scenario was designed and implemented to show how the tool's concept addresses the requirements of the users, namely in asset search and retrieval, and their insertion into CAD projects in Rhinoceros 3D.

The scenario starts by authenticating the user, or creating a user account for newcomers. In both cases, the tool interfaces with the REST API and uses its authentication mechanism. Authentication also enables session management, in which a session is created and maintained while the user is actively using the tool. When a session is dropped, the user has to login again, although this could be later managed by using local storage or cookies as deemed convenient in the upcoming development cycle. The following figure 30 shows the authentication dialogue on top of the tool's editor.

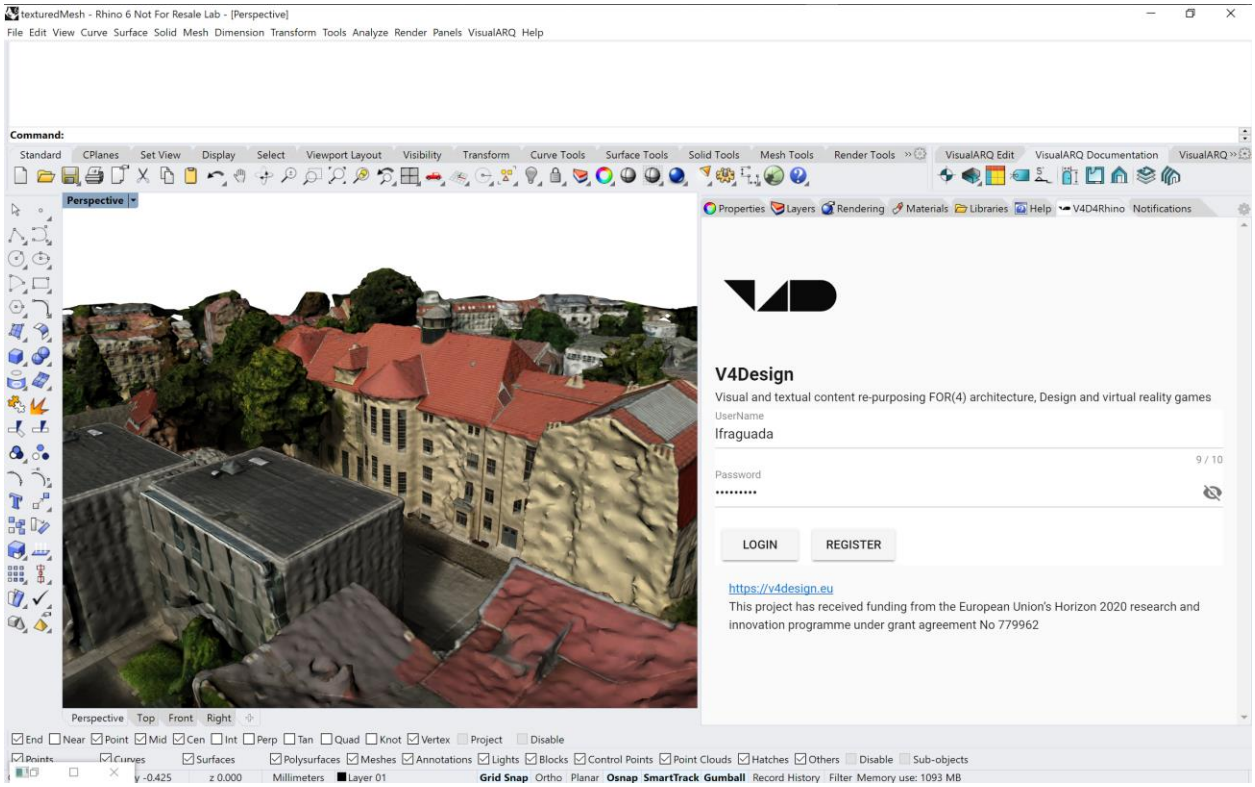


Figure 32 Authentication dialogue integrated

After logging in and creating a session, the user accesses the "latest assets" view, which shows the most recently generated assets in the DSRS, which are buffered by the REST API. Alternatively, the user can query for assets directly, using concepts, locations, and dates. Under this, the tool sends a query request to the API, which channels it back to the DSRS and retrieves the results. In the authoring tool, the results are shown in the same manner as the "latest assets" view.

In order to import an asset into the Rhinoceros 3D editing environment, the user has to download it first. For that, the user needs to select the asset from the list of results. This

selection opens an “asset details” view, showing the asset’s tile, thumbnail, metadata, and other related information. Upon inspecting this information, the user can choose to download the asset and use it in the tool. The following figure 33 shows one imported asset alongside the “asset details” dialogue.

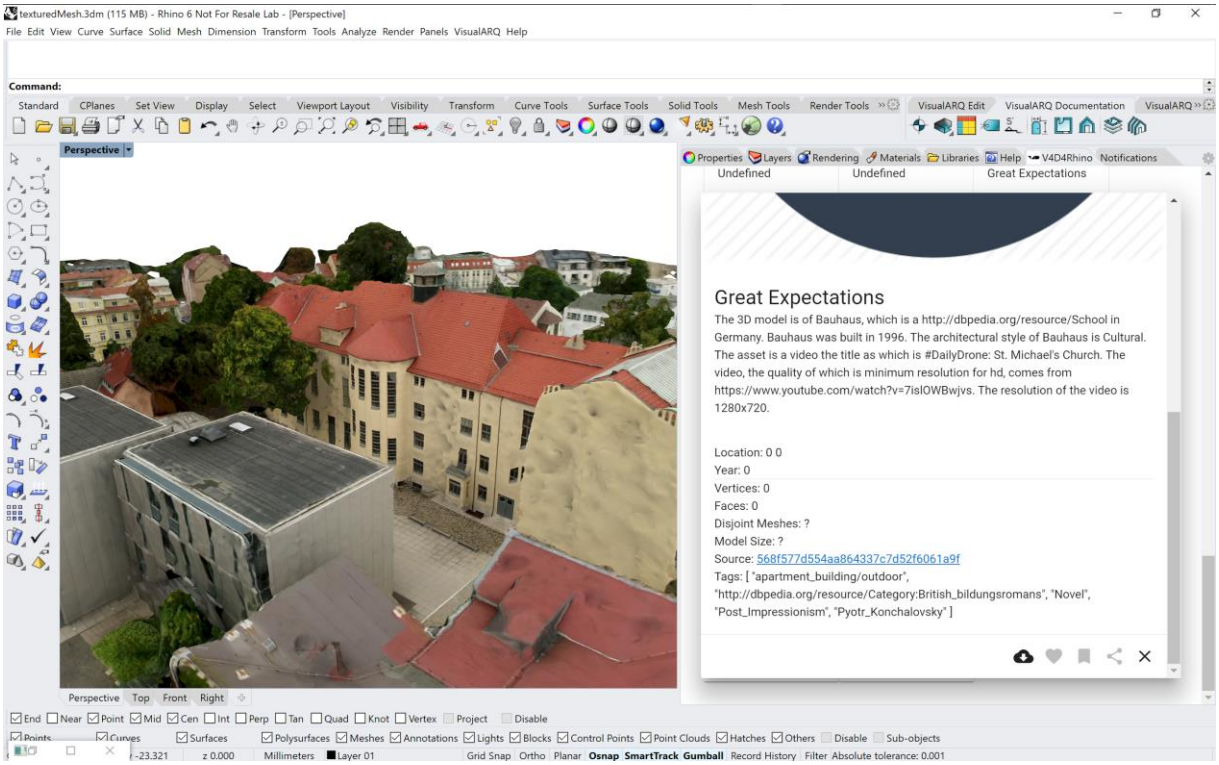


Figure 33 Imported asset alongside the “asset details” dialogue

In addition to this scenario in which users are capable of finding, retrieving and using assets generated by the V4Design platform, an extension was implemented to show how these assets can be fully incorporated in the Rhinoceros 3D work cycle, including editing the asset computationally with the Grasshopper component (see Figure 34), which demonstrates the power of Rhinoceros 3D in manipulating generated 3D assets, and the compatibility of the generated assets with this environment.

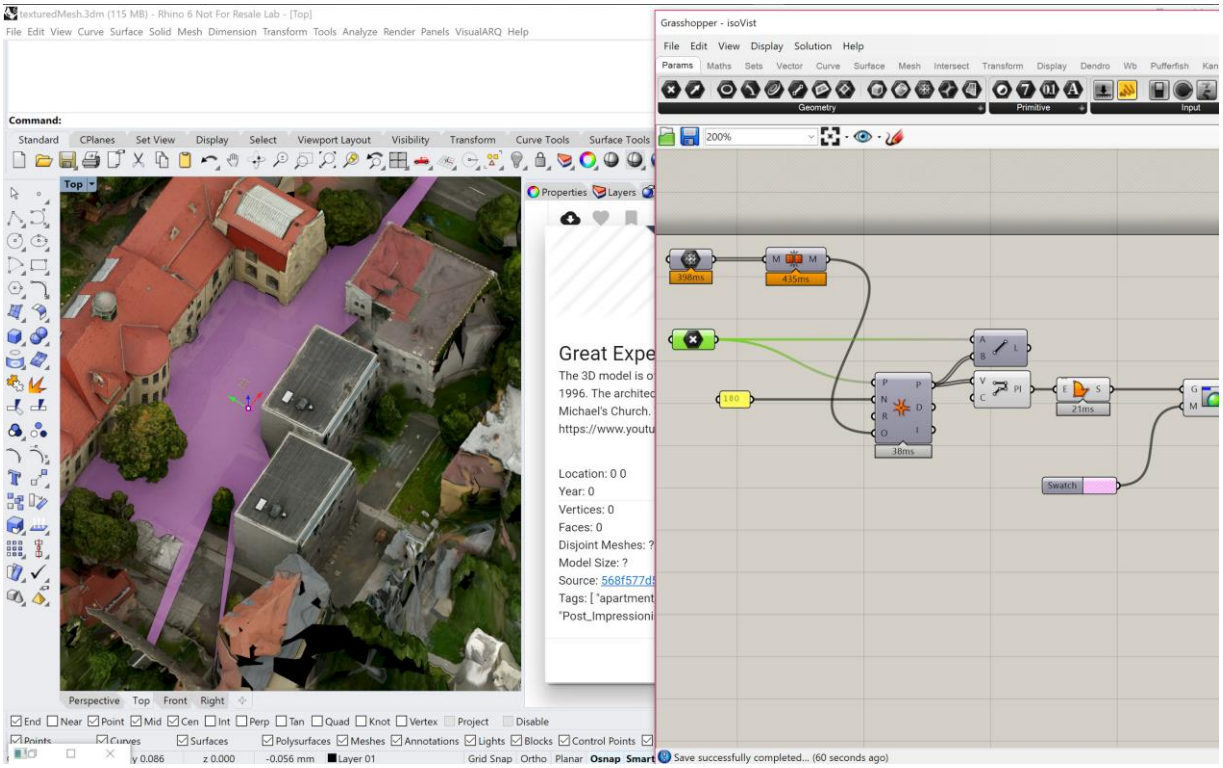


Figure 34 Manipulating assets in Grasshopper

4.3.2 Description of the envisioned second version

In the following development iteration, efforts will be invested in developing further the tool's functionalities. This includes a consolidation of the current components, namely streamlining the tool's ability to retrieve any type of asset effectively, and extending the seeding capacities to include: 1) selected items from the repositories of content providers, members of the V4Design consortium; 2) selected links that represent a collection of media items that the user personally curated; 3) crawl criteria that includes subjects, locations, and patterns to look for. In addition, the content providers' APIs will be integrated in the tool so as to permit the users to brow.

5 FUNCTIONAL TESTING AND EVALUATION

The communication, data management, and synchronization models were devised based on an in-depth understanding of the functional requirements and technical specifications associated with the V4Design process. Nonetheless, their adequacy and performance should be functionally tested to discern how the process is implemented in a practical sense, where different aspects (e.g. connections, data format, message handling, queuing, etc.) may influence how the process actually works.

Therefore, a comprehensive functional evaluation was planned and executed after the initial integration of the V1 platform prototype.

5.1 Data collection for the functional evaluation

In order to gather the required raw data for the functional tests of the pipelines and other modules of the integrated V1 platform prototype, the following scheme was collaboratively drafted to describe the required data for content providers, taking into account the current limitations of the services and their pipelines in the way they implement the technology that each encapsulates:

- Items are each composed of a short high-quality video showing a specific scene or object worthy of 3D reconstruction.
- Overlays, such as text or subtitles or others should be avoided.
- Each item should be accompanied with a set of metadata that represents the associated information available on that item in the content provider database.
- Each content provider should select 20-25 items from their collection, and manually curate the associated metadata, using the actual metadata as reference.

Accordingly, each content provider curated a selection of items for processing by the V1 platform prototype. A quality inspection of the items allowed to discard those that appear to include problematic concerns or aspects deemed incompatible with the current maturity status of the technologies implemented by the services. After this inspection, a set of 64 items was selected as the main raw dataset for the functional tests (see appendix A).

It is worth noting that the content curation and quality inspection tasks took into account the relevance of the selected items to the use cases, in terms of subject matter.

5.2 Procedure for the functional tests

The goal of this functional evaluation is set to generate a set of 12-16 assets enriched with metadata and associated components, sufficiently sophisticated as to represent the type of assets that the V4Design platform is ought to generate. At the core of each asset is a 3D model, whose successful reconstruction is key to acquire the asset. Without the model, the asset reconstruction is deemed unsuccessful even if other pipelines (aesthetics and semantics) performed flawlessly.

Given that the pipelines can execute in parallel and virtually independently from one another, and taking into account that the tests conducted should shed light on the performance, capacity, and function of each pipeline and its elements separately (to the extent possible),

the tests first proceeded by running each pipeline independently. Later, several pipelines were ran in parallel.

A set of instructions were devised for pipeline owners and operators to use the message bus for running their pipelines independently, and verify first-hand the results obtained and inspect the faults and errors incurred. The following is an example of instructions shared to run the semantic pipeline (see appendix B for a complete list).

> You need to simulate a DATA_AVAILABLE message that launches the semantic pipeline. To do so, go the message bus API, and send a DATA_AVAILABLE topic message with the following parameters (make sure you know the SIMMO ID of the item you would like to run.
>
> | | |
> |---|---|
> | URL | https://34.253.156.62:8162/admin/send.jsp?JMSDestination=DATA_AVAILABLE&JMSDestinationType=topic |
> | PARAMs | [
{
"id":"e0605ee7-fa49-4465-a378-f2e23ec9d3b5",
"has_texts":"true",
"has_images":"false",
"has_videos":"false",
"entity":"webpage"
}
] |
>
> In order to execute the semantic pipeline exclusively, set has_images and has_videos as false even if the SIMMO contains such items. The pipeline is expected to run at least one LG cycle.

The pipeline owners were asked to run the raw data through their pipelines, one item at a time, and to track their progress in a shared handbook that shows what element has been processed by which pipeline and the results achieved. As soon as the results of the processing became available, they were inspected for completion and quality, and if they meet the required threshold then they were listed under the generated data set. In some cases, items were reprocessed to test if specific errors or quality aspects have been addressed.

5.3 Results of functional tests

In general, the results of the tests reveal the dynamics of the platform in relation with input data, and its ability to cope with different input, in terms of input quality, content, metadata, and other aspects.

In terms of 3D reconstruction, more than around 60% of the available data was processed, generating 20 3D models of varying quality and size. During this data processing, potential weaknesses and incompatibilities in the implemented 3D reconstruction process were analysed, and some were addressed during this development cycle. Most of these limitations were already known, but the data processing offered a window on to their dynamics. For instance, the duration and the inherent camera motion inside the video can influence the

quality of generated models, but this does not render the resulting models useless or unusable. To the contrary, the reconstruction algorithm is able to generate models with relevant features (e.g. correct measurements and proportions, accurate sections, etc.). Also, the presence of textual overlays in the videos can cause an unmitigable effect on the outcome, but segments of videos that do not contain overlays can still be selected and used. The tests also allowed to understand how to design a user-driven process of empower 3D reconstruction and provide it with well-selected and suitable input. Also, the results show that it is possible to predict the quality and other aspects of the outcome based on a visual analysis of the input video.

In terms of semantic analysis, almost 95% of the available data was processed, generating descriptions and metadata where raw textual information is associated with the video items. The tests carried out show that the semantic pipeline is consistent in its response times for both English and for Spanish. Two versions of the analysis pipeline have been tested: without concept ranking (for summarization): ~7 seconds for 30 words, ~14 seconds for 300 words; with concept ranking: ~10 seconds for 30 words, ~45 seconds for 300 words. Currently, the Semantic Analysis pipeline caches part of the results, so some queries can get faster. Also, one current limitation is that it is possible to get timeouts for texts that take a long time to process, depending on the configuration of the client making the request to the analysis service. Language generation consistently answers back with a description in about 10 seconds. All items from the V4Design repository were processed successfully with no technical issues. However, especially on short texts such as image captions, it may happen that no useful information can be extracted (e.g. because the text does not contain any relevant entities).

In terms of the Aesthetic pipelines, all data from V4Design repository were processed with no technical issues. The tests carried out show that the Aesthetics pipeline is consistent in its response times for both Aesthetics extraction and Texture proposals services. The present execution time for a video from V4Design collection, which has 1210 frames is about 12 seconds per frame for the Aesthetics extraction module and about 17 seconds per frame for each style for the Texture Proposal module.

6 CONCLUSIONS

In this document we discussed the status and characteristics of the V4Design first prototype, which includes the developed and integrated technological services and the authoring tools for architecture and game design.

The evolution from the experimental prototype developed at M12 to the 1st prototype was discussed, revealing how the communication model was adapted, how the data management was implemented in the backend, and how different aspects of the platform were synchronized.

According to the description of the V1 architecture, three main pipelines were defined and consolidated. The 3D Reconstruction pipeline streamlines the process of extracting 3D models from media; the Aesthetic pipeline streamlines the aesthetic analysis and extraction from raw data; and the semantic pipeline streamlines the generation of descriptive rich content using semantic analysis. The functions and limitations of each of these pipelines was discussed, and a sample of results shown to exemplify the current maturity level of the implementation.

On the front-end side, the platform API, which was developed to connect the tools with the backend is discussed, with major functionalities revealed. In addition, the first versions of the authoring tools were also discussed, showing the new implemented features and the working functionalities at this stage of the project. The future plans for developing further the tools were explained.

Finally, the conducted set of functional tests to validate the integration and maturity of the 1st V4Design platform prototype was discussed. These tests have helped to consolidate further the prototype and also to trace future development actions more effectively.

At this stage in the project, preliminary user evaluations are planned in order to expand the assessment of the 1st prototype, and to illustrate further the requirements for the second prototype in terms of user interaction, and the ecosystem of usage that would surround any potential deployment in the near future.

7 REFERENCES

[1] D6.1 Roadmap towards the implementation of the V4Design platform. Deliverable published under WP6 in V4Design.

[2] D6.2 Technical requirements and architecture. Deliverable published under WP6 in V4Design.

[3] D6.3 Operational prototypes and user interfaces for architecture and VR game design application. Deliverable published under WP6 in V4Design.

[4] Tsikrika, T., Andreadou, K., Moumtzidou, A., Schinas, E., Papadopoulos, S., Vrochidis, S., & Kompatsiaris, I. (2015, January). A unified model for socially interconnected multimedia-enriched objects. In International Conference on Multimedia Modeling (pp. 372-384). Springer, Cham.

[5] Swagger framework: https://swagger.io/

[6] Oauth token system https://oauth.net/

APPENDIX A: CURATED DATA COLLECTION

The following table shows the data items shared by content providers in order to perform the functional tests on the integrated platform, and prepare the required dataset for the user evaluations.

Dataset name	Owner(s)	Type of items	Number of items	Collections	Format of items
PUC1 - Sc1	EF	Images + descriptions in single-huge json	340	8	JPEG
PUC1 - Sc2	AF	Images (no descriptions)	33	1	JPEG
PUC1 - Sc2	EF	Images + descriptions in single-huge json	2650	18	JPEG
PUC2-AF	AF	Images organized in folders (no descriptions)	1500	10	JPEG
PUC2-EF	EF	Images + descriptions in single-huge json	2550	15	JPEG
PUC4	EF	Images + descriptions in single-huge json	5	5	JPEG
Download	EF	Images + descriptions in single-huge json	4350	12	JPEG
Feb2019 New batch	EF	collections of images, with several jsons (8) for descriptions	24000	14	JPEG
DW_august	DW	Images and videos (no description)	14	1	JPEG, MP4
DW_Mesh	DW	real scanned and fitted 3D model			
Flickr	CERTH	Only images	53740	137	JPEG
Wiki	CERTH	Only images	41	1	JPEG
Symmetrical Items	CERTH	Images with large jsons for descriptions	2500	4	JPEG
Videos-AF	AF	Timestamp available for most of the videos on excel on FTP same folder as the videos.	57	8	MOV, MP4
Videos-AF	AF	Film clips 1-4 mins long, suitable for 3D extraction on FTP together with Metadata	30		MP4
Videos-SfP	SLRS	Videos (full length documentaries), some with frames	3	3	MP4
Videos-SfP-Shots	SLRS	Videos (shots for 3D reconstruction)	5	1	MP4

Videos-DW	DW	Videos (drone footage for 3D reconstruction)	32	1	MOV
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4

Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Single pan shot (flyby, flyover, flyaround)	1	1	MP4
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	4	1	MOV
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	3	1	MOV
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	5	1	MOV
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	10	1	MOV
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	1	1	MP4, HLS
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	3	1	MP4
Videos-DW	DW	Videos (drone footage for 3D reconstruction)	5	1	MP4
Videos-DW	DW	360 video publishes on Facebook (DW Travel)	1	1	Video, description
Videos-DW	DW	Video series of drone footage	237 (ongoing)	1	Video, tags

APPENDIX B: INSTRUCTIONS FOR RUNNING THE V4DESIGN PIPELINES

The following instructions have been collaboratively composed by the technical partners to guide the testing and execution of different pipelines.

Asking for a crawled resource to be reprocessed:

Go to the message bus API, and send a CRAWL_IT topic message with the following parameters.

URL	https://34.253.156.62:8162/admin/send.jsp?JMSDestination=CRAWL_IT&JMSDestinationType=topic

PARAMs	{
	"type": "SIMMO",
	"entity": "webpage/image/video",
	"value": "e0605ee7-fa49-4465-a378-f2e23ec9d3b5"
	}

When the Crawler sees that the type is an existing SIMMO, is does not crawl it but launches a DATA_AVAILABLE message with the item's details as if it were crawled from the web.

Asking to crawl a resource:

Go to the message bus API, and send a CRAWL_IT topic message with the following parameters.
Please note that only type "webpage" will be supported for now.

URL	https://34.253.156.62:8162/admin/send.jsp?JMSDestination=CRAWL_IT&JMSDestinationType=topic

PARAMs	{
	"type": "webpage",
	" value": "www.webpageaddress.com"
	}

Running the semantic pipeline:

You need to simulate a DATA_AVAILABLE message that launches the semantic pipeline. To do so, go to the message bus API, and send a DATA_AVAILABLE topic message with the following parameters (make sure you know the SIMMO ID of the item you would like to run.

URL	https://34.253.156.62:8162/admin/send.jsp?JMSDestination=DATA_AVAILABLE&JMSDestinationType=topic

PARAMs	[
	{
	"id":"e0605ee7-fa49-4465-a378-f2e23ec9d3b5",
	"has_texts":"true",
	"has_images":"false",
	"has_videos":"false",
	"entity":"webpage"
	}
	]

In order to execute the semantic pipeline exclusively, set has_images and has_videos as false even if the SIMMO contains such items. The pipeline is expected to run at least one LG cycle.

Running the 3DR pipeline:

You need to simulate a DATA_AVAILABLE message that launches the 3DR pipeline. To do so, go the message bus API, and send a DATA_AVAILABLE topic message with the following parameters (make sure you know the SIMMO ID of the item you would like to run.

URL	https://34.253.156.62:8162/admin/send.jsp?JMSDestination=DATA_AVAILABLE&JMSDestinationType=topic

PARAMs	{
	"id": "e0605ee7-fa49-4465-a378-f2e23ec9d3b5",
	"has_texts": “false”,
	"has_images": “false”,
	"has_videos": “true”,
	"entity": "webpage"
	}

Keep in mind that it is difficult to execute the 3DR pipeline exclusively. Each OBJECT_RECONSTRUCTED message will launch the Aesthetic pipeline, unless it is disabled from the 3D Reconstruction service. Also, disable the ON_UPDATE message from KB in order not to invoke LG.

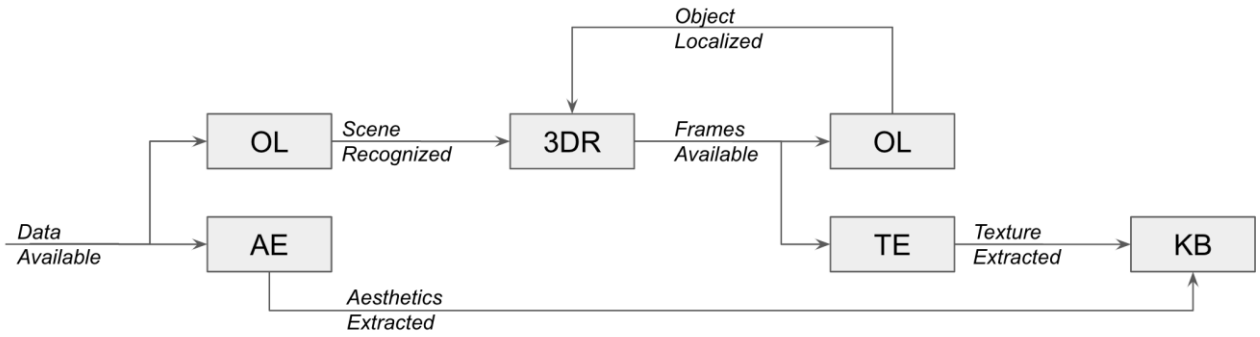
Running the Aesthetics pipeline:

You need to simulate a OBJECT_RECONSTRUCTED message that launches the Aesthetic pipeline. To do so, make sure that you are using an example of a 3D model reconstructed by V4D. Go the the message bus API, and send a OBJECT_RECONSTRUCTED topic message with the following parameters.

URL	https://34.253.156.62:8162/admin/send.jsp?JMSDestination=OBJECT_RECONSTRUCTED&JMSDestinationType=topic

PARAMs	{
	"**asset_id**": "e0605ee7-fa49-4465-a378-f2e23ec9d3b5"
	}

In order to test this pipeline separately, without invoking LG when. Disable the ON_UPDATE message launched by KB.



Retrieving the data generated by your pipeline:

Remember that you CANNOT read the messages circulated by the services from the message bus API, but you can do so from the REST API.

The current API interface is the Swagger:

https://v4design-integration.nurogate.com/Server/Swagger/#/

You can create your own API user and log in to execute the calls.

Basic API usage

- Use /User/Create or /User/Login (open the endpoint > click "try it out" > fill out the required fields > click "execute")
- Swagger-UI uses the API in a regular way just as any other application has to do, so this can be used to see how requests have to be formatted etc.
- In the response there will be the field "SessionId", the value of that field is required in the next step
- Click "Authorize" in the top left of the Swagger-UI
- In the popup fill in the session id that you got from the response and then click "Authorize" to confirm
- The Swagger-UI will now send the session id as http header "X-V4DesignSession" on each request, this is also the way other API clients have to send the session id. the name of the header is also noted in the popup that opened to fill in the session id in the previous step

Things to keep in mind when using the API

- Dates have to be formatted as “YYYY-MM-DD” as defined by the swagger standard (see RFC 3339, section 5.6[2], full-date notation)
- DateTimes have to be formatted as “YYYY-MM-DDThh:mm:ss+xx:yy” as defined by the swagger standard (see RFC 3339, section 5.6[3], date-time notation)
- It is not possible to update assets that have been automatically generated by other components using the /Assets/Update/{AssetId} endpoint because it is currently not defined how the API should handle these modifications

[2] https://tools.ietf.org/html/rfc3339#section-5.6

[3] https://tools.ietf.org/html/rfc3339#section-5.6

APPENDIX C: SOURCE CODE AND DEMOS

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.

Table 8 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/v4d-text-integration and 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/TP	Apache Licence v2.0
KB Population and Reasoning	Protected	V4Design code repository: https://gitlab.com/v4designEU/saveinkb	Apache Licence v2.0

KB retrieval (for one element)	Protected	V4Design code repository: https://gitlab.com/v4designEU/retrievebykb	Apache Licence v2.0
KB retrieval (for many assets)	Protected	V4Design code repository: https://gitlab.com/v4designEU/retrieveall	Apache Licence v2.0
Object Localization (STOL)	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4design-stbol/tree/master/objects/Mask_RCNN-master	Apache Licence v2.0
Scene Recognition	Protected	V4Design code repository: https://gitlab.com/v4designEU/sr	Apache Licence v2.0
Building Localisation (STBL)	Protected	V4Design code repository: https://gitlab.com/v4designEU/STBL	Apache Licence v2.0
3D Reconstruction	Protected	V4Design code repository: https://gitlab.com/v4designEU/reconstruction	TBD
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
MongoDB API	Protected	V4Design code repository: https://gitlab.com/v4designEU/v4d_mongodb_api	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

Demo videos for the tools can be found at the following link:

https://drive.google.com/open?id=1I8-gXfoGVWKWwmBRASRbfw3KtyAX12as