V4Design

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

H2020-779962

D6.2 Technical requirements and architecture

Dissemination level:	Public
Contractual date of delivery:	Month 10, 31 October 2018
Actual date of delivery:	Month 11, 2 November 2018
Workpackage:	WP6: System integration and tool development for content re-purposing
Task:	T6.1: Technical requirements and system architecture
Type:	Report
Approval Status:	Final version
Version:	1.0
Number of pages:	112
Filename:	D6.2_V4Design_TechnicalRequirementsAndArchitecture_20181102_v1.0.docx

Abstract

This deliverable details the technical requirements that will be taken into account during the implementation of the V4Design platform. In addition, it describes the functionalities that will be supported by the platform. Finally, it outlines the system architecture that will be used for the implementation of the platform prototypes.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	16/07/2018	ToC creation and content definition	Ayman Moghnieh, Luis Fraguada, Verena Vogler
0.5	10/09/2018	1st integrated draft circulated to WP6-related partners for comments and contribution	Ayman Moghnieh, Luis Fraguada, Verena Vogler
0.6	10/10/2018	Integration of partners' contributions and revision of all sections. Preparation of the pre-final draft and sent for internal review	Ayman Moghnieh, Luis Fraguada, Verena Vogler
0.9	18/10/2018	Internal review comments	Yash Shekhawat (NURO)
1.0	30/10/2018	Preparation of the final draft	Ayman Moghnieh, Luis Fraguada, Verena Vogler

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	Spyros Symeonidis	spyridons@iti.gr
CERTH	Elisavet Batziou	batziou.el@iti.gr
CERTH	Konstantinos Avgerinakis	koafgeri@iti.gr
CERTH	George Meditskos	gmeditsk@iti.gr
CERTH	Stefanos Vrochidis	stefanos@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

Executive Summary

This document discerns the technical requirements and specifications of the V4Design platform architecture and components, including services, middleware modules, and envisioned user tools.

It introduces and explains the tools and mechanisms used to extract and analyse these technical requirements, including documents analysed, use-cases studied, interviews carried on, and surveys conducted. It also discusses the domain of V4Design by reviewing its defined usage scenarios, and discerning the related user requirements under each scenario, and associated with each user profile.

Then, it discusses the main technical concerns of the V4Design platform architecture, being its architecture design and integration model, the logical design and elements of a V4Design service, and the data management policy established in accordance with the platform's expected data processing flow.

On an elemental level, each envisioned platform component is reintroduced before describing its technical requirements with focus on its deployment environment specifications, its data processing and local data management, its relation with other modules in the architecture, and its relevance to the user requirements, through which specific technical requirements are defined. This detailed description of technical requirements and specifications is applied to the eight V4Design services, its two middleware components, and its two envisioned user tools.

Finally, the gathered and described requirements are analysed alongside common concerns, and consequently additional specifications are described for data management including a generic definition of the schemas of the data objects contemplated, a comprehensive list of message topics to establish communication among platform components, and the logical design of the platform cycle and its chronological order.

The document concludes by highlighting the findings described, and the processes by which all partners have converged onto a common understanding of the specifications, functionalities and architecture of the intended platform.

Abbreviations and Acronyms

AMQP	Advanced Message Queuing Protocol
API	Application program interface
DB	Database
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
UIMA	Unstructured Information Management Architecture
URI	Unique Resource Identifier
URs	User Requirements

Table of Contents

1 INTRODUCTION

The V4Design project aims to create an integrated research and innovation platform that will bring together State-of-the-Art (SotA) ICT technologies with architecture and video game designers by incorporating these technologies in renown architecture and video game authoring tools. The envisioned system will cover areas of design process by reusing visual and textual content from the web and by compiling data from renown content providers, which have been included in V4Design project as partners, and repurposing it by extracting from this material enhanced 3D models, in a semi-automatic or complete automatic way. A backend system will overview the availability of this content and will be connected with architecture and video game design authoring tools so as to make it available to them.

The system main stakeholders are video game and architecture designers that want to create a novel exterior or interior spaces or other kind of artifacts that will be included as an asset in their creations.

The main goal of this document is to explain in detail the architecture of the proposed platform including main Technical Requirements (TRs) driving it and the way the proposed design will enable achieving the ambitious goals set out at the proposal. This document was devised and written with the User Requirements (URs) document (*“**D7.2: Use cases, requirements and evaluation plan**”*) in mind.

For that purposes, this deliverable starts in the first section by introducing the practice of technical requirements collection and analysis, arguing about its scope and applicability in the context of V4Design, before discussing the related methods in gathering requirements and those selected and applied in the context of the project.

In the second section, we define the domain of V4Design by discussing its usage scenarios and their related high-level user requirements. This provides an overview of the role performed by the platform as a service oriented architecture geared to perform user-driven tasks.

In the third section, we define and discuss the technical requirements and specifications from the viewpoint of the system architecture, which includes the architecture design, the logical design of a V4Design service including its generic inner-components, and the data management approach followed in the platform.

In the fourth section, we discuss the technical specifications of each of the platform components, including its services, middleware components, and the envisioned user tools. In each case, we discuss the hardware, software, and resource requirements, the functionalities, the technical components, the technical requirements as associated, the data requirements and local storage, messaging and queuing, and non-functional aspects such as reliability, scalability, security and other concerns.

In the fifth section, we analyse the collected and discussed technical specifications, and draw conclusions on their implications towards the architecture design of the V4Design platform.

Finally, we conclude by summarizing the findings discussed in this deliverable.

2 INTRODUCTION TO TECHNICAL REQUIREMENTS

In this document we introduce the technical requirements (TRs) associated with the development of the V4Design platform in order to facilitate such endeavour and help to evaluate how the resulting platform meets the expectations of both developers involved in its creation, and users to which the platform pretends to provide a valuable service.

Before we discuss these technical requirements we briefly introduce general concepts related to requirements gathering and analysis in order to provide a relevant context for the activities by which such process was executed in this project. In this section we first introduce a broad definition of requirements analysis discussing its technical relevance in software development and architecture design referencing common notions and approaches deemed relevant to the context of V4Design. Afterwards, we discuss the scope and applicability of gathering technical requirements and the same context, reflecting on the global objectives behind requirements analysis, concerns related to the scope of such activity, and concerns related to the applicability of the results in facilitating and helping the development of the platform and supporting its evaluation. In addition, we discussed the popular methods and approaches that are usually followed in gathering technical requirements, focusing on those that are relevant to V4Design. Then we discuss the methods that have been chosen for Gathering the requirements of the V4Design platform and consequently form the bulk of the activities by which the subsequent results have been obtained.

2.1 Technical requirements analysis: a broad definition

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product or service. These expectations are linked to technical features that are quantifiable and qualifiable in nature, providing relevant and detailed guidelines for the implementation. In software engineering, such requirements are often referred to as functional specifications that together describe how the system functions from a client viewpoint. They define the system methods, constraints, and objectives according to the client expectations and vision, and usually form the basis for early acceptance testing in the software development cycle. Therefore, technical requirements are usually defined in a realistic and verifiable manner, using description frameworks. For a more formal definition of requirements, and according to [1], a requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document. In this context, requirements analysis is defined as the process responsible for identifying, gathering, and studying such requirements to arrive at a definition of the problem domain and system technical specifications.

In software development, a distinction is usually made between specifications that refer to the **user requirements**, and specifications that refer to the **technical requirements**. The first is usually written in a non-technical discourse and not always directly related to specific technical concerns, whereas the second usually describes in details each and every functionality of every module as well as components of the modules, describing the entire system flow, and how data is going to flow into and from the system. In order to achieve a successful software development cycle, both types of specifications (client and system) need to be addressed. In most cases, user requirements are determined by engaging with the designated user base of the system and drafted prior to the technical requirements. These, can be subsequently developed on the basis of the user requirements, and complemented to address more generic technical requirements. These generic technical concerns tackle system attributes such as security, reliability, performance, maintainability, scalability, and usability, and as constraints on the design of the system. Such requirements are commonly defined as non-functional requirements.

In fact, even though the conceptualization and conceptual design of software systems usually precede the formal definition and analysis of requirements, these are important prerequisites for the design of the system architecture, especially in complex or composite systems. According to Liao et al [2], requirements analysis is the cornerstone of the software architecture design (Figure 1).

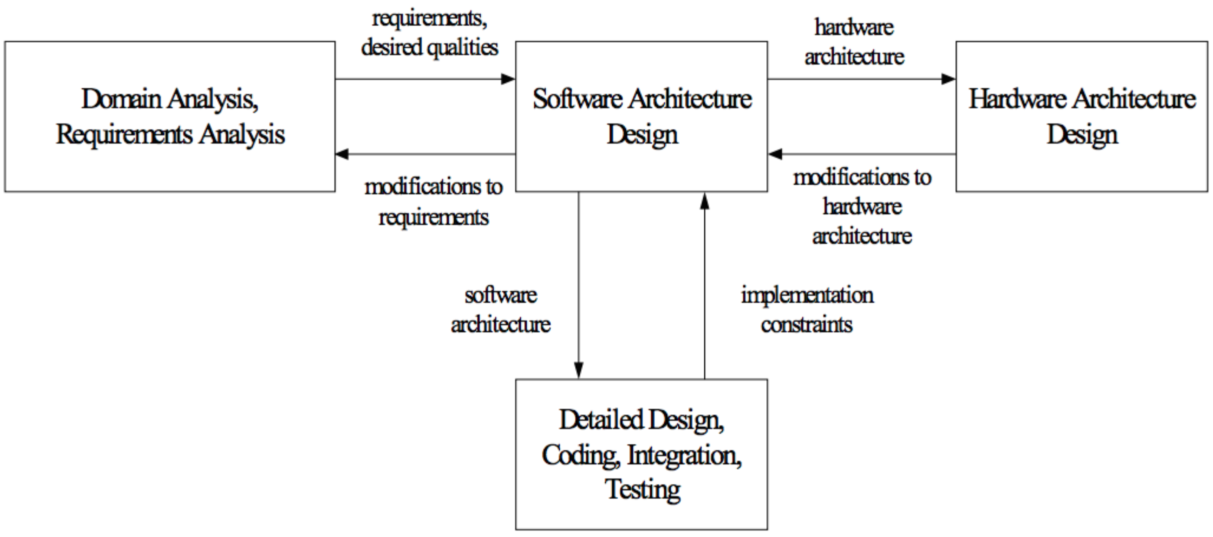

requirements, desired qualities
hardware architecture
Domain Analysis, Requirements Analysis
Software Architecture Design
Hardware Architecture Design
modifications to requirements
modifications to hardware architecture
software architecture
implementation constraints
Detailed Design, Coding, Integration, Testing

Figure 1: Relevance of requirements to the design of software architectures [2]

According to Liao, the first task of software development is almost always the domain and requirement analysis. The addressed requirements are classified as functional requirements (FRs) that describe specific functions that the system is expected to perform, or non-functional requirements (NFRs that describe general specifications that the system needs to meet (e.g. reliability, maintainability, cost, etc.) These requirements are the key input to the second task, software architecture design. As the architect reviews the requirements and proceeds with the design, some modifications to the requirements may be needed in order to insure consistency and coherence among the many elemental descriptors of the system. The software architecture then guides the implementation of the software, including the detailed design, coding, component integration, testing, etc.

Therefore, requirements analysis has arguably the most impacting role on the success of system development, and its stakes in this success can increase dramatically when the system in question is large, distributed, composite, or complex. In fact, the foremost objective of requirements analysis is to discover and describe the boundaries of the new system and how it interacts and integrates in its environment. This wholesome description helps to planify and streamline the development process, and provides a framework for detecting and resolving potential conflicts between different user expectations, and prioritizing the functionalities that the system must support. In short, requirements analysis is the bedrock of a healthy and successful project management for software development, more so when the project in question is large, collaborative, and iteratively developed.

According to Lehman et al. [3], software systems are constantly evolving and adapting to new requirements in order to survive, and such coevolution between systems and user expectations is governed and documented through the elicitation, categorization, and analysis of requirements. These allow to understand and account for maturing trends, tackle bottlenecks coherently without increasing the system’s complexity and plan efficiently the selection and integration of the next batch of functionalities. Software maintenance and evolution of systems was first addressed by Meir M. Lehman in 1969 [4]. Over a period of twenty years, his research led to the formulation of eight “Laws of Evolution” [5]. Key findings of his research include that maintenance is comparable to evolutionary developments and that maintenance decisions are aided by understanding what happens to systems (and software) over time. In this context, developing, updating, and upgrading requirements is a critical activity.

Therefore, the effective lifetime or relevance of technical requirements as defined during the system conceptualization is arguably limited, and should be revisited iteratively (e.g. through interaction between users, stakeholders and developers). In the case of new innovative systems, requirements are expected to be refined after initial user testing, once the system development has reached operational levels (M12), and again after each comprehensive system evaluation (M18-M20, M26-M28, M34-M36).

There are different frameworks for classifying requirements for analytical purposes, some are specialized in specific areas of application such as manufacturing and engineering, but others are more genetic and address the broader field of system design and implementation. one such framework is FURPS, which stands for “Functionality, Usability, Reliability, Performance, and Supportability”, was developed by Grady and Caswell in the last 1980s [6], and still stands as one of the most used for defining both functional and non-functional software attributes. According to this framework, there are five main functionality concerns described as:

- ***Functionality***: general functional requirements defined as the feature set.
- ***Usability***: human factors related to user experience, responsiveness, aesthetic concerns, etc. usually affecting the interface design and the interaction and response models of the system.
- ***Reliability***: availability of system services and functionalities, failure tolerance and recovery, accuracy, stability and other similar concerns.
- ***Performance***: resource consumption both in total and per functionality, data throughput and processing capacity, and scalability.
- ***Supportability***: agility in repair, maintenance policy, testability, modularity, configurability and other similar concerns.

The relevance of these concerns can vary from an architecture to another, and also according to the development stage in which the architecture is found. At the early stage of development, and especially for innovative solutions that cannot rely on similar existing solutions as reference, the main concern of requirements is the functionality. As the development cycle progresses, other concerns gain importance, for instance usability, when the user experience design and implementation can be addressed.

2.2 Scope and applicability of technical requirements analysis in V4Design

V4Design aims to develop a service-oriented platform that enables users to find and reuse digital assets in games, virtual environments, 3D designs, and others. The platform is composed of a series of specialized services, each performing a specific function. The platform concept and its services have been defined in ***"D6.1 Roadmap towards the implementation of the V4Design platform"*** (see D6.2, section 2 and 3). Accordingly, the V4Design platform is conceived as a distributed system that loosely integrates different services and technical components. These components communicate using a message bus solution that allows them to exchange information about their activities, the data that they process and the results that they generate as these become available. In addition, the V4Design platform includes a centralized data repository data storage system that safeguards the raw data collected by the platform and the data generated by its services. The platform integrates several user tools that together represent the platform's user interface. These tools communicate with the platform backend (services and data storage) via an API designed to abscond the backend mechanisms from the user tools in order to streamline user interaction.

The design of such distributed system is a complicated endeavour because it entails many decisions that could impact the architecture scalability and performance on the long run. Therefore, understanding the Technical Requirements (TRs) of the intended architecture design is critical before engaging the design and development of its integrated components. In order to develop and evaluate the V4Design platform, it is essential that its technical requirements and specifications be elaborated in a manner that serves both as a reference and a benchmark. For this purpose, and early-on in the platform conceptualization, design, and implementation cycle, these requirements are defined and analysed systematically. For this purpose, it is important to define the scope of this exercise and its applicability.

In order to elicit the V4Design platform requirements in a useful manner, we first define the following global objectives of requirements analysis:

- ***Create a reference document that details the technical requirements of the platform***, functional as well as non-functional requirements, to serve as guidelines for its development, and to ensure that all stakeholders agree on what the platform is supposed to accomplish.
- ***Describe the tasks that the platform will be able to perform*** concisely and clearly for its users, and the operational concerns that relate to them.
- ***Define the specific functionalities and technical specifications of each service*** individually in order to facilitate its development, integration, and evaluation during the course of the project.
- Gather the technical specifications, constraints, and ***implications of the platform architecture design*** in a manner that facilitates the integration of individual modules and services.
- ***Define the performance metrics*** that should govern the platform and its components in a manner that facilitates the subsequent evaluation of the platform.

In order to gather the requirements in the manner that accomplishes these objectives, we must follow an approach that takes advantage of the related information sources available in the context of the project, and the descriptive documentation that elicit how the platform is expected to function, to be used, and to provide service. In addition, we must involve the stakeholders (users as well as product owners) in the process of gathering these requirements, both for elemental aspects (such as services) and for global aspects (e.g. architecture design specifications).

Therefore we define the following list of concerns that should be addressed at this stage during the analysis of the requirements:

- Elicit **technical requirements** that stem from the project's defined **use cases** and use scenarios, which form the bulk of the user requirements. although use cases and use scenarios (D7.2) may evolve during the project, we can now derive initial set of user requirements that is expected to encompass fundamentally important functional requirements that relate to the technical objectives of the project and the overall role that the platform is expected to play within its users ecosystem.
- Elicit technical requirements that stem from the **platform's architecture** and **architecture design**, which act as constraints and technical specifications for the services envisioned that will be integrated in the platform.in addition these requirements will describe the manner by which the platform will be deployed, serviced, and scaled.
- Elicit technical requirements that stem from the nature of **each envisioned service** and technical component of the platform in an elementary manner and separately for each component, including backend technical components, middleware components, user tools, and others.

Once these requirements have been gathered, we should argue about the applicability and the context of developing, testing and evaluating, and deploying the platform. This should explain how the requirements would be employed in the context of the project, and the relevance do the different activities, objectives, and milestones.

Therefore we define the following list of concerns related to the applicability of requirements that should be addressed on the basis of the analysis of gathered requirements:

- Describe or classify technical requirements according to their relevance for the platform as a whole, and for the components as elemental modules.
- Differentiate between requirements that are expected to influence the development of the platform components, acting as guidelines or reference that constrains such activities, and requirements that are expected to act as a benchmark for the evaluation of the platform and its components.
- Evaluate the relevance of the gathered requirements with relation to the planned iterative versions of the platform, separating between long-term requirements or requirements that are not expected to change, and other requirements that are more relevant to the initial or intermediate versions planned according to the development plan.

2.3 Methods and approaches for gathering specifications

Before we discuss a systematic approach to gathering technical requirements, it is worth noting why such an approach is needed. In fact, Requirements gathering is a critical, foundational step in all system development. It will either set the project on a course to success if done well or doom it to failure if done poorly. As previously argued, each case of system development can differ from others in many aspects that impact how the requirements should be defined in order to provide solid guidelines for development and evaluation. There are many difficulties related to gathering requirements for instance:

- Prospective users are not always able to identify their specific requirements in a clear manner to the development team. In case of new systems or innovative systems, it is hard to engage potential users early-on in the process of conceptualization, and acquire clear requirements since these potential users may not have a clear idea about their own expectations.
- Usually, “**big picture” requirements** are gathered first because they are the easiest to define and describe since they are implied by the system concept and general purpose. But requirements that describe more in-depth and/or localized concerns are generally much more difficult to detect, gather and describe effectively.
- There is a fundamental difference between user stories and functionalities that the system could support. While usually the functionalities are extracted from user stories and define according to the details of the stories, it is difficult to assume that specific user stories represent the overall expected functionalities because, while user stories are usually drafted to describe how is system as envisioned to work and function, it is seldom possible to make sure that these stories represent the most genetic aspect of the system, or the most genetic functionalities that the system must support. This is why design and implementation is undertaken to ensure that the developed system adheres intimately to the expectations of its users, evaluating its role and performance at each stage or at each iteration, at making necessary adjustments to its design, functionality, and performance.
- Finally, in the relevant case where the system development is assumed by a distributed team or composite team, or if the system is envisioned to integrate components each developed by a different entity, defining and gathering requirements in a coherent manner is also difficult because it entails the need to remedy different and sometimes contradictory approaches to the implementation of processes, some of which are shared among the different components.
- Finally, traducing requirements of different types into clear technical requirements that can be measured and evaluated is not a straightforward task, and relies on the ability of the system architects to adhere to a common and detailed vision of the architecture.

Many techniques are available for gathering technical requirements. Each has value in certain circumstances, and in many cases, there is an inherent need to rely on multiple techniques to gain a complete picture from a diverse set of clients and stakeholders. The most popular techniques for gathering requirements are the following:

Document analysis

This technique is usually the precursor for gathering requirements, or the first activity performed in this context. The analysis of existing documentations that describe the system concept, objectives, expected outcome, development process and plan, the needs to which the system is ought to respond, the users and their profiles, and any other preliminary documentation. The information extracted from these documents cannot provide a comprehensive and complete set of technical requirements; nevertheless it can define the general traits of the system and its most pervasive characteristics. Such specifications would delimit other requirements gathering activities and help them to focus on concerns relevant to the development and evaluation of the system.

One-on-one interviews

This technique is the most common for gathering requirements, by which clients or (future) users of the system are interviewed individually and asked about their expectations and needs, and the manner by which they envision the system. These interviews are usually planned ahead, and can be entirely or partially structured. In general, open-ended questions work best to extract the interviewee’s viewpoint and vision, which closed questions aim to obtain specific answers to specific concerns. In case of innovative systems or new systems with no comparative references, probing questions can uncover hidden requirements and can shed light on unforeseen aspects that can impact the original concept of the system.

Group interviews and focus sessions

Group interviews are similar to the one-on-one interview in many aspects, but they usually involve several interviewees. They are two known (but not always disjoint) types of group interviews, one that involve people with similar role and background, and one that involve representatives of the different profiles that will be involved with the system (e.g. users, administrators, mediators, learners, teachers, experts, advanced, etc.). Group interviews usually require more preparation and more formality to obtain the information required from all the participants. Keeping the group focused on the tasks is generally the main challenge of this approach, but it can result in a rich set of requirements in a generally short period of time.

Joint application development sessions

This activity targets the developers and the technical profiles involved in the system design, architecture, and development. It is a popular approach to extract non-functional requirements, and to establish a comprehensive framework for the development, implementation, and deployment of the system. These sessions share a lot of similarities with group interviews, and they usually are prolonged until all the objectives are met and a detailed and comprehensive set of requirements is obtained.

Questionnaires and specification templates

Templates are arguably the most effective approach to gathering technical requirements, and it is a common practice to circulate structured questionnaires prior to focused interviews with system architects and product owners, since they help to prepare these interviews by focusing them on the remaining ambiguities. Questionnaires are effective in involving a large number of stakeholders in the process of gathering requirements, and their structure helps to identify “popular” or generic requirements easily, and isolate requirements that are specific to certain situations but not others. Also, this activity can harmonize the visions of different architects and system developers involved in conceptualizing and implementing the solution.

Prototyping

Prototyping is a relatively modern technique for gathering technical requirements, which stem from agile approaches to system implementation. It is a "try and learn" approach to development that pretends to increase the speed of development. Based on a preliminary set of requirements that is extracted from the system concept and overall definition, a quick-and-dirty prototype is implemented for evaluation. It helps to concretize the approach defined by the system and illustrate the objectives more concretely to the involved stakeholders. In essence, this approach works well with smaller systems that are more user-oriented and interaction-driven, or cantered on user-experience. However, it may not work well with larger, more complex systems as these are harder to simplify and prototype effectively without a considerable investment.

Use cases and scenarios

Use cases are basically stories that describe how discrete processes work. These stories include specific profiles called "actors" describe how the system works from their perspective. Use cases may be easier for the users to articulate, and they can be later distilled into the more specific detailed requirements. On their own, use cases may not describe the requirements in a complete manner, they tend to focus on clear processes that are directly implied by the system concept. Use cases are also defined by the stakeholders in charge of conceptualizing and developing the system as a manner to illustrate the concept to potential users. Overall, use cases play a diversified role in the development process, and drafting use cases is a cheap and straightforward way to provide a basic system description. Therefore, they are often used as a precursor to gathering requirements.

There are other, less popular methods for extracting technical requirements that include brainstorming sessions organized with stakeholders, request for proposals that resemble questionnaires but are usually structured and drafted by the clients instead of the system analysts, among others. In some cases, experiments are conducted to extract data that in turn is employed to extract specifications with technical implications. However, these approaches are deemed less relevant for V4Design.

2.3.1 Selected approaches for gathering requirements in V4Design

In the context of V4Design, and in order to extract the technical specifications of the envisioned platform and technical requirements for the system components from collaborating partners, we follow a composite technique that groups the methods described in Table 1 below.

Technique	Description	Expected outcomes
Document Analysis	We analyse the project proposal and plan of work, and the information regarding technical specifications made available by the partners on the wiki site.	General description of the architecture requirements, and technical objectives to be met by the system.
Use scenarios and use cases	We study the defined use cases and scenarios, declared in D7.2) that explicitly state the user processes that the platform should support to accomplish specific objectives.	This allows us to get the user requirements and prioritise them according to the use cases, as these will be implemented in series. The user requirements are then mapped onto the technical requirements.
Component definition survey	We conduct a template-based survey with the developers/owners of V4Design components that are to be integrated in the platform (see appendices A, B and C)	The characteristics and specifications of each service have been defined by their owners, including a correlation between the envisioned functionalities and the technical requirements..
Interviews	We interview developers/owners of V4Design components to establish a systematic understanding of critical technical aspects, and debate different alternatives for addressing them.	Detailed definition of critical technical specifications that impact the integration of the architecture and the performance of its modules.

Table 1: selected techniques for gathering requirements

From the document analysis, an overview of the technical specifications of the platform is extracted, described, and documented in a traceable manner. This includes a definition of the architecture model adopted for the platform, which defines the platform components including the services that integrates, the middleware solutions that that utilizes, the manner by which it is deployed in a production environment, and the approach it follows to interact and provide service for its users.

Based on the drafted user scenarios and use cases, the main functional requirements are extracted and formally defined both on platform level and service level. these requirements encapsulate the user expectations and the main functional aspects of the platform, which ought to be met, evaluated, and valued a different stages and the project.

The component definition survey, which was conducted at the beginning of the project, provides an in-depth definition of each component that is integrated in the architecture of the platform. It reveals the original plans that the teams responsible for developing the have devised in order to address different genetic technical specifications, such as scalability, data management, consistency, expected throughput, among others. These specifications represent the non-functional requirements of the platform defined on a component level. In complementation, the requirements definition survey, which was conducted during the development of this deliverable, cantered on detailing the requirements of each platform component, complimenting previously obtained information and knowledge about the inner-workings of these components with additional information that illustrate how these components are expected to meet their objectives, how they would be hosted or deployed, how they would treat data locally and globally within the ecosystem of the platform, how they would communicate with other related platform components, and to what purpose would they establish such communication. In essence, the survey aimed at completing the initial exercise for gathering requirements which is represented by the previously mentioned activities.

Finally, and based on specific necessities to obtain information or to elaborate on specific concerns related to the definition of component-level requirements, several open-ended and mostly unstructured interviews have been conducted with the owners of specific services and platform middleware. These interviews explain how to complete the requirements gathering exercise at this stage in the project.

Although this approach for gathering the technical requirements (TRs) of the V4Design platform has generated a comprehensive set of requirements, both functional and non-functional, both top level (generic) and low level (specific), and both platform-related and components-related, there is a need to revisit these requirements at several stages in the project. This should be done especially after the completion of evaluation activities that aim to ascertain how different versions of the platform under development meet the overall expectations and the specific requirements originally conceived at the beginning of the project, and representing the motivation and justification behind such endeavour.

During the development of the platform and its components, the requirements defined in this exercise should serve as guidelines and evaluation criteria under which the development activities must be governed. This is made easy by separating **platform level requirements** (mainly architecture design specifications) from **component level requirements** (related to elementary functions implemented by each component), and therefore each team of developers should be able to isolate easily the requirements pertaining to the component it is developing.

3 DOMAIN DEFINITION: OVERVIEW OF V4DESIGN USAGE SCENARIOS AND USE CASES

Four different use cases have been defined to describe the user expectations related to the platform. These use cases include a total of six different scenarios two of which being a slight variation of a single-use case. In the following, we describe these use cases and scenarios concisely. We then analyse their direct implications for the platform in terms of requirements, especially what pertains to the user expectations and the role that the platform should perform for them in each case. More detailed descriptions of these use cases in the deliverable ***"D7.2 Initial use case scenarios and user requirements"***.

3.1 V4Design use cases and scenarios

In the following, we offer a summarized description of the V4Design use cases and scenarios, describing in each case the overall theme and interaction scenario, and the related high-level user requirements as defined in D7.1. We then use this information as a reference to contextualize other technical requirements in the following section.

Use Case 1: Architectural design, related to existing or historical buildings and their environments.

Users: Architects, designers and artists

Scenario 1.1: Support the design process of pavilions, land art, scenography (Landscape). 3D models of buildings, debris, the surrounding landscape (e.g. from ancient times up until the 19th century) will be extracted to support the design process of large objects (pavilions, land art, interior architecture etc.).

Scenario 1.2: Architectural design, related to existing or historical buildings and their environments (Building). 3D models, images and maps of the immediate vicinity and reference models of similar size and style, will be used to study various design options.

HLUR	HLUR Title	HLUR Description
HLUR_1.1	Extraction of 3D models	Architects and designers can extract 3D models of places, buildings and objects out of videos and images of buildings, landscapes, artworks or sensitive space elements.
HLUR_1.2	Extraction of CG assets	Architects and designers can extract 3D textures, computer graphics (CG) materials from 2D images of buildings, landscapes, artworks or sensitive space elements.
HLUR_1.3	Architectural design tool to form innovative ideas	Architects and designers have a tool that can assist in formulating new, innovative architectural ideas
HLUR_1.4	Multiplicity of assets	Assets can be 3D objects, 2D videos/images, textual information, audio etc.
HLUR_1.5	User interaction and control	Architects and designers will be able to access the 3D assets (3D models, point clouds, Meshes) in a 3D environment and they will be able to edit and manipulate them.

HLUR_1.6	Extraction of 2D assets	Architects and designers can extract 2D patterns of artworks and culturally sensitive space elements in editable vector format
HLUR_1.7	Asset accessibility and searching refinement	Architects and designers can have access to a variety of extracted assets and have the ability to filter and refine their search results.
HLUR_1.8	Related and suggested assets	Architects and designers can have access to a variety of other related or suggested assets to the asset they are working on.

Table 2: High-level user requirements for use case 1 (D7.2 - section 3.1.5)

Use Case 2: Architectural design, related to artworks, historic or stylistic elements

Users: Architects, interior architects and product designers

Scenario 2.1: Architectural design, related to artworks, historic or stylistic elements (Object, Interiors). 3D models inspired by artworks of a specific style, historic spatial elements and arrangements will be easily accessed for the design, modelling and actual fabrication of novel collections of small scale industrial objects (e.g. furniture), with reference to these styles.

HLUR	HLUR Title	HLUR Description
HLUR_2.1	Extraction of 3D models	Architects and designers can extract 3D models of places, buildings and objects out of videos and images of buildings, landscapes, artworks or sensitive space elements.
HLUR_2.2	Extraction of CG assets	Architects and designers can extract 3D textures, cg materials from 2D images of buildings, landscapes, artworks or sensitive space elements.
HLUR_2.3	Architectural design tool to form innovative ideas	Architects and designers have a tool that can assist in formulating new, innovative architectural ideas
HLUR_2.4	Multiplicity of assets	Assets can be 3D objects, 2D videos/images, textual information, audio etc.
HLUR_2.5	User interaction and control	Architects and designers will be able to access the 3D assets (3D models, point clouds, Meshes) in an 3D environment and they will be able to edit and manipulate them.
HLUR_2.6	Extraction of 2D assets	Architects and designers can extract 2D patterns of artworks and culturally sensitive space elements in editable vector format
HLUR_2.7	Asset accessibility and searching refinement	Architects and designers can have access to a variety of extracted assets and have the ability to filter and refine their search results.
HLUR_2.8	Related and suggested assets	Architects/Designers and game developers can have access to a variety of other related or suggested assets to the asset they are working on.

Table 3: High-level user requirements for use case 2 (D7.2 section 3.2.5)

Use Case 3: Design of virtual environments, related to TV series and VR video games

Users: Visual content producers (film, TV industries)

Scenario 3.1: Creation of a VR video game based on the scenes of a telenovela. 3D models of interior elements and scenes will be extracted from existing video contents to build interactive media and VR games with the same assets, scenes and characters.

HLUR	HLUR Title	HLUR Description
HLUR_3.1	Multiplicity of assets	Assets can be 3D objects, 2D videos/images, textual information, audio etc.
HLUR_3.2	Related and suggested assets	Game developers can have access to a variety of other related or suggested assets to the asset they are working on.
HLUR_3.3	Immersive educational environment	The student finds her-/himself in an environment, where she or he can study by watching a 2D video (= episode of Nico's Weg) and do the corresponding exercises with a clear and comprehensive UI and gameplay that meets the requirements in regard to language learning didactic.
HLUR_3.4	State of the art gameplay	Game developers envision state of the art gameplay (with gaming elements) that takes place in three dimensional environments inspired by 2D content (Nico's Weg)
HLUR_3.5	Avatar Extraction	The components should also be able to extract the following: (i) Avatar, structure, etc. from DW telenovela; (ii) Avatar, structure, etc. from non-DW footage
HLUR_3.6	Tool for Language related Game Design	Game developers would like to have a tool that can assist in the design of new, immersive VR environments for language learning purposes

Table 4: High-level user requirements for use case 3 (D7.2 section 3.3.5)

Use Case 4: Design of virtual environments, related to actual news for VR (re-) living the date

Users: Worldwide users that want to live or relive news events in a VR environment

Scenario 4.1: Creation of a VR application based on historic events. Selected parts of past and more recent news coverage which will be transformed to 3D and VR environment that will allow users to have a more realistic information experience.

HLUR	HLUR Title	HLUR Description
HLUR_4.1	Multiplicity of assets	Assets can be 3D objects, 2D videos/images, textual information, audio etc.
HLUR_4.2	User interaction and control	Game developers will be able to access the 3D assets (3D models, point clouds, Meshes) in an 3D environment and they will be able to edit and manipulate them.
HLUR_4.3	Extraction of 2D assets	Game developers can extract 2D patterns of artworks and culturally sensitive space elements in editable vector format

HLUR_4.4	Related and suggested assets	Game developers can have access to a variety of other related or suggested assets to the asset they are working on.
HLUR_4.5	Multiple environments	Ability to develop multiple environments for the same scene and change them using scrollbar.
HLUR_4.6	Data about the initial asset	Get data about the video that an asset is extracted from
HLUR_4.7	Tool for History related Game Design	Game developers would like to have a tool that can assist in the design of new, immersive VR environments for reliving a significant past event

Table 5: High-level user requirements for use case 4 (D7.2 section 3.4.5)

Although the definition of these use cases is broad and general, several high-level user requirements can be extracted for each case, providing reference for the specific needs that should be covered under these cases. These high-level requirements correlate with more elementary requirements that describe specific functions and functionalities that ought to be achieved by the system due to the broad nature of high-level user requirements, these can share several elementary requirements, or in other words each elementary user requirements can be associated with one or more high level requirements.

The complete set of identified and described user requirements is presented in D7.1 and D7.2. This set includes four types of user requirements classified according to the MoSCoW prioritization technique [6], which divides requirements into four priority sets, being: **must-have, should-have**, **could-have**, and **would-have** (see D7.2). The first two sets “must-have” and “should-have” represent the essential requirements that together describe the Minimal Viable Product or the most reduced version of the platform that meets the objectives behind its conceptualization and development. We therefore qualify these essential requirements as critical and important, and prioritize the assessment and evaluation of the developed components and services accordingly.

3.2 V4Design user profiles and user requirements

In the course of the use cases, several user profiles have been identified as primary targets for the V4Design platform. Accordingly, the platform should address their needs and expectations effectively, and as defined by the User Requirements (UR), described in D7.2. These profiles include:

- Architects (interior, exterior, urban, etc.)
- Artists and Designers (products, games, etc.)
- Visual content producers (cinema, TV, etc.)
- Others (requirements not specifically associated with any profile)

The user requirements associated with each profile differ from those associated with another. These differences are not negligible, as it can be verified by comparing what each technically implies (tables 6, 7, 8 and 9). For instance, the requirements of designers centre of search and extraction of assets, while architects and content providers use the system not only to find interesting assets, but also to manipulate them and use them inside the system’s tools. Content providers also have a crucial need to upload and process data on demand, a function that designers do not consider as highly relevant.

From a technical stance, all the user requirements converge on a coherent set of functionalities that should be implemented and organized. Some of these requirements are tool-cantered (e.g. GUI and interaction related requirements), while others are directly related to the backend of this service architecture (e.g. get data, upload and process new data, etc.).

UR#	Implied technical requirements
UR_56	get sizes of objects detected in a video
UR_57	get textual and semantic data, and text summaries of an asset
UR_58	get asset extended semantic analysis
UR_59	search for audio assets
UR_64	get asset metadata
UR_66	search for available 3D models
UR_67	get lighting, colour grading, skymap setting from image or video
UR_1	upload videos/images
UR_70	access to high-quality 3D models

Table 6: Designer-specific user requirements

UR#	Implied technical requirements
UR_48	process data on demand
UR_51	get 3D model with metadata
UR_52	get asset catalogue
UR_53, UR_54, UR_55, UR_60, UR_61	Features of GUI

Table 7: Content provider-specific user requirements

UR#	Implied technical requirements
UR_3, UR_4, UR_41	get textures from an asset
UR_5 --> UR_9	Front-end extraction of different aspects

UR_30 --> UR_40, UR_42, UR_43, UR_45, UR_46	Features of GUI
UR_44, UR_2	get 3D model with metadata
UR_47	Tool is open-source
UR_1	upload videos/images
UR_70	access to high-quality 3D models

Table 8: Architect-specific user requirements

UR_10, UR_20, UR_23, UR_24	get semantic data / tags
UR_11, UR_12, UR_15, UR_16, UR_19, UR_22	get asset metadata
UR_18	get aesthetics from an asset
UR_14, UR_21, UR_26, UR_50, UR_68	get 3D model with metadata
UR_27, UR_41	get textures from an asset
UR_29	get aesthetics from an asset
UR_62	import asset
UR_25, UR_28, UR_49, UR_63, UR_65, UR_68, UR_69	tool GUI features

Table 9: Other user requirements

This clusterisation of user requirements by user profile will provide a reference for the integration of the platform components and its evaluation from a technical stance. The cohesion of functionalities by user profile may indicate where components should be chained together and where they should be integrated in an ad-hoc manner. This is explored in more details in section 5.

4 ARCHITECTURE DESIGN SPECIFICATIONS AND TECHNICAL REQUIREMENTS

In general, requirements specify what a system must do, while its **architecture design specifications** describe how it will be organized in order to fulfil these requirements. Intuitively, one might consider that the definition of requirements precedes and act as a precursor for the architecture design. However, these two tasks overlap to a large extent.

This relationship between architecture design and system architecture design has been studied by Clements et al., showing how they are intertwined [8]. The essential outcomes expected from the system are basically requirements that define the objectives of the system and the goals it should achieve. They determine and delimit the design of the architecture, for instance they imply how the system should perform, how it should reach its users and react to their needs, and how it should fit within the ecosystem of existing systems and services related to the field or area of application of the system.

On the other hand, while essential outcomes represent an early set of requirements upon which the architecture design is based, the design process itself can reveal challenges or restrictions for the system components. These discovered challenges, which are usually more elaborate in distributed architectures and composite or large-scale systems, are also a type of requirements that individual system components must meet. In general, these challenges stem from possible conflicts between the essential outcomes and therefore need to be addressed, or from constraints related to the envisioned deployment environment of the system, or similar concerns.

Finally, the finalized design of the architecture itself and the choices it entails do impose some overall constraints on the system. These constraints are essential for the proper functioning of the chosen architecture model and the successful implementation of the system. For instance, in order for an architecture design to work, engineers could specify a minimum throughput for system components, or dictate a data transfer policy, and so on. For instance, if the architecture design implies a loose integration model for the system components, then this constraint is considered as a requirement that the development of the system components must meet.

In this context, we introduce the notion of **architecture design specifications**, which describe any specification that is architecturally significant. Such requirements mandate how the system architecture is conceived, designed, and implemented, and are seldom changed after the initial system implementation, unless a radical change in the system’s nature, goals, or performance is required.

4.1 Architecture design of the V4Design Platform

In the following figure 2, we describe the logical design of the V4Design platform, which shows how the different logic elements are organized. The Platform is composed of eight **services** (in black), each specialized in performing a specific function with the data collected and stored in the platform. In addition, the platform incorporates two **user tools** (in grey) that provide service for the user profiles targeted by the platform. The platform **middleware** (in yellow) is composed of two components: a message bus through which all communication with and between the services is channelled; and a REST API that provides interfacing capacity for the user tools with the rest of the platform. Finally, the platform integrates a **data storage** and retrieval system capable of connecting to external sources and acquiring raw data, which is stored alongside the data generated and processed by the services.

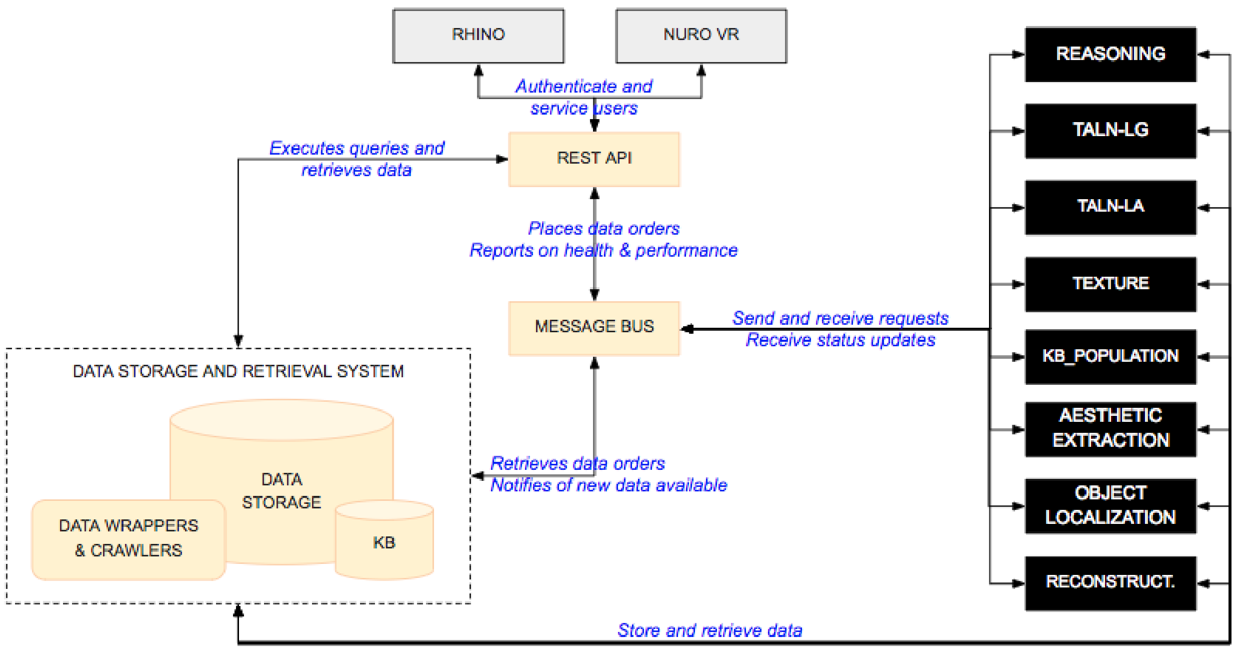


Figure 2: logical design of the platform architecture

Based on this architecture design and the development plan conceived for the platform and documented in D6.1, a logical design for the first version of the platform has been developed. This design is shown in the following figure 3. Accordingly, the first version of the platform will incorporate at most five different servers, housing the following components: Four distinct services, the message bus, the REST API, a representation or abstraction of the user tools (e.g. a demo web page with query and retrieval capacity), and the data storage and retrieval system. Each of these components will be a primitive and limited version of the envisioned concept. In some cases, processes will be simulated inside components in order to support the development of cross-service processes.

Distributed hosting

The selected model of the platform integration dictates that services are independent components, each hosted on its own server that is provided and cared for by the service owner. Some individual services can be combined and hosted on a single server, and in this case this set of individual services will be considered as a single service from a platform perspective, communicating with the rest of the platform components through a single interface, or a single message encoder/decoder solution.

Integration model

The integration model selected for the V4Design platform is a distributed architecture with a single centralized storage, and a message bus that plays a centralized role in integrating the different services and components. Each service and middleware component should establish communication with the message bus in order to send and receive data and requests to other services.

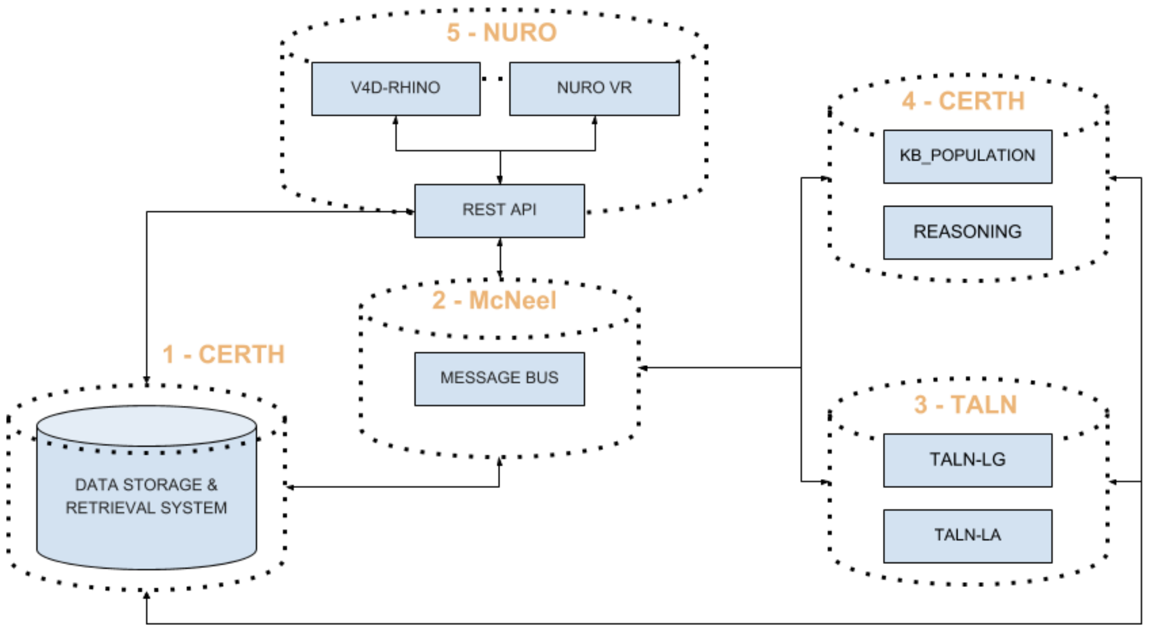


Figure 3: logical design of the first version of the platform

Security

Each service will be responsible for its own security on a local level, especially if it is communicating with external APIs or if it supports user interaction.

The message bus will provide an extra security layer by asking each component to authenticate themselves in order to communicate with the rest of the platform components.

The user tools are responsible to manage their users' authentication and permissions, and administrate their usage cota. The tools can centralize these functionalities under the Rest API middleware, or they can choose to address them locally.

Performance and scalability

The platform should be able to support more than a 1000 user per instance, amounting to an estimated 100 simultaneous connections, without affecting the platform performance. Since the platform manages media objects and data objects of large size, loading and safeguarding data may be delayed or limited by the data transfer capacities of the networks. Policies for alleviating data exchange between the platform's front-end (user tools) and backend (services and data storage) should be adopted.

Each service should be scalable in a manner that accommodates the long-term requirements of the platform. Therefore, services that are originally designed to process a single request at a time should implement a scalability paradigm that overcomes this limitation. This is required for the final version of the integrated platform and could include the introduction of (or at least support for) parallel processing or the deployment of several instances of the service.

Consistency and quality of service

In the deployment environment, the services should be available continuously. Periodical upgrades and maintenance activity can be scheduled during which components and services can be upgraded and maintained. Services and components should be able to recover in the case of failure. Fatal failures from which recovery is impossible should be logged and analysed. Fatal failures that affect the entire platform, for instance failure of the message bus or the REST API, or the data warehouse, should be reported to the service owners.

Maintaining a separate testing and development environment

For each component, a separate testing and development environment should be maintained in order to test and validate new development versions prior to its deployment in the production environment where stability and performance are key. Components hosted on development environments may use the same middleware components of the production environment.

Data management

A single data warehouse service will be built for V4Design. It will host and service all types of data, including semantic and non-semantic data, generated and required by the services, and provided to the users via the user tools.

In order to retrieve data from the data warehouse, all services are expected to issue a "query message" through the message bus, and cannot access data directly from the data warehouse (unless they already know the URIs of the requested data objects). The data warehouse will in turn respond with a "data ready" message that contains the URIs of requested objects.

The data warehouse will allow services to ingest data objects directly through a "push" mechanism. A message is broadcasted by the data warehouse when new data (raw or processed) is ingested/stored.

The services can store some data locally for performance and optimization reasons, but cannot provide locally-stored data to other services. Data queries are only allowed for the data storage module(s). Services can implement data queues when deemed necessary to streamline data locally.

Messaging

The platform message bus will be built on the basis of Apache Active MQ open-source solution. Each service or middleware component should establish communication with the platform's message bus in order to interact with other platform components.

Services and components will be required to authenticate themselves through the message bus for security reasons. Support for this feature is not required for the first version of the platform.

Encoding and decoding messages from the message bus is a task pertinent to each system component that uses the message bus to communicate with other components. The messages are encoded following the AMQP protocol.

Many libraries for encoding in AMQP are available for different development languages and frameworks (e.g. see activemq.apache.org/cross-language-clients.html). The message's content encoding should follow an agreed schema developed collaboratively by the teams responsible for developing the platform services and components. The CAP protocol should serve as a reference for this schema, especially the manner by which it describes the data objects associated to the message.

There are 4 types of messages supported by the message bus:

- **Broadcast**: a service broadcasts a message to inform other services of changes in its status-quo, including the arrival of new data, the completion of a process, and so on.
- **Service-to--Service**: a service places a request to another service, which could include a query for data, a trigger for a function, and so on.
- **I'm alive**: the service informs the message bus that it is up and running
- **I'm shutting down /Unavailable**: the service informs the message bus that it is going offline. The message bus will also periodically check the services reachability automatically.

Logging

Each component or service integrated in the platform can log and track its activity, the data it processes, and the results it generates. No centralized logging system will be implemented. For instance, the message bus will log all exchanged messages and will track the availability of the services, and the REST API will log user activity and the queries they perform.

All logging information and data can be kept for a limited period of time that befits the objectives behind saving it (for instance, tracking performance). This period should not exceed six months. Afterwards, logging information and data should be flushed from local storages.

User data and privacy

The platform tools can collect data on their users in accordance with EU privacy laws. In this case, users are to be explicitly notified and permission solicited on an individual basis before data is collected. In case a user does not approve the request for storing data pertinent to his or her use of the system, the user should be allowed to use the platform without any restrictions.

No user data should be shared with 3rd parties under any circumstances. In case such exchange is deemed necessary for the success of the platform, or for any critical purpose, permission for such an exchange should be granted by the V4Design project coordinator, after explicit consultation with the consortium.

Data backup and restore policy

Backup of critical data of dynamic nature (e.g. reference sets, training benchmarks, raw data collected, etc.) should be performed on a local level, i.e. by each service and component separately. Local backup policies should be drafted in accordance with the needs of each service or component, and the type of data stored locally.

Backup policies and mainly relevant for the platform's centralized data storage, where virtually all data collected, processed, and generated in the platform is stored.

Restoration is performed in the event of data corruption or data loss.

Use of external libraries, tools, and software components

All software components developed by third parties and utilized in the development of the platform services and components should be open-source with a creative-commons license that does not limit, inhibit, or curtain any possible exploitation of the V4Design platform and its services. The use of off-the-shelf legacy solutions should be avoided.

4.2 Logical design of a V4Design service

In order to standardize and facilitate the development of services for the V4Design platform, an abstract model that reflects a service's conceptual design was devised to encapsulate the most important architectural requirements at service-level. This model is shown in the following figure 4.

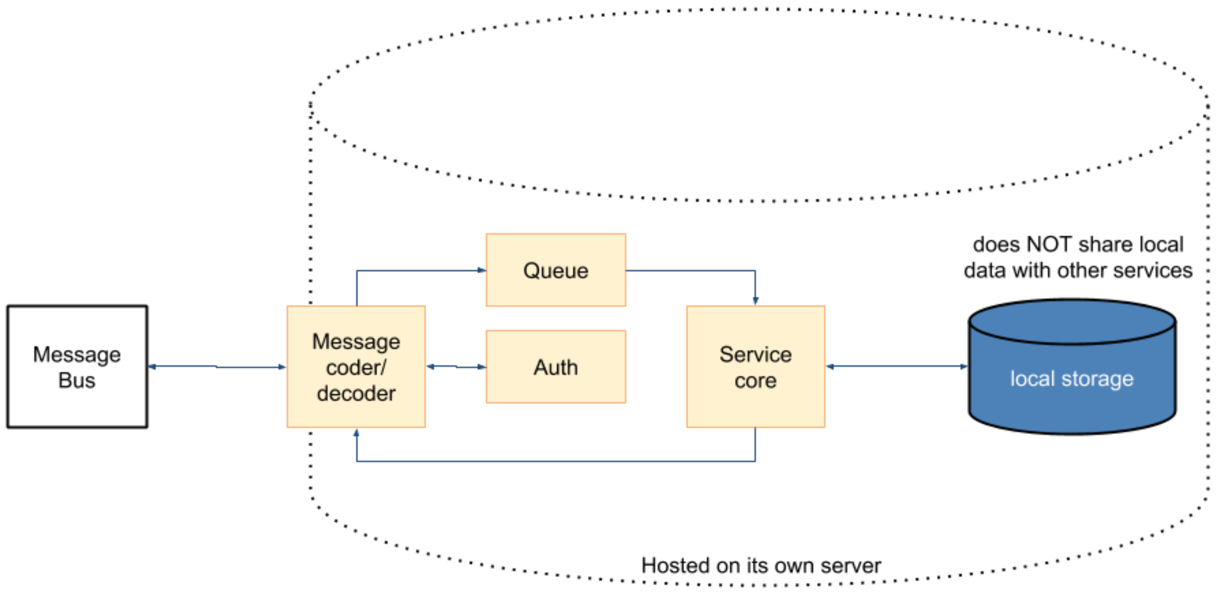


Figure 4: Conceptual design of a V4Design service.

According to this model, a service is a system component that is hosted on its own server and communicates with the rest of the platform components via the message bus. It is composed of five logical units being:

The message coder / decoder: responsible for encoding and decoding messages in the proper format to interface with the message bus.

The Authentication unit: responsible for maintaining and providing the necessary credentials for the service in order to authenticate itself in the platform through the message bus.

The service core: is the unit that encapsulates the core algorithm and/or method(s) provided by the service to the platform. It represents the service's contribution to the processes supported on the level of the platform.

The Queue: is an internal queuing mechanism that stores received requests and prioritises them according to the local policy of the service, acting as a buffer between the service core and the message bus.

The local storage: is a data storage unit used exclusively by the service core for storing or buffering data in order to support the execution of the methods encapsulated in the service core.

4.3 Data management specifications

A “***data object***” is a self-contained piece of data that components treat as a singular unit. Data objects can be related and chained. They contain metadata fields that describe their origin, characteristics, and purpose. The following logical diagram explains how the data objects are described in the context of V4Design services and platform components.

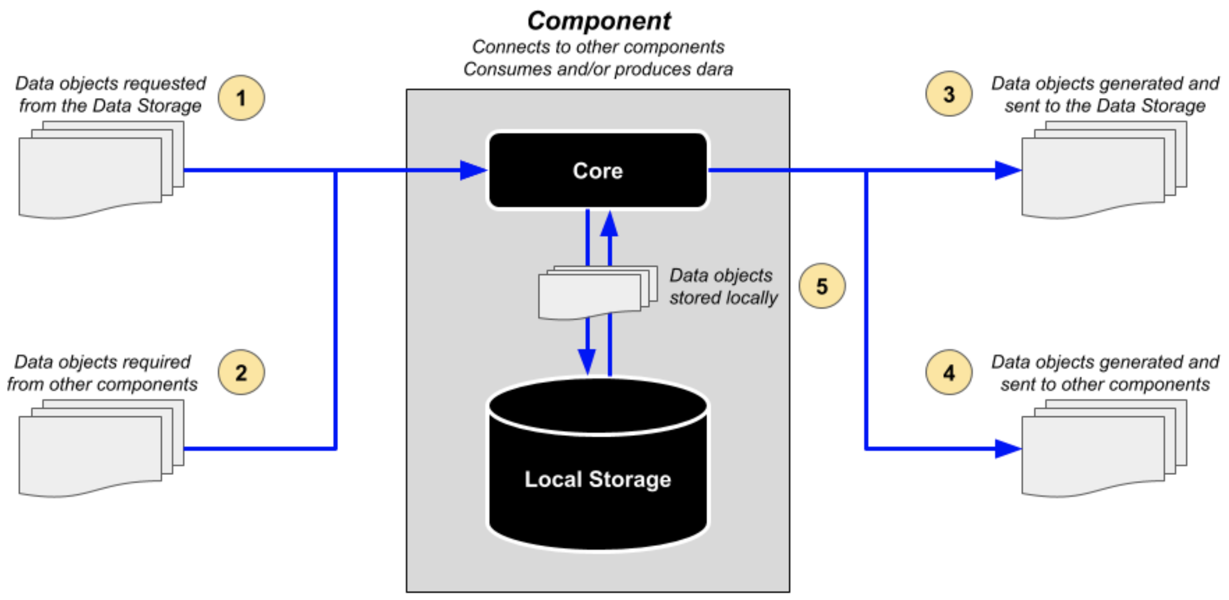

Component
Connects to other components
Consumes and/or produces dara
Data objects requested from the Data Storage
1
Data objects required from other components
2
Core
Data objects stored locally
5
Local Storage
3
Data objects generated and sent to the Data Storage
4
Data objects generated and sent to other components

Figure 5: Data management at component-level

Each component can have five different relations with data:

1- First, the component **reads data stored in the data storage** component to process locally. Usually, this describes the operations by which the component fetches raw data from the Data Storage upon receiving a notification of its availability. This raw data can be relevant to different components of the platform, these are notified through a broadcast message indicating the date the elements’ URIs or parameters for retrieval from the Data Storage.

2- Second, the component **reads data generated by other components** in the architecture, and processes this data locally. This data is not raw in the sense that it has been previously processed by another system component, and is considered result generated by the specific component.

3- Third, the component **generates data** and wishes to store it in the data storage. This data represents the results generated by this component, and is final in a sense that it is ready for user consumption.

4- Fourth, the component **generates intermediary data** in the sense that other services should use it as input to generate consumption-ready data. The component wishes to make this data available for these other consumers.

5- Fifth, for reasons pertaining to the nature of the processes that the component implements, and in order to improve performance in some cases, or make the functioning more agile. The components may choose to **store data locally**. This data may not be served or exchanged with any other component in the architecture, and is considered local data only accessible and used by the specific component. This data can be cached information, partially processed data objects, buffered input data, and buffered output data, among other applications.

As previously stated, the platform will provide a centralized endpoint (Data Storage and Retrieval System) through which modules will be able to store and retrieve data objects. The following figure 6 shows the logical design of this component, which includes the following items:

- **The Core Data Storage Module**, a web service that encapsulates all the methods that the project’s applications can access for data handling. It comprises the only connection point between the applications and the databases of the project. The module acts as the main read/write interface of the platform and its main responsibility is to route the data requests received into the appropriate database API with the help of a predefined API catalog.
- **A static file directory** that includes any kind of file that is not meant to be stored as a whole into a database (they can be indexed though using records in a DB). Such file examples are the 3D models extracted from the 3D reconstruction task and the multimedia downloaded by the crawling and scraping component. This directory along with the Core Data Storage module must be placed in a dedicated server as their access must be centralized for all the project modules.
- **The database set**, which can be composed of different database technologies storing different types of objects (e.g. Knowledge Base containing RDF triplets, and an SQL database for storing related metadata). Each database can exist in a different machine.
- **Database APIs** which expose storing and retrieval functionalities, without the need of providing direct access to the databases. They may also exist in different machines as they only communicate with the Data Storage module. These database APIs will encapsulate the read/write logic, according to the underlying data storage technology.

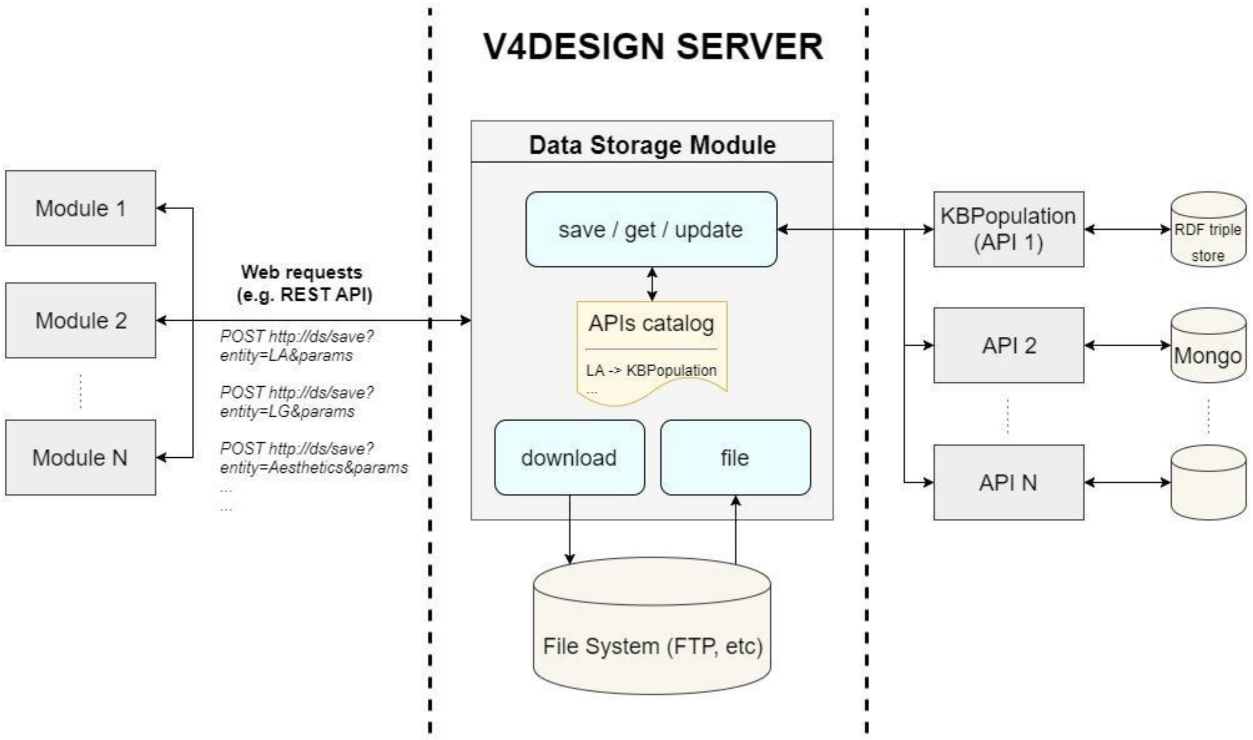


Figure 6: Data storage and retrieval system

The Data Storage module will support the following abstract web methods (the specifics of these methods will be defined according to the storage requirements each component will have):

- *save*: Store a new data object into a database.
- *get*: Retrieve a data object from one database.
- *update*: Modify a data object already existing in a database.
- *download*: Given a URL as input, download the target file in the server’s file system.
- *file*: Retrieve a file existing in the server’s file system.

While calling one of the first three methods, a required parameter is the entity type of the resource to be saved/updated/retrieved. This type will then be mapped into the APIs catalog of the Data Storage module and the request will be redirected to its respective database. Any additional parameters that refer to specific data managing functionalities (e.g. sorting of the retrieved data) must be given using a predefined schema, according to the data that need to be read/written.

The “download” function needs as input only the URL to be downloaded and the directory on which the file will be stored. Last, the latter of the aforementioned parameters is the only one required for the “file” function.

In the proposed architecture, there is no need to directly link the Data Storage and Retrieval System with the Message Bus. Apart from particular exceptions, data shall not be circulated into Message Bus and that is why they can be handled separately. When a module performs a data storage (or update) action, it is responsible to update the Message Bus giving a brief description of the action and how the resource can be retrieved by other modules (for example by providing its identifier).

To better explain how the modules operate using the Data Storage and Retrieval System an example of how two modules of the V4Design system, Crawling/Scraping and Aesthetics extraction, communicate is illustrated in Figure 7. The exact steps of the whole process are numbered accordingly.

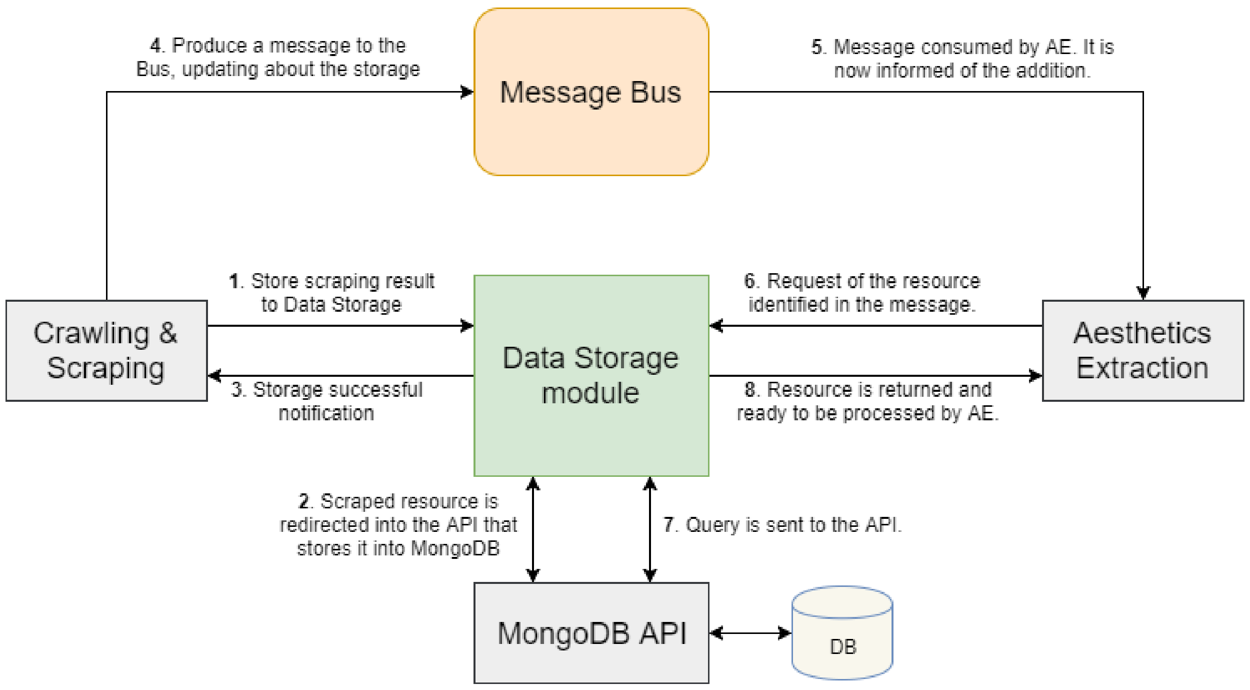

Message Bus
4. Produce a message to the Bus, updating about the storage
5. Message consumed by AE. It is now informed of the addition.
Crawling & Scraping
1. Store scraping result to Data Storage
3. Storage successful notification
Data Storage module
6. Request of the resource identified in the message.
8. Resource is returned and ready to be processed by AE.
Aesthetics Extraction
2. Scraped resource is redirected into the API that stores it into MongoDB
7. Query is sent to the API.
MongoDB API
DB

Figure 7: Example communication between modules

The data storage and retrieval component will implement the functionalities required by the following URs.

TR NB	Description	Function	Function performed	Related URs
TR_DS_1	Data Storing	Data push	Sending and storing of resource(s) to a target database.	UR_1, UR_59, UR_62
TR_DS_2	Data retrieval	Data pull	Retrieval of resource(s) from a target database.	UR_2, UR_3, UR_4, UR_35, UR_37, UR_51, UR_69, UR_70

5 ELEMENTARY REQUIREMENTS AND SPECIFICATIONS OF THE V4DESIGN COMPONENTS

In the following chapter we define the technical requirements and technical specifications for each service and system component that ought to be integrated in the V4Design platform. These requirements have been extracted from the component definition survey and the requirements definition questionnaire, which were circulated and filled by the developments teams. They would serve as guidelines for the development, deployment, and evaluation of these components and services.

In discussing these requirements and specifications, we discern between the platform services, its middleware components, and the user tools that it supports, since each type vary in the manner by which requirements are described, considered relevant, and used to support the development activities. For instance, interfacing with the platform's message bus is a primary concern for the V4Design services, but is irrelevant for the user tools. On the other hand, user tools have a diversified set of functions to perform in order to support a rich user experience, while each service tend to be cantered on supporting a limited number of function.

Therefore, we first discuss the technical requirements and specifications related to the V4Design services, then we discuss the middleware components, and finally we address the user tools.

5.1 Technical requirements and specifications of V4Design services

According to the V4Design concept and architecture design, eight different services are planned to be developed and integrated to provide support for the specialized processes encapsulated in the platform. Together, these services are responsible for ingesting raw data and transforming it into the data objects required by the user.

In the following, we discuss the technical requirements and specifications of each of these services separately, addressing the technologies and components involved in its development and required for its deployment and proper functioning. We describe its functions and inner logical components, and relate them to the elementary user requirements defined in section 2 and in Appendix D of this document. In addition, we discuss the manner by which the service manages data, as well as its non-functional requirements such as security, scalability, reliability and capacity. Finally, we also describe how it will interface with other services through the platform's message bus.

5.1.1 Language Analysis - UPF

The Language Analysis module addresses the analysis and capture of the natural language textual material into structured, ontological representations, so that appropriate system responses can subsequently be inferred by the reasoning module (CERTH), and that textual summaries can be produced (TALN-Language Generation). The module combines multilingual dependency parsers and lexical resources, and a projection of the extracted dependency-based linguistic representations into ontological ones.

Development environment

Java, C++, Python, Graph-transduction grammars

Minimum hardware requirements

Operating System: Linux CPU: unspecified RAM: 10GB Disk Space: 10GB

Server software requirements

Linux system with Docker (>=18.x), preferably with Swarm or Kubernetes for multi-container deployment. Software developed in Java (>=8.x), based on UIMA.

Global functions

Function	Description	Data input	Data output	Components
Language Analysis	Full text analysis pipeline	Raw text	Language-independent abstract structures, syntactic and semantic annotations on top of the sentences (JSON)	Linguistic Analysis, Concept extraction, Relation Extraction

Functional/technical components

Component name	Data input	Data output	Function(s) performed
Linguistic Analysis	Raw text	Annotated text	Tokenization, Part-of-speech tagging, Lemmatization, Surface-syntactic parsing
Concept extraction	Annotated text	Annotated text	Word Sense Disambiguation, Entity linking
Relation Extraction	Annotated text	Language-independent abstract structures	Deep-syntactic parsing, Conceptual relation extraction

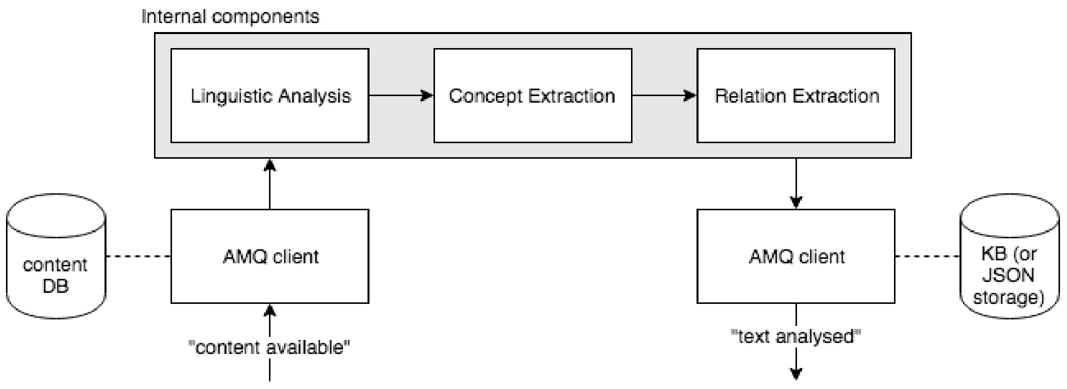


Figure 8: Logical design of Language Analysis

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
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	UR_10, UR_11, UR_12, UR_13, UR_14, UR_15, UR_16, UR_17, UR_18, UR_19, UR_20, UR_21, UR_23, UR_35, UR_56, UR_57, UR_64
TR_LA_2		Concept extraction	Word Sense Disambiguation, Entity linking	
TR_LA_3		Relation Extraction	Deep-syntactic parsing, Conceptual relation extraction	

Data output

Data object name: Language analysis results
Service function : Language analysis
Data schema: not yet fully defined, JSON structure

Messages Encoder language

Java

Messages sent

Message name	Function	Receiver(s)	Related data objects
"Text analysed"	Language Analysis	KB population, Summarization	Text Analysis results

Messages received

Message name	Function	Sender	Related data objects
"New content available"	Language Analysis	Data storage	Input text

Queuing solution

Not foreseen, relying on AMQ for queueing

Expected processing capacity

~ 5 to 10 requests/minute per scaleout node

Expected Availability and reliability

100%

Security policy

Relies on the authentication mechanism of the message bus, no additional security policy is implemented.

Scalability policy

Supports both horizontal and vertical scaling

Local storage solution

The Language Analysis component does not use any local storage.

5.1.2 **Language Generation - UPF**

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 starts from abstract representations, modelled, e.g., as RDF triples, which are stored in a semantic repository. LG follows a request for a summary of most relevant contents related to a specific keyword (or entity), or comes along a generated 3D model.

Development environment

Java, C++, Python, Graph-transduction grammars

Minimum hardware requirements

Operating System: Linux
CPU: unspecified
RAM: 10GB
Disk Space: 10GB

Server software requirements

Linux system with Docker (>=18.x), preferably with Swarm or Kubernetes for multi-container deployment. Software developed in Java (>=8.x), based on UIMA.

Global functions

Function	Description	Data input	Data output	Components
Language Generation	Generate textual reports, descriptions, or summaries	Output from Text Analysis, Knowledge Base	Text	Text Planning, Linguistic Generation

Functional/technical components

Component name	Data input	Data output	Function(s) performed
Text Planning	Output from Text Analysis or Knowledge Base	Ordered sequence of linguistic predicate argument	identify contents related to the queried entity, assesses their relevance relative to this entity

Linguistic Generation	Ordered sequence of linguistic predicate argument	Text	Generates text in target language

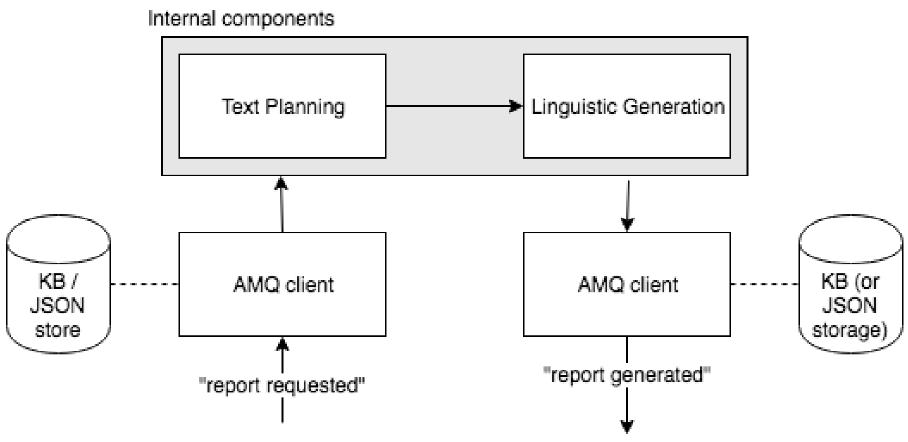


Figure 9: Logical design of Language Generation

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
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	UR_21, UR_23, UR_57, UR_64
TR_LG_2	Render the selected content as text.	Linguistic Generation	Generates text in target language	UR_10, UR_11, UR_12, UR_13, UR_14, UR_15, UR_16, UR_17, UR_18, UR_19, UR_20, UR_21, UR_23, UR_57, UR_64

Data output

Data object name: Language generation results
Service function: Language generation
Data schema: not yet fully defined, plain text or JSON structure

Messages Encoder language

Java

Messages sent

Message name	Function	Receiver(s)	Related data objects
"Report generated"	Language generation	Broadcast	Textual report

Messages received

Message name	Function	Sender	Related data objects
"Report requested"	Language generation	TBD	Pointer to information in KB

Queuing solution

Not foreseen, relying on AMQ for queueing

Expected processing capacity

~ 5 to 10 requests/minute per scaleout node

Expected Availability and reliability

100%

Security policy

Relies on the authentication mechanism of the message bus, no additional security policy is implemented.

Scalability policy

Supports both horizontal and vertical scaling

Local storage solution

The Language Generation component does not use any local storage.

5.1.3 **V4Design Crawler - CERTH**

Desktop app that includes all the crawling and scraping functionalities envisioned in the project, and defined under T2.1 in the project document. It encapsulates web crawling tools designed to extract freely available textual and visual content from open web resources, including from social media.

Development environment

Java

Minimum hardware requirements

Operating System: Windows
CPU:
RAM: 8GB
Disk Space: >20GB

Server software requirements

Java 8

MongoDB >=3.4

Global functions

Function	Description	Data input	Data output	Components
Crawling & scraping pipeline	Executes the service procedure. Intended to be run offline.	Web entry points or queries.	Content from web resources saved into a database.	All Components.

Functional/technical components

Component name	Data input	Data output	Function(s) performed
Web crawling	Web entry points	Discovered nodes	Discovers nodes to scrap
Query expansion	Queries in the form of keywords	Extended version of the input queries	Discovery of extra keywords relevant to the input query
Web search	Queries in the form of keywords	Discovered nodes (and possibly their content)	Depending on the available APIs, scraping may also be performed
Web scraping	Web resource URLs	Web pages content (textual and visual)	Extracts content from web pages
Social media crawling & scraping	Queries (e.g. hashtags, user accounts)	Social media posts	Extracts content from social media
Resource filtering	Resources (web & social media) along with their scraped content	A subset of the input that is determined as suitable for out informational needs	Application of classifiers that categorize the resources as appropriate or not for our purposes

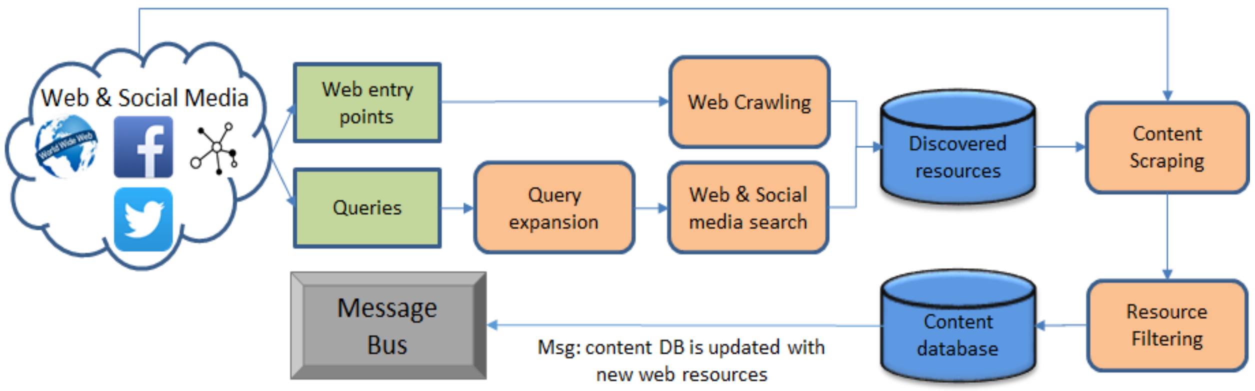


Figure 10: Logical design of a V4Design Crawler

Related technical requirements

TR NB	Description	Function	Function performed	Related URs

TR_CR_1	Using a set of URLs as web entry points, collect all the hyperlinked URLs, up to a predefined depth.	Web crawling	Discovers nodes to scrape	UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56
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	Extracting content 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	UR_10, UR_16, UR_21, UR_24, UR_55
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	

Data output

Arbitrary data objects are generated that include exactly the fields needed to store for the initial infrastructure. The fields we store for each entity are listed below:
- Webpage: URL, raw content (in html), metadata, textual content (plain text)
- Image: original URL, thumbnail URL, source webpage URL, caption, title, tags
- Twitter: raw text of the post

In order to have a unified data model for any type of multimedia, we plan to use the SIMMO model https://github.com/MKLab-ITI/simmo defined in D6.1. Any adaptations or enhancements required for the purposes of this project will be performed.

Messages Encoder language

Java + ApacheMQ JMS API

Messages sent

Message name	Function	Receiver(s)	Related data objects
Web_resources_updated	Crawling & scraping pipeline	WP3 and WP4 modules	Most probably SIMMOs

Messages received

Crawling & scraping comprises the starting point for a processing pipeline, so no messages are going to be received.

Queuing solution

Not foreseen, and not relevant if the Crawler does not receive messages

Expected processing capacity

Difficult to predict due to dependence on external factors, will be calculated empirically

Expected Availability and reliability

100%

Security policy

Relies on the authentication mechanism of the message bus, no additional security policy is implemented.

Scalability policy

Supports both horizontal and vertical scaling.

Local storage solution

Database: MongoDB Version: >=3.4

Data schema

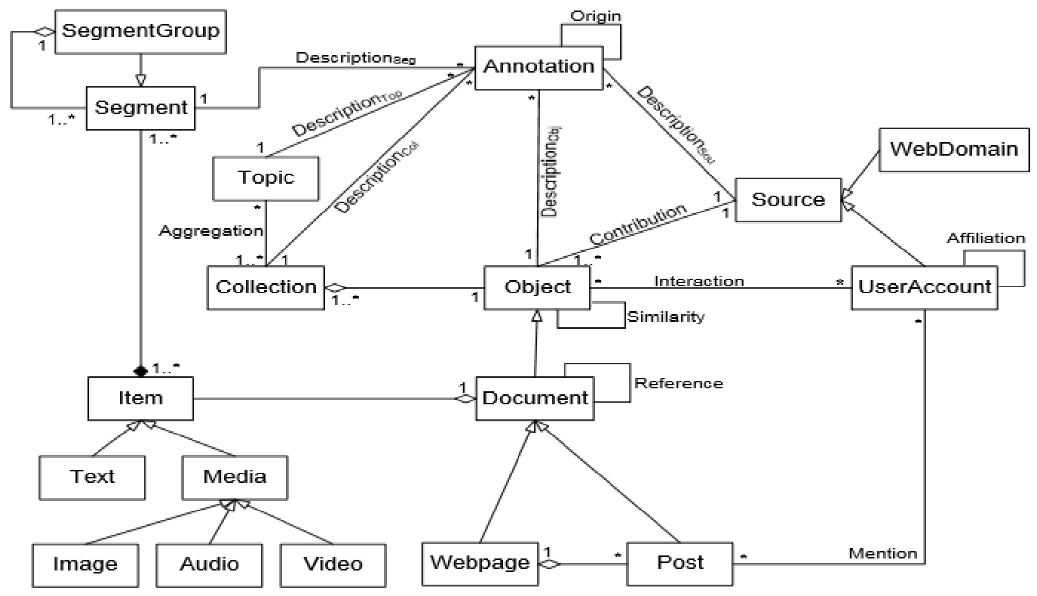


Figure 11: Data schema (SIMMO-based) of V4Design Crawler

5.1.4 Aesthetics Extraction and Texture Proposals (AE&TP) - CERTH

The Aesthetic Extraction (AE) and Texture Proposals (TP) service aims to extract and categorize the aesthetics of paintings and images 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.

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.

Development environment

Python, Tensorflow, Keras

Minimum hardware requirements

Operating System: Windows
CPU: N/A
GPU RAM: 2-3 GB
RAM: N/A
Disk Space: 30 GB

Server software requirements

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

Global functions

Function	Description	Data input	Data output	Components
AE&TP	Extracts the aesthetics from a set of images (paintings, buildings, objects), builds appropriate aesthetics models, and use them to define the aesthetics type of a new image and to perform style transfer and texture proposal.	- A batch of images so as to create the aesthetics models (offline) - An image that contains a building, object or painting to define its type (online)	- Aesthetics models, - Aesthetics type - Texture proposals	AE, TP

Functional/technical components

Component name	Data input	Data output	Function(s) performed
Aesthetics extraction (AE)	- Annotated images with style and creator to build the appropriate model for the buildings, objects and paintings classes - Single image that contain a building, object or painting so as to define its aesthetics category	- Aesthetics models based on style and creator and top-50 images for each category - Annotated image with the appropriate aesthetic label	Aesthetics extraction from paintings, clustering, model extraction and storing on a local file storage
Texture proposals (TP)	Aesthetics model or source image with the desired style and goal image/model	Texture proposal with the opted style	Transfer painting style from the desired image or aesthetic model and pass it to the goal image

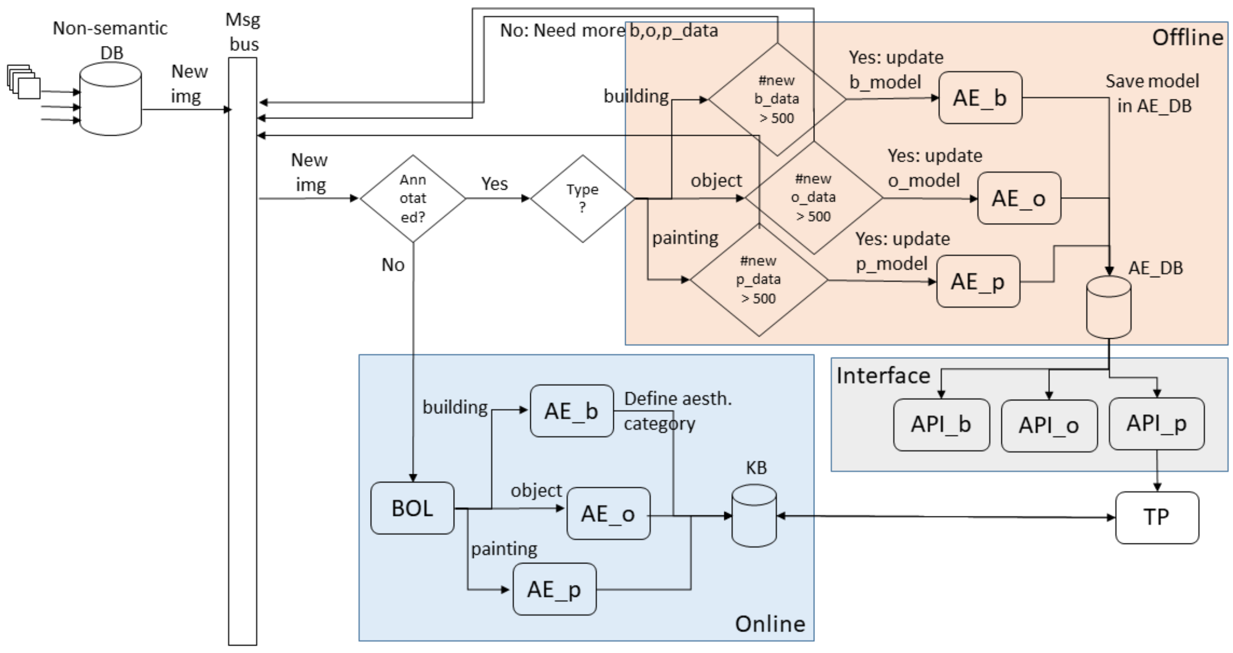


Figure 12: Logical design of Aesthetics Extraction and Texture Proposals

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
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	UR_3, UR_4, UR_41
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	UR_3, UR_4, UR_25, UR_27, UR_41, UR_42

Data output

Data object name: aesthetics_model
Storage format: h5
Data schema:

- 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}

Data object name: texture_proposal
Storage format: JPEG/ PNG
Data schema:

- Field name: texture
- Type: image
- Allowed values: 8bit image

Messages Encoder language

Java (possibly the same with crawler service)

Messages sent

Message name	Function	Receiver(s)	Related data objects
New_Aesth_model	AE&TP	TP service, Semantics service,	aesthetics_model
Update_KB	AE&TB	Semantics service	Input_image
New_texture_proposal	AE&TP	Semantics service	texture_proposal

Messages received

Message name	Function	Sender	Related data objects
New_img	Notifies AE module that a new image exist in the File Storage	Message_bus	Input_image

Queuing solution

Planned for M10

Expected processing capacity

1 request at a time

Expected Availability and reliability

Processes 1 request at a time, unavailable when processing

Security policy

Relies on the authentication mechanism of the message bus, no additional security policy is implemented.

Scalability policy

The number of requests and the processing time are linearly dependent

Local storage solution

File system storage. Data object name: paintings Storage format: {JPEG, PNG} Estimated disk space: ~1G for each aesthetic model, ~0.5G for the style transfer model, ~30G for saving the images and their annotations that will be used to create the aesthetics model.

5.1.5 **KB Population - CERTH**

The KB population service is responsible for mapping the incoming information from the different V4Design modules 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) extracted from textual analysis, along with 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 service will support different mapping services, according to the format of the input that we will get from the other components, e.g. XML, JSON, etc. The service will be also responsible for updating the KB with information coming from structured repositories, such as the Europeana API.

Development environment

Java

Minimum hardware requirements

Operating System: Windows 10 CPU: Intel® Xeon® Silver 4108 RAM: >5GB Disk Space: >50GB

Server software requirements

Apache Tomcat Java 8 GraphDB

Global functions

Function	Description	Data input	Data output	Components
setData	remote method to upload data to the KB	analysis results from various modules	updated KB (RDF triples that correspond to the incoming data)	KBPopulation

Functional/technical components

Component name	Data input	Data output	Function(s) performed
KBPopulation	analysis results from various modules	updated KB (RDF triples that correspond to the incoming data)	mapping to RDF

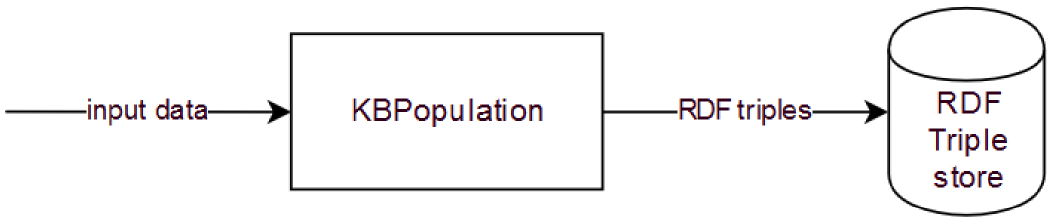


Figure 13: Logical design of KB Population

Related technical requirements

There are no requirements directly involving KBPopulation. The service is potentially relevant to all requirements, since it stores metadata generated by other services.

TR NB	Description	Function	Function performed	Related URs
TR_KB_1	Map analysis results from other modules	KBPopulation	RDF mapping and KB population	UR_2
TR_KB_2	Provide an API over the KB for querying metadata			
TR_KB_3	Map metadata about texture resolution			UR_3
TR_KB_4	Map analysis results from building localisation			UR_10, UR_20, UR_57
TR_KB_5	Map analysis results from object localisation			
TR_KB_6	Map analysis results from aesthetics			
TR_KB_7	Map analysis results from text analysis			
TR_KB_8	Map analysis results from reasoning			
TR_KB_9	Map metadata about quality			UR_12, UR_37
TR_KB_10	Map geo-location of assets			UR_15
TR_KB_11	Map date (creation date)			
TR_KB_12	Map author info			UR_16
TR_KB_13	Map copyright info			
TR_KB_14	Map visible colours			UR_18
TR_KB_15	Map metadata coming from 3D model reconstruction			UR_20
TR_KB_16	Map results from text generation			UR_21
TR_KB_17	Ability to associate assets with relevant external Web Pages			UR_22
TR_KB_18	Map results from text generation			UR_23
TR_KB_19	Associate assets with preview thumbnails			UR_30
TR_KB_20	Ability to map texture mayerial metadata			UR_41
TR_KB_21	Support the linking of assets with relevant ones			UR_50

TR_KB_22	Support the annotation of assets with reuse rights and copyrights			UR_55
TR_KB_23	Map analysis results from text generation			UR_57

Data output

The component generated RDF triples, following the V4Design ontologies that will be developed. An initial model can be found here

Messages Encoder language

JAVA + RDF4J (rdf4j.org/), Gson (https://github.com/google/gson) and perhaps an XML parser

Messages sent

Message name	Function	Receiver(s)	Related data objects
DataUploaded	setData	ReasoningService	the relevant triples (?)

Messages received

Message name	Function	Sender	Related data objects
DataAvailable	setData	some component which has results to provide	it depends on the component

Queuing solution

Not currently contemplated, but could be necessary in a later stage

Expected processing capacity

At the beginning, the service will not support concurrency, e.g. any request to map data will be processed only when the previous one is finished. If needed, concurrency can be supported.

Expected Availability and reliability

100%

Security policy

Basic local authentication (username/password) can be supported

Scalability policy

Not sure yet.

Local storage solution

Native RDF triple store, GraphDB V8

5.1.6 Semantic Integration and Reasoning - CERTH

The reasoning service will be responsible for further analysing the knowledge captured in the Knowledge Base. More precisely, the module will try to build 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. To this end, this module will develop the context-aware reasoning and information coupling algorithms operating on-top of the available ontological knowledge built by the KB Population module, supporting the decision-support aspects of the V4Design platform, according to the use case requirements that will be defined. All in all, the reasoning process aims to derive facts and higher-level implicit knowledge from information already generated by the aforementioned V4Design modules, and asserted in the ontologies, preparing the information to be presented to the user. 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.

Development environment

Java

Minimum hardware requirements

Operating System: Windows 10 CPU: Intel® Xeon® Silver 4108 RAM: >5GB Disk Space: >50GB

Server software requirements

Java 8

Global functions

Function	Description	Data input	Data output	Components
startReasoning	remote method to start the reasoning task	-	updated KB with new knowledge (RDF triples)	ReasoningService
startSearch	remote method to be called in order to retrieve results from the data storage	filtering data from the interface	query results in the form of RDF model	ReasoningService

Functional/technical components

Component name	Data input	Data output	Function(s) performed
ReasoningService	-	updated KB with new knowledge (RDF triples)	derives implicit relations
ReasoningService	filters	query results in the form of RDF model	query formulation / enrichment

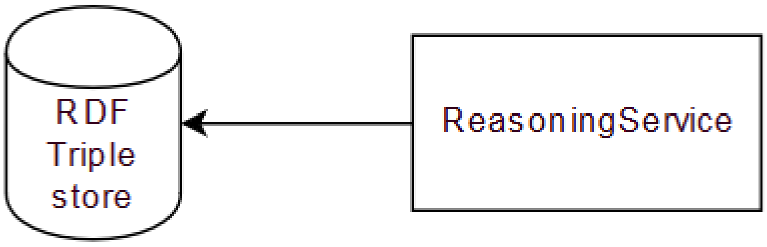


Figure 14: Logical design of Reasoning

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
TR_RQ_1	Support searching functionality (translation of user requests into one or more queries over the data storage	Reasoning Service	query formulation / enrichment	UR_2, UR_50
TR_RQ_2	Infer geolocation from location tag		Inference of implicit relations and context	UR_15
TR_RQ_3	Propagate annotations from other modalities to the 3D models		Inference of implicit relations and context	UR_20
TR_RQ_4	Find relevant external Web page, based on the annotation provided by other components		Inference of implicit relations and context	UR_22
TR_RQ_5	couple searching functionality with text analysis on the keywords		query formulation / enrichment	UR_35
TR_RQ_6	Find assets relevant to other assets		Inference of implicit relations and context	UR_50

Data output

The component generates RDF triples, following the V4Design ontologies that will be developed. An initial model can be found here.

Messages Encoder language

JAVA and SPIN rules (http://spinrdf.org/)

Messages sent

Message name	Function	Receiver(s)	Related data objects
ReasoningFinished	startReasoning	TBD	-
SearchFinished	startSearch	GUI	TBD

Messages received

Message name	Function	Sender	Related data objects
DataUploaded	startReasoning	KBPopulation	-
SearchRequestAvailable	startSearch	GUI	TBD

Queuing solution

Not sure yet

Expected processing capacity

At the beginning, the service will not support concurrency. If needed, concurrency can be supported

Expected Availability and reliability

100%

Security policy

Basic local authentication (username/password) can be supported

Scalability policy

Not sure yet

Local storage solution

Native RDF triple store, GraphDB V8

5.1.7 Spatio-Temporal Building and Object Localization (STBOL) - CERTH

Spatio-Temporal building and object localization in images and video frames service initially aims to define their type, i.e. whether the image or video contains a building, object or a painting and then semantically segment 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 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.

Development environment

Python, Tensorflow, Keras

Minimum hardware requirements

Operating System: Linux OS
CPU: N/A
GPU RAM: 2-3GB RAM: N/A
Disk Space: 30GB

Server software requirements

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

Global functions

Function	Description	Data input	Data output	Components
STBOL	Defines whether a video or image contains a building, object or painting and segment them in a spatio-temporal manner	Video or image that contain buildings, object or painting	- Define the type of the given data (building, object, painting, other)	BOL, STBOL

Functional/technical components

Component name	Data input	Data output	Function(s) performed
BOL	-Video - Image	Tagged image/video with the detected type of interest	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
STBOL	- Video - Image	Binary masks that define regions of interest on the given data	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

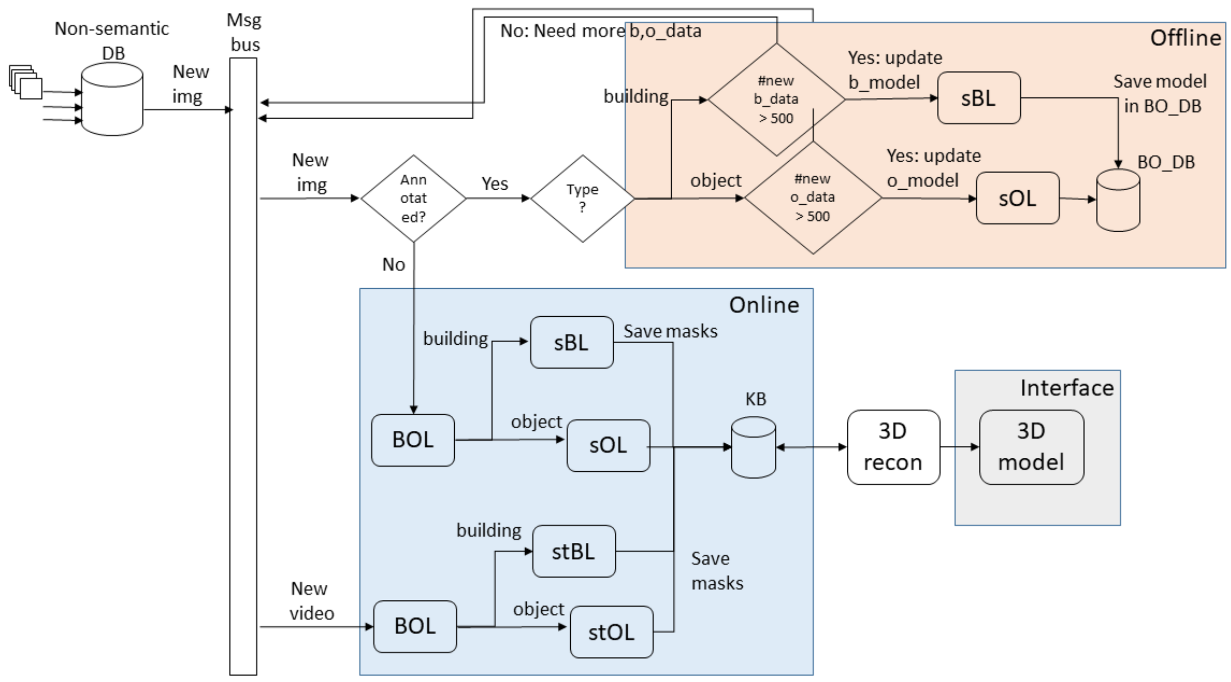


Figure 15: Logical design of Object Localization

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
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	UR_63
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	UR_25, UR_63

Data output

Data object name: stBOL_model

Storage format: h5

Data schema:

{alley;amphitheater;apartment_building;aqueduct;arcade;arch;archaelogical_excavation;arch ive;auditorium;balcony;barn;barndoor;bazaar;beach_house;boathouse;bridge;building_facad e;bus_station;campus;castle;catacomb;cemetery;church;construction_site;corridor;dam;dep artment_store;downtown;gas_station;general_store;gift_shop;harbor;hospital;hotel;house;i ndustrial_area;inn;lighthouse;mansion;manufactured_home;mosque;motel;museum;natural _history_museum;oast_house;palace;parking_lot;pavilion;playground;restaurant;schoolhous e;stadium;supermarket;temple;tower;train_station;tree_house;wind_farm;windmill;yard}

Field name: object

Type: DCNN-model (.h5)

Allowed values: {sofa; table; chair; lamp; mug; etc.}

Field name: type

Type: string

Allowed values: {object, building}

Data object name: building_object_mask

Storage format: JPEG/ PNG

Data schema:

Field name: mask

Type: image

Allowed values: 8bit image

Messages Encoder language

Java (probably)

Messages sent

Message name	Function	Receiver(s)	Related data objects
New_Image_Type	BOL	Semantics service	BOL_model
New_Video_type	BOL	Semantics service	BOL_model
New_Image_Mask	STBOL	3D-reconstruction, Semantics service,	stBOL_model
New_Video_Mask	STBOL	3D-reconstruction, Semantics service	stBOL_model

Messages received

Message name	Function	Sender	Related data objects
New_image	Notifies STBOL service that there is a new image to be processed	Message_bus	Input image
New_video	Notigies STBOL service that there is a new video to be processed	Message_bus	Input video

Queuing solution

Planned for M12

Expected processing capacity

1 (Depends on the number of GPUs that are going to be available)

Expected Availability and reliability

Processes 1 request at a time, unavailable when processing

Security policy

No local security policy will be supported

Scalability policy

The number of requests and the processing time are linearly dependent.

Local storage solution

File system storage.

Data object name: buildings, objects

Storage format: {JPEG, PNG}

Estimated disk space: ~1G for each scene recognition model, semantic segmentation model,

~30G for saving the images and their annotations that will be used to create the aforementioned models.

5.1.8 **3D Reconstruction - KUL**

The reconstruction service will be responsible for conversion 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. Multiple output formats will be available for extracted 3D models.

Development environment

Windows 10, Visual studio 2017, C++ , C#

Minimum hardware requirements

Operating System: Windows 10
CPU: Server-grade for high computation (Intel Xeon, AMD epyc)
GPU: High-end Nvidia graphics card (>=GTX 1080)
RAM: >32GB
Disk Space: >1TB

Server software requirements

.net core, framework redistribution (latest version)
Microsoft visual C redistribution (2017)

Global functions

Function	Description	Data input	Data output	Components
Analyze	Initiate analysis on data wrt reconstruction capabilities. yes/no output.	Video, images, visual analysis input	Internal format to determine multi/single view reconstruction.	Reconstruction AnalysisService
InitializeReconstruction	@ succesfull analysis a reconstruction may be initialized.	Reference to Analyze output (layout details tbd)	handle/reference to a reconstruction object	MultiView Reconstuction
ProcessReconstruction	Any processing can be requested for a reconstruction, depending on data input	- Reconstruction ref/handle - Flags / type of processing to be requested. Could - (optional) settings: for further tinkering of the process if so desired		MultiView Reconstuction

Functional/technical components

Component name	Data input	Data output	Function(s) performed
ReconstructionAnalysisService	Video, images, visual analysis input	Internal format to determine multi/single view reconstruction	derives implicit relations
MultiViewReconstuctionService	Output of ReconstructionAnalysisService	3D models, pointclouds	Image matching, reconstruction, format conversion

CommunicationService	Local storage	-	Rest-api

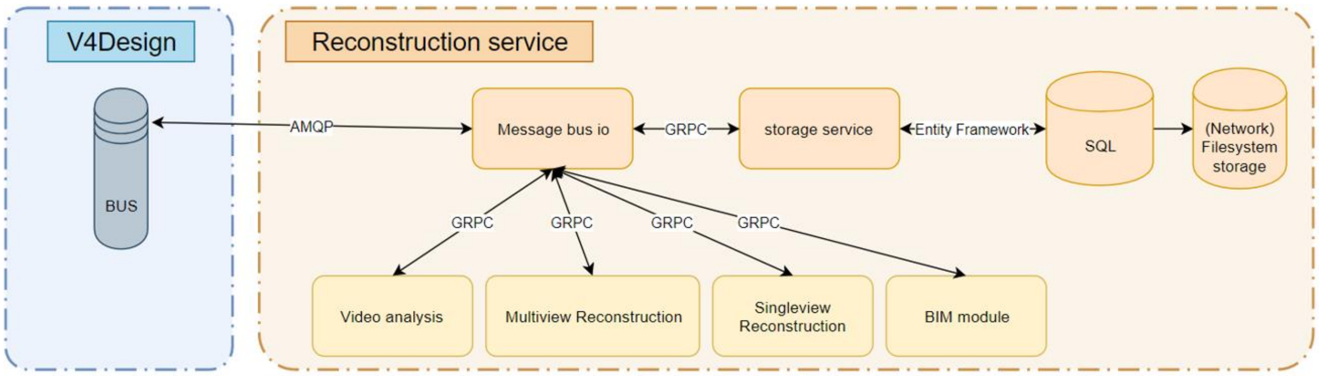


Figure 16: Logical design of 3D Reconstruction

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
TR_3D_1	Extract and build a 3D model	Reconstruct	Build a 3D model from the collection of images or video frames	UR-8, UR-2, UR-7, UR-20, UR-6, UR-16

Data output

3D model with metadata

Messages Encoder language

.net core/framework

Messages sent

Message name	Function	Receiver(s)	Related data objects
ReconstructionStart	Reconstruct	TBD	Raw input data
ReconstructionEnd	Reconstruct	TBD	TBD
ReconstructionUpdate	Reconstruct	TBD	New output data: models, pointclouds, etc.

Messages received

Message name	Function	Sender	Related data objects

VisualAnalysisDone	ReconstructionAnalysis	TBD	TBD

Queuing solution

The service includes a queuing solution

Expected processing capacity

Susceptible towards request image/video size and needs further experimenting

Expected Availability and reliability

Once service reaches 100% computing capacity, new requests will be put on hold

Security policy

No local security policy will be supported

Scalability policy

Horizontal scaling possible if service is deployed on multiple machine

Local storage solution

File system

5.2 Technical requirements and specifications of V4Design middleware components

The V4Design platform's architecture design defines two middleware components, being the V4Design REST API and the message bus.

5.2.1 The V4Design message bus - McNeel

The V4Design message bus is in charge of integrating the different components of the V4Design platform, supporting real-time communication between them. It allows the platform to adopt a distributed architecture model by which different components can be hosted on different servers, developed under different frameworks, and serviced by different teams. This loose integration model is key to the successful evolution of the platform into a high-end operational service architecture. Therefore, the message bus is a key component that is responsible to guarantee the proper execution of the platform processes, beyond the responsibilities of its services.

Development environment

Java

Minimum hardware requirements

Operating System: Ubuntu Linux CPU: single CPU >2Ghz RAM: 1 GB Disk Space: 0.5 GB

Server software requirements

Java Runtime Environment - JRE 1.7 Maven 3.0.0 build system Apache ActiveMQ broker engine, V 5.15.0

Supported messaging protocols

AMQP, in addition to OpenWire, STOMP, MQTT, WSS

Supported message types

Message Type	Description	Message bus action
Broadcast	a service broadcasts a message to inform other services of changes in its status-quo, including the arrival of new data, the completion of a process, and so on.	The message bus will relay broadcast messages to all authenticated components.
Service-to-Service	a service places a request to another service, which could include a query for data, a trigger for a function, and so on.	The message will be relayed exclusively to the destination service.
I'm alive	the service informs the message bus that it is up and running	The message bus will confirm authentication.
I'm shutting down / unavailable	the service informs the message bus that it is going offline. The message bus will also periodically check the services reachability automatically.	The message bus will NOT respond.

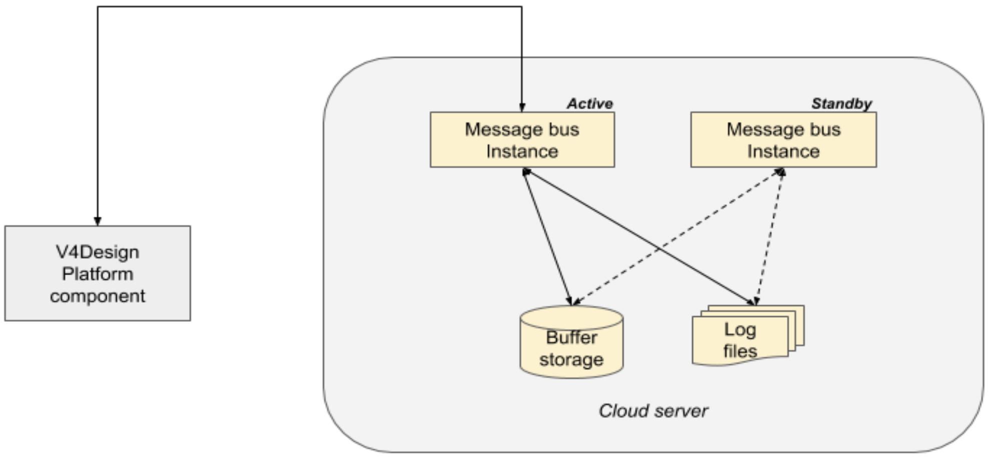


Figure 17: Logical design of Message Bus

Queuing solution

The message bus supports the creation of component-level and architecture-level queues. Messages need not to be queued for processing by the message bus, this is done in real-time.

Expected processing capacity

Maximum performance between 1000 and 2000 messages/second.

Expected Availability and reliability

The message bus will always be available. A redundant instance is installed and placed on standby, and will be automatically activated if the main instance fail.

Security policy

The message bus will support component authentication.

Scalability policy

Horizontal scaling is supported up to 20,000 messages per second.

Local storage solution

The message bus uses a local database to store message and other metadata for increased performance, reliability, and traceability. For this purpose, we use Apache KahaDB as a file based persistence database.

A complete description of this database properties is available at http://activemq.apache.org/kahadb.html

5.2.2 The V4Design REST API - NURO

The V4Design REST API provides the functionality necessary for front-end applications to query and retrieve assets from the V4Design Asset Repository. The RESTful API will provide specific calls to query the Asset Repository 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. Specifically, the V4Design REST API connects to the database in charge of managing the Asset Repository objects without needing to go through the V4Design Message Bus system.

This API will be utilized by both the V4Design front end user interface for Architects and Video Game designers (Rhino3D and Unity plugins). Each of the application plugins will present the user with an interface to enter in any query filters relevant to the assets produced by the V4Design system. Once the filters have been entered, the application will format an API call and transmit this to the API endpoint. The endpoint will respond with a list of potentially relevant assets based on the query filters. The front end applications can then be programmed to respond appropriately to the user by presenting the results, and eventually making the results available for download.

Development environment

Linux, node.js, Swagger or Apiary

Minimum hardware requirements

Operating System: Linux
CPU: 4 Cores+
RAM: 8GB +
Disk Space: 100GB+

Server software requirements

Apache Webserver on a Linux based OS with MySQL database. PHP as fpm if possible (not mod_php)

Supported call protocols

REST APIs

Supported call types

A complete list of calls supported by the API, alongside the specifications of each call are included in Appendix E.

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
TR_RA_1	Create a user profile	Create user	Helps user tools to create users	UR_022
TR_RA_2	Create login credentials for a user	User Login	Helps user tools to login to the system and authenticate them	
TR_RA_3	Create new asset	Create Asset	Helps user tools to create asset details	UR_1, UR_59, UR_62, UR_014, UR_008
TR_RA_4	Upload new asset	Upload Asset	Helps users to upload assets	
TR_RA_5	Get latest asset (texture, 3D model) from DB	Get Latest assets	Gets the latest asets from the database	UR_2, UR_3, UR_4, UR_35, UR_37, UR_51, UR_69, UR_70
TR_RA_6	Search V4Design DB for assets	Search	Searches the database for assets based on a specific field	UR_2, UR_3, UR_4, UR_35, UR_37,

				UR_51, UR_69, UR_70
TR_RA_7	Rate an asset	Rating	Rates an asset	UR_014, UR_008
TR_RA_8	Comment an asset	Comments	Adds a comment to the asset	

Queuing solution

No queuing solution is contemplated

Expected processing capacity

Depends on CPU

Expected Availability and reliability

Depends on network connection. Best case 100%

Security policy

OAuth 2.0

Scalability policy

Not specified

Local storage solution

Database with MySQL

5.3 Technical requirements and specifications of V4Design user tools

According to the development plan of the V4Design platform, two user tools are planned to be developed and integrated in the platform to interface with the user profiles defined in the use cases and use scenarios. In the following, we discuss the technical requirements and specifications of the tools.

5.3.1 V4Design for Rhino - McNeel

The V4Design for Rhino project is a plugin for the Rhinoceros 3D (Rhino) software platform developed by Robert McNeel & Associates. Rhino is typically used by design professionals to produce 3d models of objects, spaces, buildings, urban environments, etc. Rhino includes accurate NURBS as part of its geometry kernel, and thus, the models produced are useful for fabricating accurate physical representations of the 3d models. Due to this, Rhino is used by architects to produce 3D models of their building designs.

The V4Design for Rhino plugin will present the Rhino user with a GUI capable of querying the V4Design asset repository. The user will be able to search for V4Design assets by a host of asset metadata such as asset location, asset type (3d model, image, etc.), and other relevant metadata. Once the query has been entered, the results of the query will be presented to the user through the V4Design for Rhino GUI. These results should include a graphical way to review the results, including a thumbnail image of the asset, textual description of the asset, and any other relevant data that can be useful to the user when selecting an asset from the query results. The user will then be able to introduce this asset into the Rhino modelling environment for further interrogation, manipulation, etc.

User profiles

Profile name: basic

Role description: a normal Rhino user. Such a user would be able to access the V4Design assets from the plugin in Rhino. The user can query the asset repository and eventually download (read) the asset to be included into the Rhino model. The user would be able to create personal libraries of models from the V4Design asset repository, which are essentially lists of V4Design asset URIs. These personal libraries are stored along with their user profile.

Permissions: Read (access V4Design assets), Evaluate assets (like a particular asset)

Restrictions: None for the moment, could change later.

User Authentication mechanism

There are two cases where the V4D4Rhino plugin might require user authentication:

1- Tracking liked V4Design assets

2- Tracking personal V4Design asset libraries

These cases should be evaluated in relation to the effort required to provide this functionality. Alternatively, such data could be stored locally on the user's machine. The advantages of this are that there would then be no need for user authentication. The disadvantage of this is that the user would only be able to access this personal information from the machine on which they have used the V4D4Rhino plugin.

If it is decided that user authentication is necessary from the V4D4Rhino plugin, the plugin could use the user's associated "Rhino Account" which is how users can 'log in' to Rhino and thus validate they are the owner of the Rhino license. Rhino Accounts is now available to use for third party developers as a way to authenticate users. The system allows a user to link either their Google or Facebook accounts.

Rhino Accounts also allows for two factor authentication.

Usage / Deployment environment

Works on top of Rhino, or as a Rhino based standalone application.

The V4D4Rhino plugin would be installed via the Rhino Installer Engine, which is a separate executable installed along with Rhino 3d. This executable registers the .rhi extension on Windows and the .macrhi extension on macOS. The installation package is merely a compressed archive (.zip) with the contents of the compiled plugin and any resources needed to support the plugin. The V4D4Rhino.rhi or .macrhi package can be installed by double clicking on the file. This initiates the Rhino Installer Engine to evaluate the package. Once the package has passed the evaluation stage, the contents are copied in the appropriate locations on the user's hard drive. The next time the user opens Rhino, the V4D4Rhino functionality can be accessed. The V4D4Rhino plugin installer will be made available through Rhino's repository of plugins at https://food4rhino.com

More information on Rhino Plugin Installers: Windows | macOS

Alternatively, the V4D4Rhino plugin can be distributed on Yak, Rhino's package manager. Yak is accessed from within Rhino. Yak is an experimental project in Rhino 6, and will be a feature of Rhino 7. Users can access the package manager with a command in Rhino that brings up a window with the available packages. A user would see the V4D4Rhino package (or if they already have it installed, they might see if there is an update), as well as options to install it. Once installed from the package manager, the V4D4Rhino functionality will be available.

More information on the Rhino Package Manager:
https://developer.rhino3d.com/guides/yak/

Prerequisites in deployment environment

The V4D4Rhino requires Rhino 6 either running as an evaluation license (90 days) or as a commercial or educational license.

The user will need to register for a Rhino Account.

Technical specifications of deployment environment

Operating System: Windows 10, 8.1, or 7 SP1 or macOS 10.13.x
CPU: Intel i5, i7, etc.
RAM: 8 GB
Disk Space: 600 MB
Others:

- Internet connection for Rhino Accounts
- OpenGL 4.1 capable video card is recommended.

More information in technical requirements:
https://www.rhino3d.com/6/system_requirements

Not supported operating systems:

- Linux
- Windows 8
- Windows XP 64-bit
- Windows Vista, NT, 95, 98, ME, or 2000
- Windows 32-bit all versions
- Virtualization systems on OS X such as VMWare and Parallels
- OS X 10.10.4 (early versions of Yosemite) and any earlier versions

Usage limitations

The tool is made to work locally, one instance per user, so no scalability or multi-user support

is required. This concern is more pertinent to the V4Design backend architecture.

Logical components

Component name	Data input	Data output	Function(s) performed
Query Composer	User parameters	User Query	FormatQuery
Connector	User Query	API Call	callAPI
Listener	Query Results	Serialized Query Results	readResults
Results Converter	Serialized Query Results	Rhino Model	buildModel
modelVisor	buildModel	N/A	User interaction

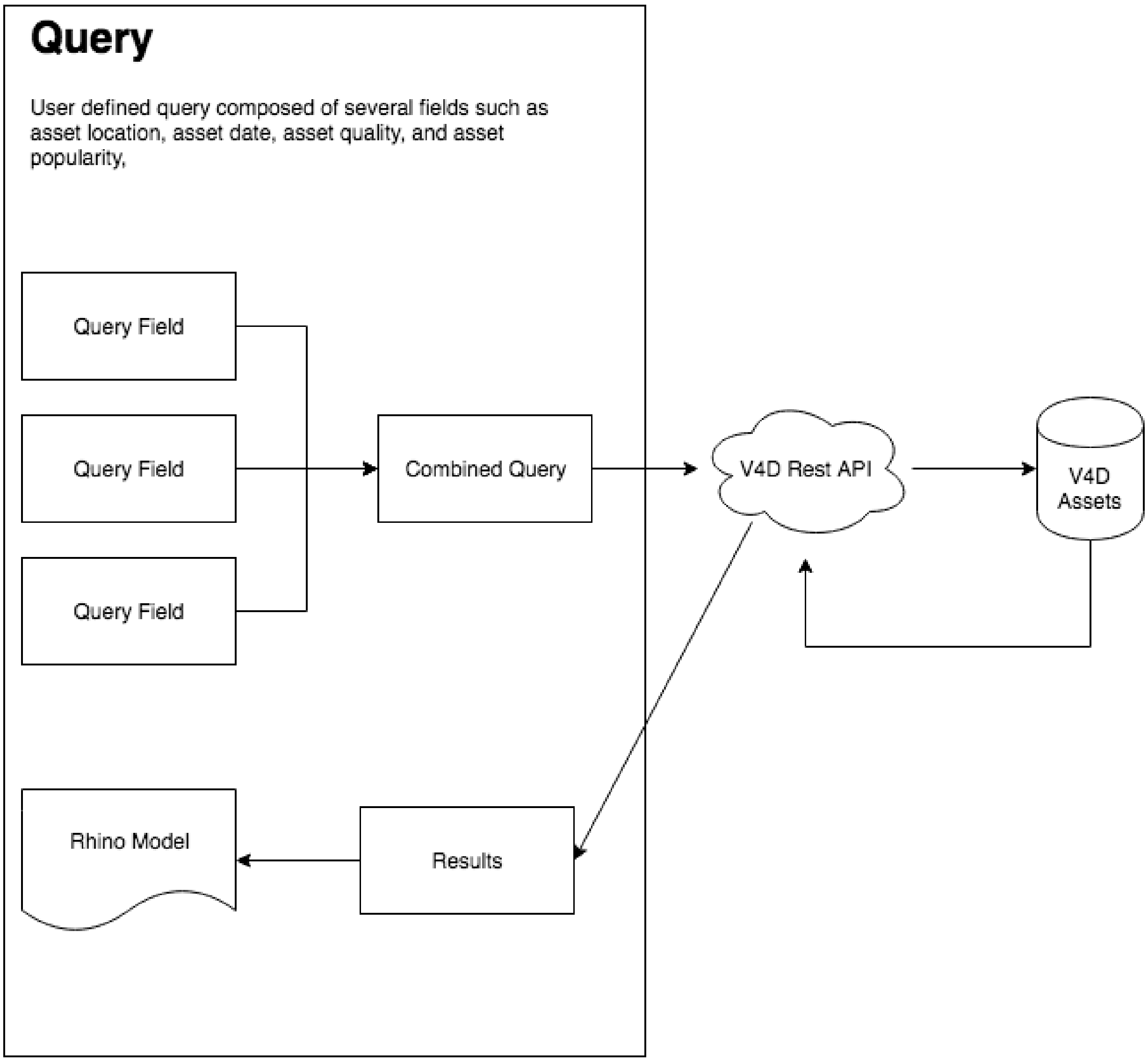

Query
User defined query composed of several fields such as asset location, asset date, asset quality, and asset popularity,
Query Field
Query Field
Query Field
Combined Query
V4D Rest API
V4D Assets
Rhino Model
Results

Figure 18: Logical design of V4Design for Rhino

Supported functionalities

Functionality	Description	Data Objects
Query	Create query string to interrogate the V4Design asset repository.	JSON formatted request to the V4Design Rest API. Returns results based on query as a JSON formatted reply.
Discover	View V4Design assets previewed with a thumbnail and basic description.	
Evaluate	Like an asset	JSON formatted message to the V4Design Rest API to record this evaluation. JSON formatted result stating that the evaluation has been recorded.
Collect	Create personal collections or libraries of objects.	JSON formatted collection of asset ids.
Include	Include V4Design asset in Rhino model	URI to the V4Design asset so that Rhino can download it and include it in the model.

Related elementary user requirements

UR NB	HLUR NB	Description
UR_30	HLUR_203 HLUR_207	As an Architect I want UIX: 3D-gallery i.e. A distraction free interface with rendered preview thumbnails.[2]
UR_32	HLUR_203 HLUR_207 HLUR_208	As an Architect I want a UIX a detailed view of a Gallery of 3D model (with/without texture) and usage examples from other users
UR_33	HLUR_203 HLUR_207 HLUR_208	As an Architect I want UIX a detailed view of Additional data: palette of visible colours + bounding box size, author, copyright
UR_34	HLUR_203 HLUR_207 HLUR_208	As an Architect I want a UIX a detailed view of Team/Public relation functions: Rating system, Personal notes/ marking/ save to favourites, share functionality for social media.
UR_36	HLUR_203 HLUR_207	As an Architect I want UIX of Tags organized in tree structure and search field for typing tag. Personal tags (non-public tags)
UR_37	HLUR_203 HLUR_207	As an Architect I want UIX: Detailed search by features: - Quality (3D model/ texture), Footage features, augmented data

UR_38	HLUR_203	As an Architect I want UIX: Download settings (saveable profiles): - Mesh quality/format, Texture quality/ format/ layers (checkboxes), Material definition file, Colour palette (e.g.: adobe swatches)
UR_46	HLUR_203 HLUR_207 HLUR_208	As an Architect I want Simple and clear visual UI (User Interface). Simple enough for non-specialised users
UR_47	HLUR_203	As an Architect I want have access to the code of the tool
UR_69	HLUR_205 HLUR_207 HLUR_208	Batch Download/Load of assets. Have the ability to load multiple assets at once.

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
TR_RN_1	Retrieve 3D models from V4D Asset Repository	Combined Query	Query and retrieve 3D models of different resolutions from the V4Design repository	UR_002
TR_RN_2	Retrieve textures from V4D Asset Repository		Send boundbox and retrieve the texture delimited by the box	UR_003
TR_RN_3	Retrieve asset metadata		The Query results have their metadata integrated, or made accessible (via URIs)	UR_006, UR_007, UR_039
TR_RN_4	Query V4D asset repository		Query resources by any field	UR_012, UR_013
TR_RN_5	Personal assets library, discover new assets		Query mechanism encourages discovery	UR_025
TR_RN_6	Save asset as different file format	Save assets	Save assets locally in different compatible formats	UR_014, UR_008
TR_RN_7	View V4D 3D models in VR	Asset viewer	View 3D models in an interactive way (rotate, zoom, etc.)	UR_010
TR_RN_8	Preview and markup of V4D assets		Obtain low-resolution instances of images, videos and 3D models	UR_011, UR_009
TR_RN_9	Visualize retrieved V4D assets		View any V4D asset (image, video, 3D model, etc.)	UR_020
TR_RN_10	Simple UI		Intuitive and simple GUI design	UR_022
TR_RN_11	Open source	Tool	Source code to be made available	UR_023

API connector - development language

JavaScript

Development libraries used

Vue.js for UI.

Local Storage

Local storage of preferences, cached assets, etc. will occur on the local file system.

Data objects stored in Local Storage

Data object name: User Defined V4Design Asset Libraries
Storage format: JSON
Data schema:
For each field, please specify:

- Field name: V4Design asset URI array
- Type: string array
- Allowed values: string array with URI to V4Design Asset metadata collected in this user defined library

- Field name: Collection name
- Type: string
- Allowed values: name of the user defined library

5.3.2 VR Authoring tool - NURO

The authoring tool for VR game development will be based on Unity Engine for game development. Unity3D (www.unity3d.com) is a cross-platform game engine primarily used for development of 2D and 3D games. Unity is the most used game engine and is available for free to the community. Games on unity can be developed using C# and other design tools included in the software. Games for 27 different platforms, such as iOS, Android, Windows, PlayStation, Xbox as well as VR devices such as Oculus Rift, Google Cardboard, Steam VR, PlayStation VR, Gear VR, windows Mixed Reality as well as Daydream can be developed using Unity.

User profiles

Profile name: Basic
Role description: a normal unity user. To access V4Design, they will have to login using the tool
Permissions: Dependent on the type of user. Generally it is read and download assets
Restrictions: None

User Authentication mechanism

The users will be tracked for downloading the V4Design assets, liking and reviewing the assets. Also the authorisation of the user's will be done using V4Design user roles defined on the server.

Usage / Deployment environment

As a unity plug-in, the authoring tool will be packaged and any user can download and install the package on their system's unity3d software. The packages will be in DLL format.

Prerequisites in deployment environment

The main prerequisite for the deployment environment is Unity3D, and other prerequisites specified by Unity, namely:
WebGL: Any recent desktop version of Firefox, Chrome, Edge or Safari.
Universal Windows Platform: Windows 10 and a graphics card with DX10 (shader model 4.0) capabilities

Technical specifications of deployment environment

Operating System: Windows 7 SP1+, macOS 10.11+, Ubuntu 12.04+, SteamOS+
CPU: Intel i5, i7, etc. + SSE2 instruction set support.
RAM: 8 GB
Disk Space: 1+ GB
Others: Graphics card with DX10 (shader model 4.0) capabilities. For instance NVIDIA Graphics card (minimum 2 GB memory)

Usage limitations

The tool is made to work locally, one instance per user, so no scalability or multi-user support is required. This concern is more pertinent to the V4Design backend architecture.

Supported functionalities

Functionality	Description	Data Objects
Querying	Querying V4Design database for assets	V4Design assets
Importing	Downloading the binaries of the assets	V4Design assets
Developing	Development of environments using the assets	V4Design assets
Scripting	Attaching predefined scripts to assets	V4Design assets
Uploading	Uploading assets	V4Design assets

Related elementary user requirements

UR NB	HLUR NB	Description

UR_53	HLUR_203 HLUR_208 HLUR_212 HLUR_215	As a Content Provider I want to receive statistics about which items are being seen and/or downloaded from my repository, so I can generate reports on the impact of my content - To know what extent one can re-use and repurpose, and possibly have to attribute, the on-going works
UR_61	HLUR_212	As a content provider I want to have game analytics from the authoring tool - Any game require analytics to better serve to the customers
UR_63	HLUR_212 HLUR_213 HLUR_215	As a film production company I want to be able to put location of the assets, such as putting the asset in the exact place as intended. A 3D drag-and-drop would be required
UR_65	HLUR_214 HLUR_215	As a film production company I want to be able to choose each asset for a time span - This will help in IP protection and also help in development of updated assets
UR_69	HLUR_205 HLUR_207 HLUR_208	Batch Download/Load of assets. Have the ability to load multiple assets at once.

API connector - development language

C#

Development libraries used

Some external scripts may be used.

Local Storage

Local storage of preferences, cached assets, etc. will occur on the local file system.

Data objects stored in Local Storage

Data object name: Assets and scripts
Storage format: media and source files

Related technical requirements

TR NB	Description	Function	Function performed	Related URs
TR_VR_1	Retrieve 3D models from V4D Asset Repository		Query and retrieve 3D models of different resolutions from the V4Design repository	UR_002
TR_VR_2	Retrieve textures from V4D Asset Repository	Combined Query	send boundbox and retrieve the texture delimited by the box	UR_003
TR_VR_3	Retrieve asset metadata		The Query results have their metadata integrated, or made accessible (via URIs)	UR_006, UR_007, UR_039

TR_VR_4	Query V4D asset repository		Query resources by any field	UR_012, UR_013
TR_VR_5	Personal assets library, discover new assets		Query mechanism encourages discovery	UR_025
TR_VR_6	Save asset as different file format	Save assets	Save assets locally in different compatible formats	UR_014, UR_008
TR_VR_7	Manipulate V4D 3D models in VR	Manipulate assets	View and manipulate 3D models in an interactive way (rotate, zoom, etc.)	UR_010
TR_VR_8	Preview and markup of V4D assets	Asset viewer	Obtain low-resolution instances of images, videos and 3D models	UR_011, UR_009
TR_VR_9	Visualize retrieved V4D assets in VR		View any V4D asset (image, video, 3D model, etc.)	UR_020
TR_VR_10	Simple UI/UX		Intuitive and simple GUI design	UR_022
TR_VR_11	Provide open source code	Tool	Source code to be made available	UR_023

6 REQUIREMENTS OVERVIEW

Theme	TR NB	Description	Function	Function performed	Related URs
	TR_DS_1	Data Storing	Data push	Sending and storing of resource(s) to a target database.	UR_1, UR_59, UR_62
	TR_DS_2	Data retrieval	Data pull	Retrieval of resource(s) from a target database.	UR_2, UR_3, UR_4, UR_35, UR_37, UR_51, UR_69, UR_70
	TR_CR_1	Using a set of URLs as web entry points, collect all the hyperlinked URLs, up to a predefined depth.	Web crawling	Discovers nodes to scrape	UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56
	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		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	UR_10, UR_16, UR_21, UR_24, UR_55
	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	
Crawling and data storage	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	
Textual analysis	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	UR_10, UR_11, UR_12, UR_13, UR_14, UR_15, UR_16, UR_17, UR_18, UR_19,

	TR_LA_2		Concept extraction	Word Sense Disambiguation, Entity linking	UR_20, UR_21, UR_23, UR_35, UR_56, UR_57, UR_64
	TR_LA_3		Relation Extraction	Deep-syntactic parsing, Conceptual relation extraction	
	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	UR_21, UR_23, UR_57, UR_64
	TR_LG_2	Render the selected content as text.	Linguistic Generation	Generates text in target language	UR_10, UR_11, UR_12, UR_13, UR_14, UR_15, UR_16, UR_17, UR_18, UR_19, UR_20, UR_21, UR_23, UR_57, UR_64
Visual analysis	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	UR_3, UR_4, UR_41
	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	UR_3, UR_4, UR_25, UR_27, UR_41, UR_42
	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	UR_63
	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	UR_25, UR_63

				interest	
	TR_3D_1	Extract and build a 3D model	Reconstruct	Build a 3D model from the collection of images or video frames	UR-8, UR-2, UR-7, UR-20, UR-6, UR-16
Semantics analysis	TR_KB_1	Map analysis results from other modules	KBPopulation	RDF mapping and KB population	UR_2
	TR_KB_2	Provide an API over the KB for querying metadata			
	TR_KB_3	Map metadata about texture resolution			UR_3
	TR_KB_4	Map analysis results from building localisation			UR_10, UR_20, UR_57
	TR_KB_5	Map analysis results from object localisation			
	TR_KB_6	Map analysis results from aesthetics			
	TR_KB_7	Map analysis results from text analysis			
	TR_KB_8	Map analysis results from reasoning			
	TR_KB_9	Map metadata about quality			UR_12, UR_37
	TR_KB_10	Map geo-location of assets			UR_15
	TR_KB_11	Map date (creation date)			
	TR_KB_12	Map author info			UR_16
	TR_KB_13	Map copyright info			
	TR_KB_14	Map visible colours			UR_18
	TR_KB_15	Map metadata coming from 3D model reconstruction			UR_20
	TR_KB_16	Map results from text generation			UR_21
	TR_KB_17	Ability to associate assets with relevant external Web Pages			UR_22
	TR_KB_18	Map results from text generation			UR_23
	TR_KB_19	Associate assets with preview thumbnails			UR_30
	TR_KB_20	Ability to map texture mayerial metadata			UR_41
	TR_KB_21	Support the linking of assets with relevant ones			UR_50
	TR_KB_22	Support the annotation of assets with reuse rights			UR_55

		and copyrights			
	TR_KB_23	Map analysis results from text generation			UR_57
	TR_RQ_1	Support searching functionality (translation of user requests into one or more queries over the data storage	Reasoning Service	query formulation / enrichment	UR_2, UR_50
	TR_RQ_2	Infer geolocation from location tag		Inference of implicit relations and context	UR_15
	TR_RQ_3	Propagate annotations from other modalities to the 3D models		Inference of implicit relations and context	UR_20
	TR_RQ_4	Find relevant external Web page, based on the annotation provided by other components		Inference of implicit relations and context	UR_22
	TR_RQ_5	couple searching functionality with text analysis on the keywords		query formulation / enrichment	UR_35
	TR_RQ_6	Find assets relevant to other assets		Inference of implicit relations and context	UR_50
Middleware	TR_RA_1	Create a user profile	Create user	Helps user tools to create users	UR_022
	TR_RA_2	Create login credentials for a user	User Login	Helps user tools to login to the system and authenticate them	
	TR_RA_3	Create new asset	Create Asset	Helps user tools to create asset details	UR_1, UR_59, UR_62, UR_014, UR_008
	TR_RA_4	Upload new asset	Upload Asset	Helps users to upload assets	
	TR_RA_5	Get latest asset (texture, 3D model) from DB	Get Latest assets	Gets the latest assets from the database	UR_2, UR_3, UR_4, UR_35, UR_37, UR_51, UR_69, UR_70
	TR_RA_6	Search V4Design DB for assets	Search	Searches the database for assets based on a specific field	UR_2, UR_3, UR_4, UR_35, UR_37, UR_51, UR_69, UR_70
	TR_RA_7	Rate an asset	Rating	Rates an asset	UR_014, UR_008
	TR_RA_8	Comment an asset	Comments	Adds a comment to the asset	
Authoring tools	TR_RN_1	Retrieve 3D models from V4D Asset Repository	Combined Query	Query and retrieve 3D models of different resolutions from the V4Design repository	UR_002
	TR_RN_2	Retrieve textures from V4D Asset Repository		send boundbox and retrieve the texture delimited by the box	UR_003

	TR_RN_3	Retrieve asset metadata		The Query results have their metadata integrated, or made accessible (via URIs)	UR_006, UR_007, UR_039
	TR_RN_4	Query V4D asset repository		Query resources by any field	UR_012, UR_013
	TR_RN_5	Personal assets library, discover new assets		Query mechanism encourages discovery	UR_025
	TR_RN_6	Save asset as different file format	Save assets	Save assets locally in different compatible formats	UR_014, UR_008
	TR_RN_7	View V4D 3D models in VR	Asset viewer	View 3D models in an interactive way (rotate, zoom, etc.)	UR_010
	TR_RN_8	Preview and markup of V4D assets		Obtain low-resolution instances of images, videos and 3D models	UR_011, UR_009
	TR_RN_9	Visualize retrieved V4D assets		View any V4D asset (image, video, 3D model, etc.)	UR_020
	TR_RN_10	Simple UI		Intuitive and simple GUI design	UR_022
	TR_RN_11	Open source	Tool	Source code to be made available	UR_023
	TR_VR_1	Retrieve 3D models from V4D Asset Repository	Combined Query	Query and retrieve 3D models of different resolutions from the V4Design repository	UR_002
	TR_VR_2	Retrieve textures from V4D Asset Repository		send boundbox and retrieve the texture delimited by the box	UR_003
	TR_VR_3	Retrieve asset metadata		The Query results have their metadata integrated, or made accessible (via URIs)	UR_006, UR_007, UR_039
	TR_VR_4	Query V4D asset repository		Query resources by any field	UR_012, UR_013
	TR_VR_5	Personal assets library, discover new assets		Query mechanism encourages discovery	UR_025
	TR_VR_6	Save asset as different file format	Save assets	Save assets locally in different compatible formats	UR_014, UR_008
	TR_VR_7	Manipulate V4D 3D models in VR	Manipulate assets	View and manipulate 3D models in an interactive way (rotate, zoom, etc.)	UR_010
	TR_VR_8	Preview and markup of V4D assets	Asset viewer	Obtain low-resolution instances of images, videos and 3D models	UR_011, UR_009
	TR_VR_9	Visualize retrieved V4D		View any V4D asset	UR_020

		assets in VR		(image, video, 3D model, etc.)	
	TR_VR_10	Simple UI/UX		Intuitive and simple GUI design	UR_022
	TR_VR_11	Provide open source code	Tool	Source code to be made available	UR_023

7 REQUIREMENTS ANALYSIS AND APPLICATION

In the previous sections, we have introduced and discussed the technical specifications and requirements of all the V4Design platform components, in addition to the specifications of its architecture and integration model. Although some of these specifications may change or evolve throughout the project as new requirements become clear or as adjustments are made to original approaches, the bulk of these specifications us solid enough to perform a first-hand analysis of the platform's technical requirements.

In this section, we conduct an analysis of the specifications and requirements that aims to consolidate certain functional and non-functional aspects related to the architecture design specifications, and to provide guidelines for using these specifications in the implementation and evaluation of the platform. The objectives of this analysis is to congregate, aggregate, and compare related or connected technical requirements and specifications in order to insure overall compatibility, between the modules as integrated elements of a single architecture, and between these modules and the platform-level processes. On the basis of this analysis, we draw recommendations for the developers of the V4Design architecture modules.

In order to perform such as analysis, we define the following three axes of concern upon which the collected technical requirements and specifications will be analysed:

- ***Data management***, including data exchange mechanisms between different modules
- ***Messaging***, including the definition of message topics
- ***Platform cycle***, including the chaining of components to implement the platform processing pipeline.

7.1 Data management specifications

In section 3 we described the specifications of the data management in the V4Design platform, arguing about the relation between data objects and platform components, introducing the Data Storage and Retrieval component as the platform's centralized repository, and explaining the necessary mechanisms for storing and retrieving data in this storage. In complementation, we have defined the data objects used as input and those generated by each module in the platform in section 4. In the following, we discuss a unified data schema that specifies how data is formatted throughout the system. This unified data schema is applied to all data exchanged in the system through the Data Storage and Retrieval. Other formats can be used locally (inside the services' local storages) or by the user tools (stored separately outside the platform, on the tool's own server).

Given that the Crawler and other modules in charge of ingesting data into the system use the SIMMO data model [10], and that the Data Storage and Retrieval is implemented in a manner compatible with this format, we recognize that the system's raw data will be in the SIMMO format, including all data imported by the user.

In figure 19, we show an abstraction of the SIMMO format, showing how each data object contains an instance of a Source object where metadata about its origin, use, and acquisition is stored, and one or more Text object and Media object. The Media object may be either an image or a video. Media objects may contain thumbnail images and other metadata. Similarly, the Text object may contain semantically related metadata.

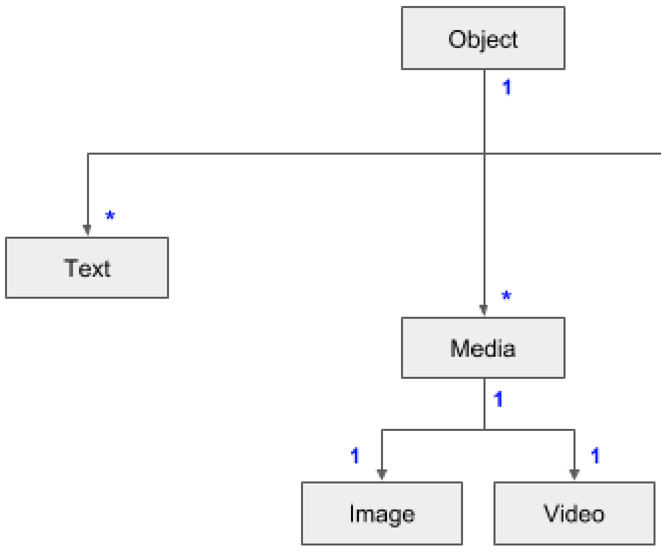


Figure 19: an abstraction of the SIMMO data model

Therefore, and since each service adds another component or metadata to the data object it uses as input, we construct the V4Design data object schema on the basis of SIMMO, augmenting the original model with instances of objects, each the result of a given service. These are integrated in the model in a manner that takes into account their interrelation with the SIMMO object's elements. This V4Design data object schema is presented in the following figure.

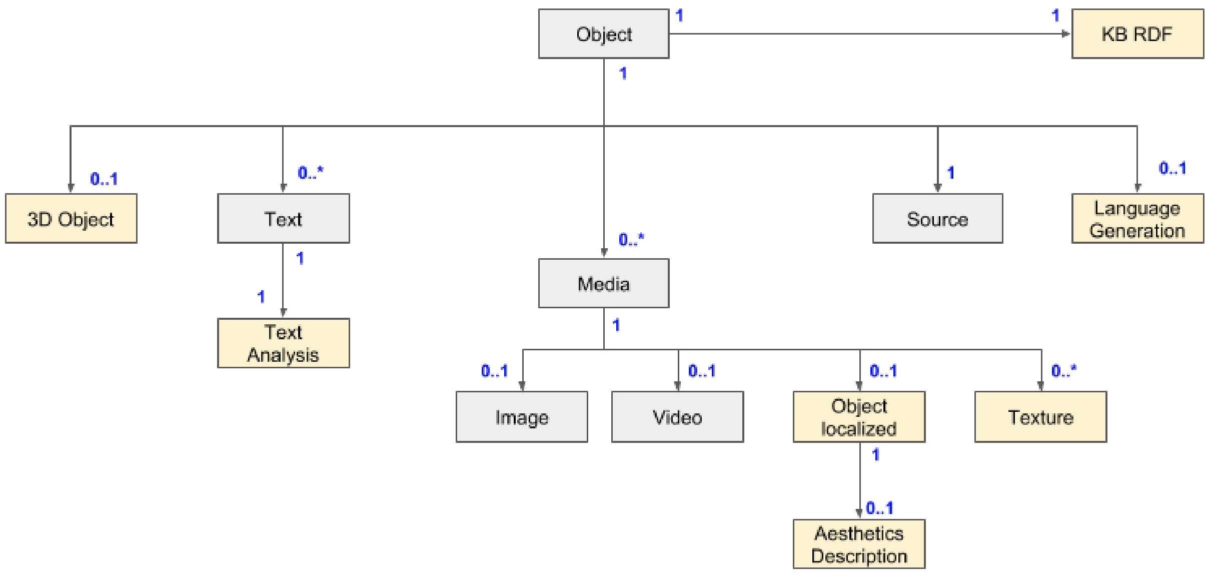


Figure 20: The V4Design data object schema

Therefore, each platform component should send its output to the Data Storage and Retrieval following this schema. Modifications to this schema could be contemplated on the basis of necessities and changes that services can incur during their integration and adaptation to the platform's processing cycle. For instance, a service may output a series of objects instead of a single result self-contained in a single object; or a service may associate a single modular output to a series of input objects. In all cases, the proposed data object schema is a good initial solution since it accommodates all services according to their current specifications.

7.2 Messaging

In section 4 we have identified the messages that each component sends and expects to receive. In addition, we have discussed the specifications of the message bus in section 4.2. Furthermore, details about the message bus were presented in the deliverable D6.1, including important technical specifications such as no data is channelled through the message bus, and that data is referenced by URIs.

According to the model chosen for the platform, the **messages are decentralized**: i.e. each component is responsible for informing other components of its actions. If the messages are about data (instead of status or other concerns), and if they are associated with newly generated output from the component, then the data must be pushed to the Data Storage and Retrieval before emitting the message. This way, this data is already available before it is requested by other components.

In addition, since the data storage will respond to any push request with a response indicating the URIs of the newly saved data, there URIs should be included in the subsequent message emitted by the component. This implies that the data storage will NOT emit messages regarding the status of the data. The messages emitted by the data storage are mainly responses to data queries (in addition to technical messages, such as status and authentication).

Consequently, messages about new data are emitted by the crawlers & wrappers, and messages about processing data are emitted by the services.

Therefore, an effective chaining of the platform services will allow the services to work together without the need to query for new data; they can directly retrieve the data objects using their URIs from the messages that they receive. Only services that need to make a decision on whether their process data or not (e.g. depending on its availability or completeness), may have to make data queries.

In the following table, we summarize the message topics defined on the basis of the information and specifications collected and discussed in section 4. It shows that the platform requires no less than 14 topics, some are broadcasts such as "data_available", others are directed to a single receiver such as in the case of queries and responses, and others are shared among more than two components.

Topic ID	Component	Input	Output	Data stored in storage
DATA_AVAILABLE	Crawler	URL or query	SIMMOs	Raw SIMMOs

DATA_AVAILABLE	DataWrapper (EDM)	API parameters	SIMMOs	Raw SIMMOs
QUERY_RESPONSE	Data Storage and retrieval	Query raw data	an array of SIMMOs	N/A
QUERY_RESPONSE	Data Storage and retrieval	Query processed data	an array of assets	N/A
OBJECT_LOCALIZED	Object localization	Single SIMMO	one JSON per media describing where the mask image is stored, including Type and Tags	Store JSON by original SIMMO ID & Media ID
TEXT_ANALYZED	Language Analysis	Single SIMMO	one JSON per Simmo	Store JSON by original SIMMO ID & (Text ID or Media ID)
OBJECT_RECONSTRUCTED	3D Reconstruction	SIMMO Collection?	3D model + JSON (list of possible reconstructions)	Store generated 3D model + metadata
LANGUAGE_GENERATED	Language generation	Single SIMMO with ontological info from Knowledge Base + TA JSON	one JSON per Simmo	Store JSON by original SIMMO ID & (Text ID or Media ID)
TEXTURE_EXTRACTED	Texture Proposals	Single SIMMO + user input (zone)	JPG + JSON per SIMMO	Store JSON by SIMMO ID & Media ID
AESTHETICS_EXTRACTED	Aethstetic Extraction	Single SIMMO	one JSON per Simmo + JSON of OL	Store JSON by SIMMO ID & Media ID
KB_POPULATION_FINISHED	KB Population	Single SIMMO + the JSONs	No output	N/A
REASONING_FINISHED	Reasoning	KB updated	No output	N/A
SEARCH_FINISHED	Reasoning (searching)	REST API filters	Collections of assets matching the incoming query	N/A
QUERY	REST API	Query parameters	an array of assets	N/A
TEXTURE_REQUESTED	REST API	Texture parameters	Texture asset	N/A

Table 10: Message topics for V4Design platform components

In the following figure, we show how these messages connect the different platform components, allowing each to perform or to execute in turn, after the relevant data becomes available. This tight communication between the platform components is the main aspect of the platform integration.

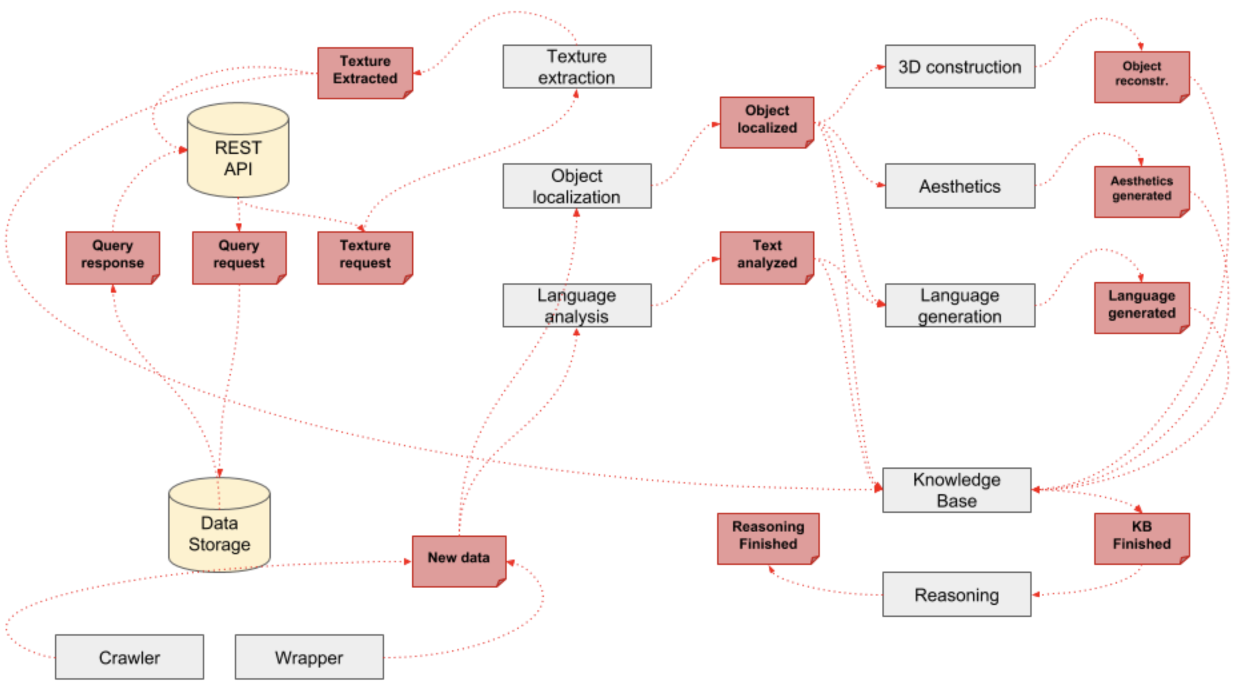


Figure 21: The V4Design data object schema

7.3 The platform cycle

The introduction of the message topics and their use to connect the different services allow to complete the processing pipeline of the platform, which refers to the chaining and the ordering of the execution of different services. Some services are designed to work with raw data as ingested by the crawlers, but some services require output from other services, and therefore they rely on a correct design of the processing pipeline to perform effectively within the architecture.

In the following figure we show how the processing pipeline and cycle are organized. The numbers reveal the chronological order by which the messages are emitted. It starts with new data, and ends with extraction of texture upon a user request. This pipeline design is expected to evolve and change throughout the project.

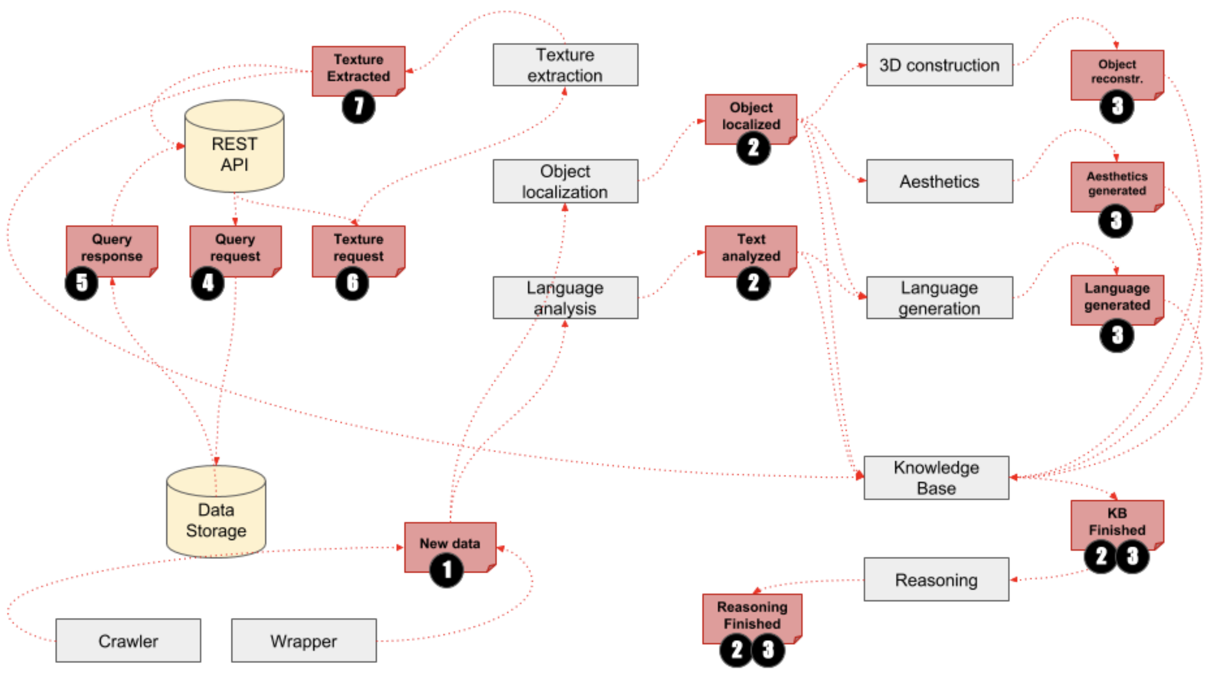


Figure 22: The processing pipeline and its cycle

During this cycle, the platform builds upon original data objects, adding new components to each one of them as it goes through the process. In the following figure, we visualize this process by applying the cycle onto the V4Design data object schema.

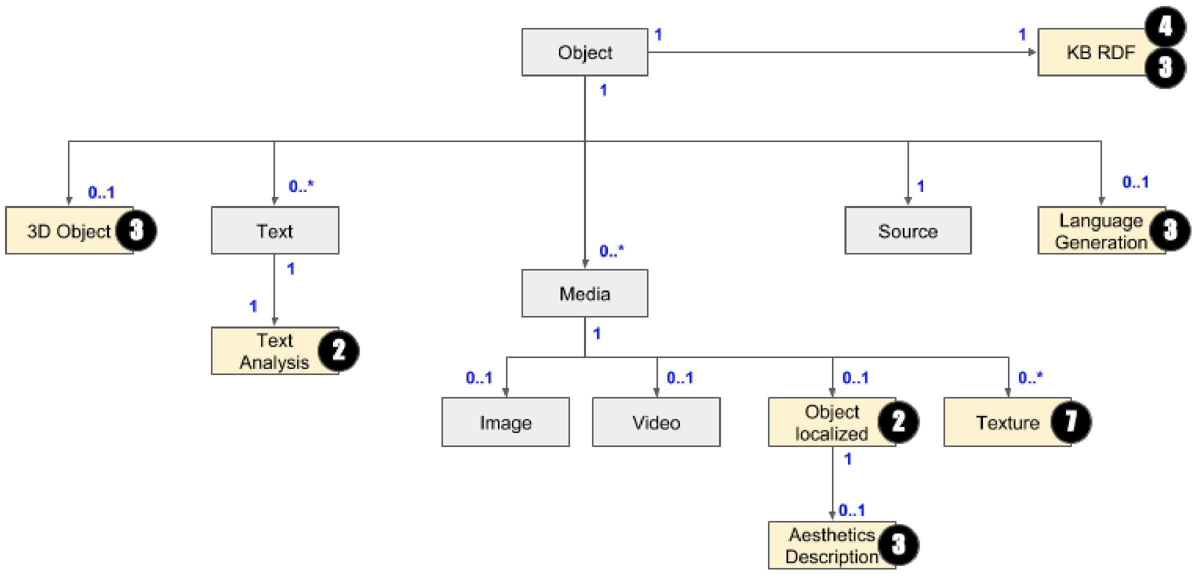


Figure 23: The platform cycle applied to the V4Design data object schema

Finally, we show how the data is generated during this cycle in the following figure that visualizes how each component reads and writes data during the cycle.

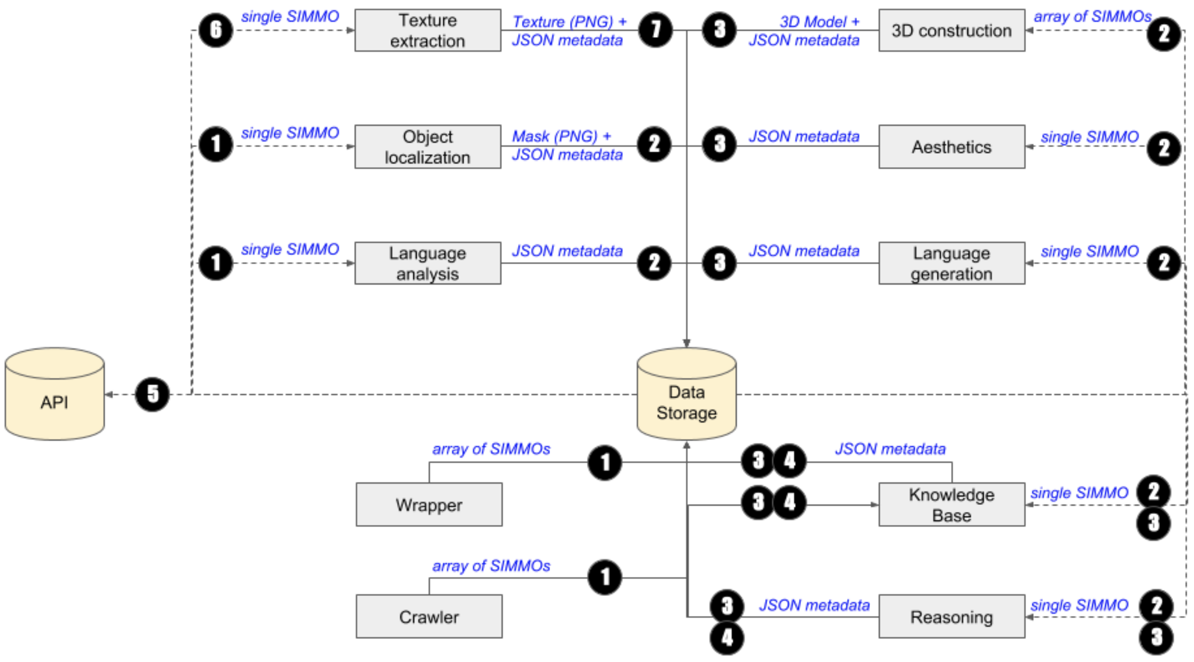


Figure 24: Data read/write operations during the platform cycle

8 CONCLUSIONS

In this deliverable we have presented and discussed the technical specifications of the V4Design platform based on early analysis that precedes the implementation of its first operational prototype. After a short introduction on the general practices for requirements collection and analysis, and the delimitation of the scope of such activity in the context of V4Design, we discussed the technical specifications, first on a platform level, and then on a module level, addressing services, middleware, and user tools. Finally, we conducted an aggregative analysis of requirements by which common concerns related to the architecture design specifications (data management, messaging, and platform processing pipeline) were revisited, and extended by relying on the elementary specifications collected.

In order to meet the user requirements defined under the usage scenarios and use cases, the implementation the platform will follow an iterative approach by which at least three distinct versions will be implemented and evaluated, one after the other. Therefore, the particular set of user requirements that are addressed under a specific version should be isolated and clearly stated in order to orient the user evaluations. For instance, it is expected that the first “proof-of-concept” version of the platform, otherwise known as V1, will incorporate rudimentary and basic functions. The following version V2 will address use cases 1 and 2 more in depth, and demonstrate the viability and added-value of the processes described by these use cases.

Therefore, we expect the specifications to evolve in accordance with the evolution of the platform’s implementation, and the results of consequent experimentation with integrated prototypes. In fact, several issues have been raised during this analysis that cannot be solved without experimenting with early prototypes. For instance, the 3D Reconstruction service needs to identify and isolate collections of media objects containing the same object to reconstruct. This decision-making paradigm is currently under research with several possible solutions considered.

Finally, this analysis was an effective coordination and knowledge sharing and building exercise by which all partners have converged onto a common understanding of the specifications, functionalities and architecture of the intended platform, which allows and facilitate ad-hoc development efforts. Indeed, the standardization of major concerns, such as data management, data schemas, messaging, integration models, roles, and interdependencies was a necessary step to insure a more streamlined development.

9 REFERENCES

[1] Eeles, P. (2005). Capturing architectural requirements. IBM Rational developer works.

[2] Liao, L. (2002). From Requirements to Architecture: The State of the Art in Software Architecture Design. Department of Computer Science and Engineering, University of Washington, 1-13.

[3] Lehman, Meir M. The programming process. internal IBM report, 1969.

[4] Lehman, M.M.; Belady, L.A. (1985). Program evolution : processes of software change. London: Academic Press Inc. ISBN 0-12-442441-4.

[5] Lehman N, Meir M., et al. Metrics and laws of software evolution-the nineties view. En Software metrics symposium, 1997. proceedings., fourth international. IEEE, 1997. p. 20-32.

[6] Grady, Robert; Caswell, Deborah (1987). Software Metrics: Establishing a Company-wide Program. Prentice Hall. p. 159. ISBN 0-13-821844-7.

[7] Clegg, Dai; Barker, Richard (2004-11-09). Case Method Fast-Track: A RAD Approach. Addison-Wesley. ISBN 978-0-201-62432-8.

[8] Clements, P., Kazman, R., & Klein, M. (2003). Evaluating software architectures. Beijing: Tsinghua University Press.

[9] Kulak, Daryl and Eamonn Guiney. “Use Cases: Requirements in Context”: pages 19, 20. Addison-Wesley, 2012

[10] T. Tsikrika, K. Andreadou, A. Moumtzidou, E. Schinas, S. Papadopoulos, S. Vrochidis, Y. Kompatsiaris, "A Unified Model for Socially Interconnected Multimedia-Enriched Objects", 21st MultiMedia Modelling Conference (MMM2015), Sydney, Australia, 5-7 January, 2015

A Appendix A: V4Design Service Definition Template

A “service” is defined as a standalone component of the platform architecture. It communicates with other services as a single entity or point. Internally, a service may integrate different components, each with a specific role of function, but externally the service acts as an integrated application. A service can be hosted on its own independent server, and is managed by a service owner that is responsible for the health of the service.

Please use the following template to define formally each service that would be integrated in the V4Design platform. The information required and defined in this template centre on the aspects that govern the relationship of this service with other services and middleware components of the integrated V4Design service platform. Please fill it to the extent possible.

DEFINITION

Official **name** of the service:

Service **owner**:

Short **description**:

STATUS

Current instance **status**:

(Deployed / Alpha / In-development / Concept)

If NOT currently deployed

Expected date for delivering a **stable version**:

If currently deployed:

Current version:

Expected date of next release:

FUNCTIONALITIES

Describe the **main functions** of the service that will be made available for the platform:

(Requests supported by the service)

Name	Description	Data input *(from other services)*	Data output *(to other services)*
		E.g. 3D Models ← non-semantic data service	

For each of these functions, specify the following:

Name	Request type / topic *(message received)*	Response type /topic *(message emitted, if any)*	Expected response time	Capacity *(nb requests handled)*

According to current plans for integration, service-to-service direct communication is to be eliminated, and all communication routed through the message bus. Data will not be encapsulated or contained within messages, unless it implements the resource and area segments of the CAP protocol used to encode the message's content.

In case you are planning to support direct communication between your service and other services, shortcutting the message bus, which are these other services? what **communication protocol** would be used?

(Please identify any communication managed outside the realm of the platform's message bus)

SERVICE REQUIREMENTS

For documentation purposes, please describe briefly the **service requirements** as a standalone application:

- Deployment environment and architecture model
- Expected capacity in processing requests
- Expected availability and reliability of the service
- Data integrity policy (*if relevant*)
- Interoperability requirements (*if relevant*)

TECHNICAL CONCERNS

Does the service **store data** locally?

(Identify the relevant data objects stored and describe the type of storage.)

In this case, does the service provide data for other services, and under which protocol?

Describe how the service handles **security**

(Identify the security protocol that the service implements locally)

Describe the service **scalability** model:

(Specify if the service scales vertically or horizontally, or does not scale)

MESSAGING CONCERNS

Events under which the service sends notification or broadcasting messages through the bus

Event name	Description	Message type / topic	Message receiver(s) (other services)	Data (if any)

Events under which the service receives requests and notifications from other services

(If yes, please elaborate)

Event name	Description	Sender(s) (other services)	Message or notification type / topic	Response type / topic (if any)

Please make sure that the Request messages supported by the service have been defined in the "functionalities" section, in relation with the service's supported functionalities.

Does the service need message scheduling?

(If yes, please elaborate)

Does the service implement its own message queue?

(If yes, please describe the queue concisely)

B Appendix B: Requirements Definition of V4Design Service

Service Name:

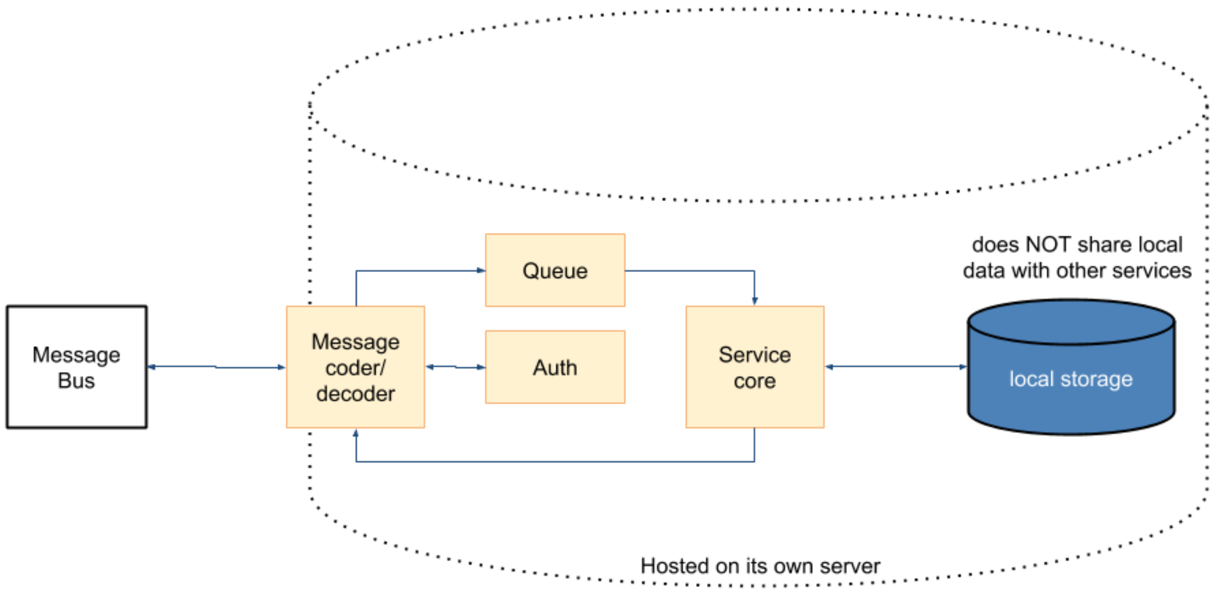
Service Owner:

Service Description:

In the following diagram, we introduce the conceptual design of a V4Design service.

Accordingly, each service is composed of the following main components:

- ***The message coder / decoder***: codes messages in AMQP protocol, and the message content according to the CAP protocol.
- ***The service queue***: queues requests or messages sent to the service in order to manage the service pipeline. Services that can processes messages in parallel may not need a queue
- ***The authentication mechanism***: stores authentication information and/or manages authentication requests where relevant (NOT considered relevant for V1 of the integrated platform).
- ***The Service core***: the service core comprises of the actual service functionalities and mechanisms.



1 - The hosting Server:

Please provide the technical requirements of the server you are planning to host the service on.

S.01. Please describe the server's minimum hardware requirements

Operating System:

CPU:

RAM:

Disk Space:

Please list the software requirements of the server you are planning to host the service on.

S.02. Please describe the server's software requirements (basic software solutions that need to be installed, e.g. Apache Tomcat, MySQL, Python, Java, etc..). Provide the version required in each case.

2 - The service Core:

C.01. Please define the functional/technical components that make us the service. Each component can represent a unitary function. Chained together, the components form the processes/pipelines implemented by the service.

Component name	**Data input**	**Data output**	**Function(s) performed**

C.02. Please provide a simple diagram that illustrate the conceptual/logical design of the service core, showing how the components are chained together.

we suggest to use https://www.draw.io/

C.03. Please define the global functions of the Service

Each global function may involve one or more components listed in C.01.

Function	Description	Data input	Data output	Components

C.04. Please link the service to the project's user requirements

Specify which of the user requirements involve this service, The Requirements are listed here:

https://docs.google.com/document/d/1QQ0qWRz5f0UasqxlM8ZuZTKzGuzZArvaYyJU2nEqSU8/edit

Trigger refers to how the service function is triggered, e.g. request received, service status changed, etc.

Requirement S. No.	Service Function	Trigger	Required data object(s)

C.05. Please define the data objects generated by the Service

For each data object, please copy-paste and fill the following:

Data object name:

Service function (see C.03):

Data schema:

For each field, please specify:

- Field name:
- Type:
- Allowed values:

3 - The message coder / decoder

Please check:

AMQP specifications: https://www.amqp.org/resources/download

CAP specifications: http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html

M.01. In which language will you write your encoder/decoder?

M.02. Are you using any external library? Which one (provide a link if applicable)?

Please check available libraries and solutions online, for instance:

Ajax: http://activemq.apache.org/ajax.html

Python: https://github.com/guardicore/haigha2

Others: http://activemq.apache.org/connectivity.html

M.03. Please define the messages that will be sent by the service

Take into account the following types of messages:

Broadcast: a service broadcasts a message to inform other services of changes in its status-quo, including the arrival of new data, the completion of a process, and so on.

Service-to--Service: a service places a request to another service, which could include a query for data, a trigger for a function, and so on.

Message name	**Function** (*see C.03*)	**Receiver(s)**	**Related data objects**

M.04. Please identify the messages that will be received by the service

Message name	**Function** (*see C.03*)	**Sender**	**Related data objects**

4 - The Queue

Q.01. Are you planning on integrating a queuing solution? At which project month?

Q.01N. If NOT, what is the expected processing capacity of the service (request per second or requests per minute?)

Q.01Y. If YES, are you using any external library? Which one (provide a link if applicable)?

5 - The Local Storage

If you are planning to store data locally (in a database or file system, please provide details on how this is does), please specify the following specifications:

T.01. What data storage solution will you adopt? (database, file system, etc..).

T.02. If you are using existing data storage solutions (e.g. MySQL), please specify:

Name: URL: Version: Dependencies, if any:

T.03. What Is the estimated disk space required for the data storage?

T.04. Define the data objects stored in the Local Storage, and their schema (fields, their types and expected values)

If you data schema is large, or you prefer using a UML-like notation, we suggest to use https://www.draw.io/

For each type of data objects, please copy-paste and fill the following: Data object name: Storage format: Data schema: For each field, please specify: - Field name: - Type:

- Allowed values:

C Appendix C: Specs and mechanisms for data storage and retrieval

Output data storage

Please describe where the output of your modules is going to be stored. The output could be either temporary, i.e. the output is needed only by some other module, so there is no need be persisted for longer, or it can be directly queried and showed in the interface, e.g. a 3D model.

Example storage types: file system (e.g. simple folders), database (e.g., MySQL), RDF triple store, other.

Module	Storage type	Details / comments
e.g. Building localisation	file system (for the images)	-
	database (for the metadata file)	MySQL

Access to the data

Please describe the way the other modules will be able to get / query for the data. More specifically, describe abstractly the content of the message that will be sent to the message bus that will designate the way the output data will be retrieved. You should have in mind that since we are talking about a distributed architecture, the data should be able to be retrieved remotely.

Module	Message content
e.g. Building localisation	the ftp path from where the images can be retrieved, along with an SQL query for the relevant metadata from the database (a query is needed only of the underlying data schema is complex, e.g. how to join tables etc.)
	other possibility: the component may expose a web API, e.g. a rest API, such as http://…#get-data?id=3, to serve as a web interface that implements the retrieval mechanism

D Appendix D: Requirements Definition of A V4Design Tool

Tool Name:

Tool Owner:

Tool Description:

1 - User Profiles

Please describe each technical profile supported by the tool (e.g. basic, advanced, admin, etc.)

S.01. For each supported profile, please describe the following

Profile name:

Role description:

Permissions:

Restrictions:

S.02. Describe the user authentication mechanism followed

Specify is users can utilize 3rd-party authentication, indicate which

2 - Usage environment

Please describe details on how the user can use this tool

S.03. What is the installation / access model? (desktop app, cloud service, etc.)

S.04. What prerequisites are required? (browser version, dependency tools, etc.)

S.05. What are the technical specifications of the usage environment?

Operating System:

CPU:

RAM:

Disk Space:

Others:

3 - Deployment environment (Do not fill for desktop tools)

Please describe details on the environment in which the tool is deployed

S.06. What software prerequisites are required?

S.07. What are the technical specifications of the deployment environment?

Operating System:

CPU:

RAM:

Disk Space:

Others:

S.08. How many simultaneously connected users can the tool support?

4 - The Tool Logical Design:

S.09. Please define the functional/technical components that make us the tool. Each component can represent a unitary function. Chained together, the components form the processes/pipelines implemented by the tool.

Component name	Data input	Data output	Function(s) performed

S.10. Please provide a simple diagram that illustrate the conceptual/logical design of the tool, showing how the components are chained together.

we suggest to use https://www.draw.io/

5 - Functionalities:

S.11. Please define the main functionalities of the tool

Each global function may involve one or more components listed in C.01.

Functionality	Description	Data Objects

S.12. Please link the tool to the project's user requirements

Specify which of the user requirements involve this tool, The Requirements are listed here:
https://docs.google.com/document/d/1QQ0qWRz5f0UasqxlM8ZuZTKzGuzZArvaYyJU2nEqSU8/edit

Requirement Serial. No.	Functionality

6 - The message coder / decoder

Please check AMQP specifications: https://www.amqp.org/resources/download

S.13. In which language will you write your encoder/decoder?

S.14. Are you using any external library? Which one (provide a link if applicable)?

Please check available libraries and solutions online, for instance:

Ajax: http://activemq.apache.org/ajax.html

Python: https://github.com/guardicore/haigha2

Others: http://activemq.apache.org/connectivity.html

S.15. Please define the messages that will be sent by the tool

Take into account the following types of messages:

Broadcast: a service broadcasts a message to inform other services of changes in its status-quo, including the arrival of new data, the completion of a process, and so on.

Service-to--Service: a service places a request to another service, which could include a query for data, a trigger for a function, and so on.

Message name	Functionality	Receiver(s)	Related data objects

7 - The Local Storage

If you are planning to store data locally (in a database or file system, please provide details on how this is done), please specify the following specifications:

S.16. What data storage solution will you adopt? (database, file system, etc..).

S.17. If you are using existing data storage solutions (e.g. MySQL), please specify:

Name:
URL:
Version:
Dependencies, if any:

S.18. What Is the estimated disk space required for the data storage?

S.19. Define the data objects stored in the Local Storage, and their schema (fields, their types and expected values)

If you data schema is large, or you prefer using a UML-like notation, we suggest to use https://www.draw.io/

For each type of data objects, please copy-paste and fill the following:

Data object name:

Storage format:

Data schema:

For each field, please specify:

- Field name:
- Type:
- Allowed values:

E Appendix E: REST API Specifications

2018-10-18

Contents

General

Requests

- GET = no body or JSON body
- POST = JSON body or binary file data
- Some requests require a valid session to perform their action. If so the session id must be sent in the request header as ‘SessionId’.

Response

- Response body as JSON
- On success send a request related response which is defined in each action
- On error send the response

  ```
  {
          “Result”: “Error”,
          “ErrorMessage”: “String”
  }
  ```

  Where “ErrorMessage” is a clue to the occurred error.
- Some errors may not produce a valid JSON response body, in this case the body must be ignored.

Request Scheme

[Server URL]/[Component]/[Action]/[Resource]

HTTP response codes

- 200 on processed action
- 400 on bad request
- 403 on missing or invalid session id
- 500 on server-side error
- 503 on server maintenance mode

Security

- HTTPS should be used for all requests
- Ticket System?
- How to send passwords?

Answered Questions

- Do we need a user/login system to upload/update assets?
  - Yes
- Do we need a review system to add comments and rate assets?
  - Yes
- What about the epoch models? Are these different assets or does each asset has two models for two epochs? For example, do we have a model version for year 1950 and for 2015 or are these two assets?
  - Different assets

API

User

Create

Creates a new user.

- POST
- [Server URL]/User/Create
- Valid Session Id required
- Request Body
```
{
        "UserName": "String",
        "Password": "String"
}
```
- Response Body
```
{
        "Result": "Success"
}
```

Login

Logs the user in and creates a session.

- POST
- [Server URL]/User/Login
- Request Body
```
{
        "UserName": "String",
        "Password": "String"
}
```
- Response Body
```
{
        "Result": "Success",
        "SessionId": "String"
}
```

Assets

Create

Creates a new asset in the database.

- POST
- [Server URL]/Assets/Create
- Valid Session Id required
- Request Body
```
{
        "Name": "String",
        "Description": "String",
        "ReferenceDate": Timestamp,
        "Longitude": int,
        "Latitude": int,
        “Polygons”: int,
        “EraYear”: int,
        “Tags”:
        [
                “String”,
                …
        ]
}
```

- Response Body

```
    {
            “Result”: “Success”,
            “AssetId”: “String”
    }
```

Upload

Uploads the model file to the server.

- POST
- [Server URL]/Assets/Upload/AssetId
- Valid Session Id required
- Request Body
  Binary File Data in fbx-Format
- Response Body

```
    {
            “Result”: “Success”
    }
```

Update

Updates the information of an asset in the database.

- POST
- [Server URL]/Assets/Update/AssetId
- Valid Session Id required
- Request Body

```
    {
            “UpdateMessage”: “String”,
            “Name”: “String”,
            “Description”: “String”,
            “ReferenceDate”: Timestamp,
            “Longitude”: int,
            “Latitude”: int,
            “Polygons”: int,
            “EraYear”: int,
            “Tags”:
            [
                    “String”,
                    …
            ]
    }
```

- Response Body
```
{
        “Result”: “Success”
}
```
Latest

Gets a list of the latest uploaded assets.

- GET
- [Server URL]/Assets/Latest
- Request Body
```
{
        “Page”: int,
        “Count”: int
}
```
- Response Body
```
{
        “TotalAmount”: int,
        “Assets”:
        [
                {
                        “AssetId”: “String”,
                        “Name”: “String”,
                        “DownloadUrl”: “String”,
                        “ThumbnailUrl”: “String”,
                        “Description”: “String”,
                        “ReferenceDate”: Timestamp,
                        “Longitude”: int,
                        “Latitude”: int,
                        “Polygons”: int,
                         “EraYear”: int,

                        “Tags”:
                        [
                                “String”,
                                …
                        ]
                },
                …
        ]
}
```
Get

Gets a specific asset.

- GET
- [Server URL]/Assets/Get/AssetId
- Response Body
```
{
        "AssetId": "String",
        "Name": "String",
        "DownloadUrl": "String",
        "ThumbnailUrl": "String",
        "Description": "String",
        "ReferenceDate": Timestamp,
        "Longitude": int,
        "Latitude": int,
        "Polygons": int,
        "EraYear": int,
        "Tags":
        [
                "String",
                …
        ]
}
```

GetHistory

Gets the change history for an asset.

- GET
- [Server URL]/Assets/GetHistory/AssetId
- Response Body

```
[
        {
                "UserId": "String",
                "Date": Timestamp,
                "UpdateMessage": "String"
        },
        …
]
```

SearchByTags

Gets a list of assets with the matching tags.

- GET
- [Server URL]/Assets/SearchByTags
- Request Body

```
{
        "Tags":
        [
                "String",
                …
        ],
        "Page": int,
        "Count": int
}
```

- Response Body

```
{
        “TotalAmount”: int,
        “SearchResults”:
        [
                {
                        “AssetId”: “String”,
                        “Name”: “String”,
                        “DownloadUrl”: “String”,
                        “ThumbnailUrl”: “String”,
                        “Description”: “String”,
                        “ReferenceDate”: Timestamp,
                        “Longitude”: int,
                        “Latitude”: int,
                        “Polygons”: int,
                “EraYear”: int,
                        “Tags”:
                        [
                                “String”,
                                …
                        ]
                },
                …
        ]
}
```

SearchByAddress

Gets a list of assets with a nearby location to a given address (e. g. a City).

- GET
- [Server URL]/Assets/SearchByAddress
- Request Body

```
{
        “Address”: “String”,
        “Page”: int,
        “Count”: int
}
```

- Response Body

```
{
        “TotalAmount”: int,
        “SearchResults”:
        [
                {
                        “AssetId”: “String”,
                        “Name”: “String”,
                        “DownloadUrl”: “String”,
                        “ThumbnailUrl”: “String”,
                        “Description”: “String”,
                        “ReferenceDate”: Timestamp,
                        “Longitude”: int,
                        “Latitude”: int,
                        “Polygons”: int,
                         “EraYear”: int,
                        “Tags”:
                        [
                                “String”,
                                …
                        ]
                },
                …
        ]
}
```

SearchByReferenceDate

Gets a list of assets with the matching reference dates.

- GET
- [Server URL]/Assets/SearchByReferenceDate
- Request Body

```
{
        “ReferenceDate”: “String”,
        “Page”: int,
        “Count”: int
}
```

- Response Body

```
{
        “TotalAmount”: int,
        “SearchResults”:
        [
                {
                        “AssetId”: “String”,
                        “Name”: “String”,
                        “DownloadUrl”: “String”,
                        “ThumbnailUrl”: “String”,
                        “Description”: “String”,
                        “ReferenceDate”: Timestamp,
                        “Longitude”: int,
                        “Latitude”: int,
                        “Polygons”: int,
                         “EraYear”: int,
                        “Tags”:
                        [
                                “String”,
                                …
                        ]
                },
                …
        ]
```

}

Rating

Rate

Adds a rating to an asset.

- POST
- [Server URL]/Rating/Rate/AssetId
- Valid Session Id required
- Request Body
```
{
        “Rating”: int,
        “Comment”: “String”
}
```
- Response Body
```
{
        “Result”: “Success”
}
```

Get

Gets the ratings for an asset.

- GET
- [Server URL]/Rating/Get/AssetId
- Request Body
```
{
        “Page”: int,
        “Count”: int
}
```
- Response Body
```
{
        “Average”: float,
        “TotalAmount”: int,
        “Ratings”:
        [
                {
                        “Rating”: int,
                        “Comment”: “String”
                },
                …
]
}
```