V4Design

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

H2020-779962

D6.1

Roadmap towards the implementation of the V4Design platform

Dissemination level:	Public
Contractual date of delivery:	Month 5, 31 May 2018
Actual date of delivery:	Month 5, 31 May 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 Draft
Version:	1.0
Number of pages:	63
Filename:	D6.1_V4DesignRoadmap_20180531_v1.0.pdf

Abstract

D6.1 provides the time-plan for the development of the V4Design platform. It will specify the functionality of the modules that will be developed within V4Design and of the platform as a whole. In addition, it provides the specifications for the technical infrastructure and a detailed description of the resources needed to achieve the aforementioned functionality.

co-funded by the European Union

History

Version	Date	Reason	Revised by
v0.1	30/04/2018	Creation of ToC and initial content and distribution to V4Design consortium	Luis Fraguada
v0.2	04/05/2018	Contribution on CERTH components	Spyros Symeonidis, Elisavet Batziou, Konstantinos Avgerinakis, George Meditskos, Stefanos Vrochidis
v0.3	07/05/2018	Contribution on UPF's components	Simon Mille, Jens Grivolla
v0.4	07/05/2018	Contribution on NURO's components	Yash Shekhawat
v0.5	08/05/2018	Contribution on KUL's components	Jens Derdaele
v0.6	11/05/2018	Creation of 1st integrated version based on partner's contributions and distribution to technical partners and technical manager	Ayman Moghnieh
v0.7	18/05/2018	Creation of 2nd integrated version based on received comments	Luis Fraguada, Ayman Moghnieh Konstantinos Avgerinakis
v0.8	24/05/2018	Internal review	Simon Mille
v0.9	28/05/2018	Creation of 3rd integrated version based on internal review	Ayman Moghnieh
v1.0	31/05/2018	Preparation of the final draft	Konstantinos Avgerinakis

Author list

Organization	Name	Contact Information
McNeel	Ayman Moghnieh	aymanmoghnieh@gmail.com
McNeel	Luis Fraguada	luis@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
UPF	Simon Mille	simon.mille@upf.edu
UPF	Jens Grivolla	jens.grivolla@upf.edu
KUL	Jens Derdaele	jens.derdaele@kuleuven.be

Executive Summary

This deliverable contributes to the fulfilment of the project milestone MS1 “Project setup and platform development roadmap”, which marks the successful initiation of the V4Design’s architecture development. It provides an overview of the V4Design platform, the platform integration approach, and describes the platform components, detailing their functionalities, specifications, and required resources of each. In addition, it discusses the specifications for the technical infrastructure of V4Design based on its selected architecture model.

First, it contextualizes the work on integrating the V4Design platform in Enterprise Application Integration, allowing for drawing from the integration patterns, practices and tools available in this domain. It starts by describing the V4Design platform concept in a manner that facilitates an early abstraction and classification of its components. These components are then described in details, specifying their development plan and integration model. Afterwards, the technical specifications of these components are discussed on two different levels, basic and advanced, and then platform-level specifications are introduced to govern the service development and integration. In addition, specifications related to communication and messaging are presented, followed by the expected development timeline and required resources for the platform. Finally, an in-depth discussion is presented about the enterprise bus solution adopted for V4Design based on its concept and specifications. This includes a comparative analysis of existing solutions that preceded the selection of the adequate solution for V4Design. Finally, specifications regarding the use of the Common Alerting Protocol (CAP) protocol are discussed.

This roadmap represents a common accord and a technical agreement between the partners responsible for developing and deploying services and components in the V4Design platform. It defines the approach for the implementation of the platform on several levels and addresses all the main concerns related to this type of development. It therefore should serve as guidelines for developing and integrating any component.

Table of Contents

1 INTRODUCTION

This deliverable presents a roadmap for the development and integration of the V4Design platform. It provides an overview of the technical concepts, concerns, existing approaches and patterns, and related solutions and services in order to contextualize the discussions about the development of the V4Design platform and the integration of its modules. These issues are discussed from a conceptual stance rather than from a technical stance, in order to engage the different profiles of project participants in the discussion about the platform development.

First, the concept of Application Enterprise Integration is defined and discussed in chapter 1, and its relevance to V4Design is defined, drawing from known generic types of integrated and distributed applications in order to facilitate the definition of the architecture model for V4Design.

Then, a description of the V4Design platform is presented in chapter 2, including its early concept and envisioned architecture as described in early project documents. These components are grouped into different modules, which will later allow us to study their similarities and differences, and standardize their integration to the extent possible.

This is followed by a functional description for the platform services and components in chapter 3, in which the functionalities of each service are discussed individually, and well as its development milestones and integration model.

We then identify the technical specifications of these components in chapter 4, and discuss on several levels: (1) basic specifications of components, (2) advanced specifications of components, and (3) platform-level technical specifications. In addition, the communication and messaging concerns are discussed, introducing the main messages that will be exchanged by the components, followed by the accorded development timeline based on the milestones of individual services and the global technical milestones of the project, and by a discussion of the required resources for the development.

Finally, we revise message bus solutions in the context of this project in chapter 5, defining the functionalities that are relevant to V4Design and identifying and evaluating different message bus solutions that are deemed of relevance to the platform. Based on this evaluation, a message bus solution is selected and its deployment as part of the V4Design architecture is discussed. In addition, we address the use of the CAP protocol as an integral part of the requirements for the platform, introducing its formal definition and studying its use in V4Design applications.

This roadmap should serve as an early common understanding between the project's partners to orient the development of the platform services and components. It raises several platform-level concerns that need to be addressed on the level of services, and defines lines of standardization in the way services are implemented, deployed and integrated. Building such understanding early on in the development process should alleviate the platform integration task and help to achieve a more coherent platform that meets the requirements in terms of functionalities and maturity level defined in the project proposal and plan of work.

2 APPLICATION ENTERPRISE INTEGRATION, DEFINITION AND RELEVANCE TO V4DESIGN PLATFORM

When building applications, system architects resort to three basic and generic architecture models for systems that are composed of several modules, each operating relatively independently from the others, otherwise known as distributed systems or applications:

A. **Client–server architectures** organize modules as client-side or server-side modules, where the clients retrieve data from the server to format and display to the users, and request the execution server-side processes. Client-server architectures are usually adopted for small-scale applications and their model is seldom used for composite applications that integrate several individual applications into a single bundle.

B. **Three-tier architectures** move the client intelligence to a middle tier to simplify application deployment. For instance, most large web applications are three-tier. However, the scalability of this model and its compatibility with composite applications is questioned.

C. **Peer-to-peer architectures** are otherwise a scalable and flexible solution for integrated applications. This model is a decentralized organization of several applications typically each hosted on its own machine. In general, peer-to-peer architectures do not require a central machine to manage the network resources for the integrated application. Instead all responsibilities are uniformly divided among all machines, known as peers, which can serve both as clients and as servers, or in other words can be services or user tools, or even specialize into specific roles within the integrated application.

In general, there are several approaches to peer-to-peer integrations, the simplest of which is a custom point-to-point integration that rests on connecting and chaining modules directly one to the others. This is usually facilitated by open APIs and other paradigms of application communication. It is an adequate approach when a small number of modules are integrated, and it is a relatively straightforward approach that can be implemented with ease and speed. However, the efficiency of custom point-to-point integration deteriorates rapidly and tends to be complex, expensive and difficult to maintain when the number of integrated modules grow, for instance the integration of 5 different modules may require up to 20 connections, each implemented in an ad-hoc manner. Therefore, in many cases, there is a need for more structured and standardized models of integration that can be sustained and scaled without difficulties. These are discussed in section 1.2.

2.1 Relevance of application enterprise integration to V4Design

On a macro-scale, the main technical objectives behind integrating the V4Design architecture are to reduce the number of point-to-point connections across an integrated application, allowing each service or application to preserve a level of autonomy, and to be loosely coupled with the rest of the architecture components. Without a proper integration strategy, the interdependencies and connections among the services will grow rapidly in complexity, and therefore becomes difficult to maintain, to scale, and to improve.

In practice, V4Design aim to integrate a set of different modules, each representing a service that provides certain functionalities to the system. Some of the integrated modules are user tools that govern the user processes throughout the system. Others are modules that execute autonomously and sometimes continuously, for instance to crawl new data from external sources or third-party providers. Some modules are concerned with storing and servicing data throughout the system, and others provide singular capacities such as data processing, metadata-extraction, feature extraction, analysis, reformatting, among others.

The high-level **technical challenges** that the V4Design platform faces as an integrated application can be summarized in the following three points:

- Integrated applications are **dependent on the network**'s health and performance, and unlike services or applications installed on a single server, the modules that make up an integrated application need to communicate and synchronize over a distributed network, which sometimes can suffer from delays, interruptions, data loss and other typical fall downs of digital networks.
- Usually, integrated services can be **highly dissimilar**, not only in purpose and in functional requirements, but also in the technological frameworks in which they have been developed, and in the manner by which each handles data. Also, differences are usually encountered in the way each application addresses security concerns.
- Integrated modules cannot be dependent on one another in the same way different services work in singular service-oriented applications, where they are tightly chained and highly interdependent. Each application should be allowed to evolve and change independently from other integrated applications, and this is usually achieved by **loose coupling**, which requires the use of transactions, queues provided by message-oriented middleware, and interoperability standards.

Therefore, models for integrating composite and distributed architectures are a highly relevant subject for V4Design because its system will be distributed among different servers integrating distinct services with varied levels of maturity and performance. The system may grow by integrating new tools and services in the future. Obsolete services may be replaced with more advanced ones in the same way local changes are applied without affecting other modules or components. In short, application integration models will guide and facilitate the construction of a platform that meets the required maturity and performance levels defined in the project's objectives.

For this purpose, we accord a specific attention to the application integration models and paradigms, not only from a technical stance, but also from a conceptual stance by which pitfalls are avoided and best practices adopted to insure a swift and successful integration of the V4Design platform under the available resources.

2.2 Generic models of integrated and distributed applications

Clearly, the contemplated V4Design system, which will be composed of independent applications distributed over a network, requires a "peer-to-peer" type of architecture. However, the V4Design system also incorporates legacy applications and heterogeneous services, each owned and developed by a different entity. Therefore, custom peer-to-peer architecture is not an adequate solution, and we need to consider more elaborate models of

peer-to-peer architectures for distributed applications. Concretely, we look at three models, being Enterprise Service Bus (ESB), Service-Oriented Architectures (SOA), and Enterprise Application Integration (EAI).

Enterprise Service Bus (ESB) architecture is essentially a set of rules and principles for integrating numerous applications together over a bus-like infrastructure. The core concept of the ESB architecture is the use of communication bus as the main paradigm of the integration and as the system's core component, enabling each application within the architecture to establish a direct communication with the bus. This decouples applications from each other, allowing them to communicate without dependency on or knowledge of other systems on the bus. The concept of ESB was born out of the need to move away from point-to-point custom integration, which becomes complex and hard to manage over time.

A Service-Oriented Architectures (SOA) is essentially a collection of services that communicate with each other using a communication protocol. A service is defined as a logical unit or a self-contained process that can be accessed remotely by other services. A service can be composed of other elemental services or components, but it is treated as a single unit or "black box" by the other services in the architecture. This independence among the services allows each service to work, evolve, and grow with no strings attached, and therefore this flexibility facilitates the integration of the application and its maintenance, and allows for greater customization.

By definition, Enterprise application integration (EAI) is a paradigm that supports the unrestricted sharing of data and processes throughout a set of networked applications in an organization. The purpose of this integration is to support the seamless integration of processes and services among the different modules of the architecture. In addition, this integration aims to provide a unified interface for all of the integrated system components that share and exchange data in an organized and mediated manner. Unlike ESB and SOA, EAI has been devised to remedy a situation in large enterprises that have developed legacy software in ad-hoc manners and now is facing the need to integrate them. It is a framework of middleware tools that allows analysts to establish integration principles within an enterprise environment and to encourage the use of software solutions in integrating the different applications in this environment.

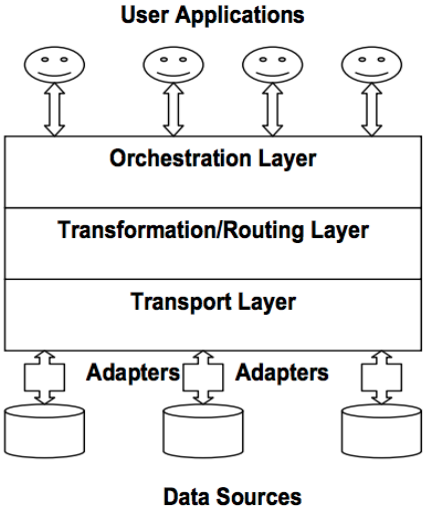

Figure 1: Gorton's generic definition of EAI

Gorton et al. **[1]** provide a simple and straightforward definition of EAI visualized in Figure 1. Accordingly, EAI has **five generic components being**:

i. ***User applications*** allow users to view and manipulate data, having custom graphic-user-interfaces designed to support this purpose;

ii. ***Adapters*** establish and maintain the access mechanisms to the data and support the deployment of data transformation services (known as intelligent adapters);

iii. Besides, and between the user applications and the data adapters is the middleware which essentially is composed of three layer, the ***Orchestration layer*** that accepts user requests and manages the processes required to accomplish them,

iv. the ***Transportation and Routing layer*** manages and rapidly executes data transformations from one format to another as required, and

v. the ***Transport layer*** that is responsible for data transport between different data sources.

It is worth noting that these models share a lot of features and are not mutually exclusive or essentially do not represent distinct options. In essence, they are different strategies to build composite architectures, and most implemented composite architectures draw on more than one model. In Table 1 we present a high-level comparison of these three architecture models for distributed and integrated applications.

	Advantages	**Disadvantages**
Enterprise Service Bus (ESB) solutions	• Common solution to which a large variety of open-source and legacy solutions exit. • Enforces a level of standardization throughout the application • Efficient and consistent, and highly reliable for small to mid-size distributed applications. • Easy to maintain and manage.	• In general, there are some concerns about the scalability of such model and its impact on slowing the evolution of the system. • Single point of failure.
Service-Oriented Architecture (SOA) solutions	• Arguably, the single two most important advantages of SOA are scalability and platform independence. • In addition, it is easy to upgrade and service, and highly reliable.	• Relatively complex to develop. • Have a relatively higher machine costs and response time.
Enterprise Application Integration (EAI) solutions	• Supports complex distributed asynchronous processes. • Very flexible in terms of architecture organization.	• Not very suitable for small architecture, but much better suited for highly complex architectures.

	• Efficient in data transportation and formatting.	• Difficult to build and sometimes to maintain. • May have performance issues related to the type of middleware used, and dependencies among its components.

Table 1: Advantages and disadvantages of ESB, SOA, and EAI

In fact, ESB, SOA and EAI represent three different levels of application integration, often compared to Russian stacked dolls, one encapsulating the others. The smallest doll is the enterprise service bus because it is at the essence of SOA and EAI architectures, and the most generic and straightforward solution. Next is service-oriented architecture, which is a specific model of implementation with little variation (REST and SOAP frameworks are the most common solutions). In SOA, services adhere to standard communications agreements, as defined collectively by one or more service-description documents. Each service needs not know the details about how other services are provided in the architecture. Finally, EAI is the most elaborate definition of application integration model. It uses SoA and ESB principles and expands on them, introducing concepts and patterns that are more relevant when integrating large legacy applications together. For instance, it is an efficient framework to adopt to integrate platforms of merging companies, or in binding the business software of different departments.

3 DESCRIPTION OF THE V4DESIGN PLATFORM EARLY CONCEPT

V4Design is a platform that aims to enable the re-use and re-purpose of multimedia content for architecture and game design and other areas of design applications. Therefore, it will develop a platform that is able to acquire relevant content, extract assets from this content, and generate more descriptive data and information about the extracted assets. This will provide architects, video game designers and other designers with an innovative tool set that enhances the creative phase of the designing process. These tools will enable the recycling and reuse of visual and textual assets extracted from movies, paintings, virtual environments, 3D models, and other digital mediums.

Hence, the purpose of the V4Design platform from a technical point of view is to enable the integration of a multitude of tools that together can support the services promised by the project's objectives. These tools include a set of data crawling and retrieval tools that will gather data from third parties and available online libraries and repositories, including those owned by the consortium partners (DW, EF, AF, SLRS), in addition to freely available content on the web.

This "raw" data will then be analysed by a series of tools that extract 3D and VR representations of objects, buildings and cityscape environments, and other multimedia data and associated textual information. More specifically, V4Design will build upon the concept of semantic integration of heterogeneous 2D multimedia in order to generate enhanced dynamic 3D structures and environments as realistic and comprehensive representation of structures of interest.

In addition, other tools wills generate textual summaries from retrieved commentaries, reviews, and critics of the analysed artwork and tuned to designers' interests and profiles. This semantic knowledge will act as complementary material to support the design process. Finally, the platform will also present the user with a set of design tools that will allow architects, designers and video game creators to leverage the value of the collected, analysed and indexed resources. These design tools will make the wealth of 3D, VR, aesthetic and textual information easily accessible by the users who can proceed to re-purpose them according to design tasks.

3.1 Choosing an integration model for V4Design

Custom point-to-point integration is not an adequate solution for an application on the scale of V4Design platform, which requires more streamlining in data transfer and the execution of multi-service processes that span several modules that are different types of open and proprietary systems, each with its own development, database, networking and operating system, etc. Therefore, V4Design has an inherent need for standardized communication among the application modules, which requires the use of middleware solutions, which at their very basic are ESB-based, and in their most overreaching or ambitious form a complete EAI solution.

Pragmatically, the scope of the V4Design implies that several aspects or concerns of Enterprise Application Integration (EAI) may not be relevant, as the size and complexity of the envisioned V4Design platform is smaller than typical cases where EAI is adopted. Therefore, we will define the type or scope of enterprise integration model to adopt for

V4Design based on the detailed definition of its components and services, and on the requirements for communication, security management, performance, and other relevant aspects of system architecture design. These concerns will be explored in the coming sections.

In the following, we describe the early concept of V4Design platform and its components and services as detailed in the project proposal and implementation plan.

3.2 Early concept and envisioned architecture design

The first vision of the V4Design architecture was developed before the start of the project, as part of the efforts of building the project proposal and combining the technical contributions of the partners into a single and powerful platform. This exercise included a definition of services, components, and processes based on the project's technological objectives as well as on the added value of existing tools and services that constitute the project's background. In addition, the project's foreground is envisioned to conceive and integrate a series of new components, which will be accounted for in the architecture integration plan.

Based on this early concept of V4Design architecture and the project proposal, we define the following set of modules as integral parts of the system architecture, describing the overall role of each module and its owner in Table 2.

Module	Description	Owner(s)
User tools	This module encapsulates the 3D design tool and the NURO VR Authoring tool planned for V4Design. Both tools will act at the platform's user interface or in other words a user-oriented toolset.	NURO, McNeel
Data Processing and Analysis	A set of tools for feature extraction and metadata classification of assets. Includes Complete Object Creation, 3D Model Extraction, Extraction of visual features, Text generation, Building Localization, and extraction of textual features.	KUL, CERTH, UPF
Non-semantic data storage	Stores non-semantic data such as 3D models, videos, data retrieved by crawlers, images, and others.	CERTH
Semantic data services and storage	This module encapsulates the semantic data services and storage, including statistics-based, rule and knowledge-based solutions to support deeper and more robust analysis of textual content. Existing experimental reasoning frameworks will be extended to support sophisticated interpretation tasks for managing incoming information and for retrieving information pertinent to the users' needs.	KUL, UPF, CERTH

Web & Social Media Crawlers and DB wrappers	V4Design crawlers will build upon CERTH's domain-specific search and social media tracker that has been deployed in several projects and is based on robust and mature open source technologies, such as Apache Nutch. This module also includes the development of data wrappers that integrate 3rd party databases.	CERTH, EF

Table 2: General service-oriented modules of V4Design platform

The early concept of the architecture design considers how these modules interconnect and collaborate to support the processes that V4Design aims to bring to life. The following Figure 2 represents this early version of the platform architecture design. It envisions the use of a Message Bus as a main component of the platform's middleware, to which all modules connect to send and receive messages from other modules. The user tools are integrated with a backend component that i) facilitates the data queries and consultations that the user wishes to execute, and ii) streamlines the processes required to respond to the user interaction. The user requirements will be implemented in the user tools and supported by the rest of the system.

The system data are classified in two general categories: semantic data and non-semantic data, and for each data type, a series of tools is associated. The semantic data services and storage module (shown in the upper-right section of the diagram) groups the services that organize, store and provide semantic data, while the non-semantic data storage module (located below the semantic data services and storage in the diagram) stores and provides non-semantic data. The web & social media crawlers and database wrappers module (bottom-section of the diagram) is responsible for acquiring the raw data that will be processed by the system. The data analysis and processing modules takes charge of this responsibility.

Currently, there are no products/tools that combine the series of services and analytic functionalities that V4Design plans to integrate. Therefore, the integration approach should take into consideration the flexibility required in early-stage application development, by adopting a simple, extendable and largely generic model of integration. Accordingly, middleware services and components are grouped into modules based on how close they interact with each other directly, and the overall capacity expected from their integration. Also, in this grouping we take into account the type of data generated, exchanged, processed, and stored. The objective of this grouping is to explore the possibility or the need of tightly integrating the components of each module into a single service from a platform architecture viewpoint. Additionally, this grouping helps to simplify our analysis of the architecture requirements as it encourages a standardisation in the definition of similar and complementary components.

According to this definition, the services grouped into a single module can communicate with the message bus either through a single interface, or through different instances of the same interface object. Using a single interface (for instance, in the case of Non-semantic data storage, which in face encapsulates different databases and sources) facilitates the execution of "parallel" tasks and to some extent chained tasks that involve more than one of the encapsulated services. On the other hand, encapsulated services may use different instances of the same interface to communicate with the message bus (for instance, in the

case of the user tools, each can operate its own interface), which is useful in case of distributed module architecture. In this case, the message bus will register each interface as an independent point of communication.

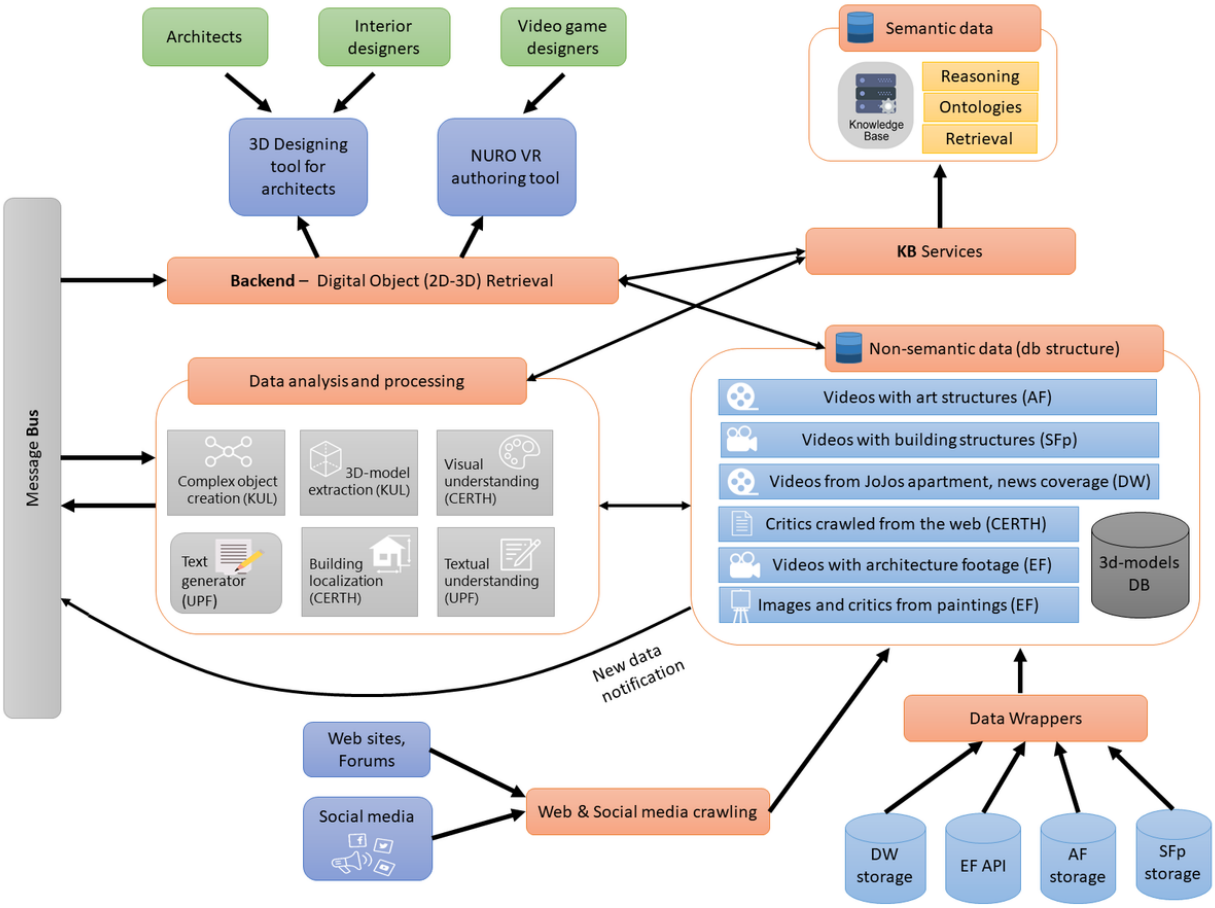

Figure 2: Early version of V4Design architecture design

From an architecture stance, these modules are considered as self-contained components that could integrate several services and different functionalities, but respond and act in the integrated system as a single unit. We group similar services and complementary services into single modules. In the architecture design, peer-to-peer communications between modules are minimized and restricted to data retrieval by URI associated to data queries. All modules notify the message bus when their availability or status changes in order to adjust message routing and inform other concerned modules. Additionally, the modules monitor the availability of related services through the message bus. For instance, the user tools may disable functionalities in case the related middleware was offline.

Accordingly, we build a high-level integration model that shows how the modules communicate through a central message bus. Limited interaction between user tools and modules with data storage can be supported. This is shown in Figure 3.

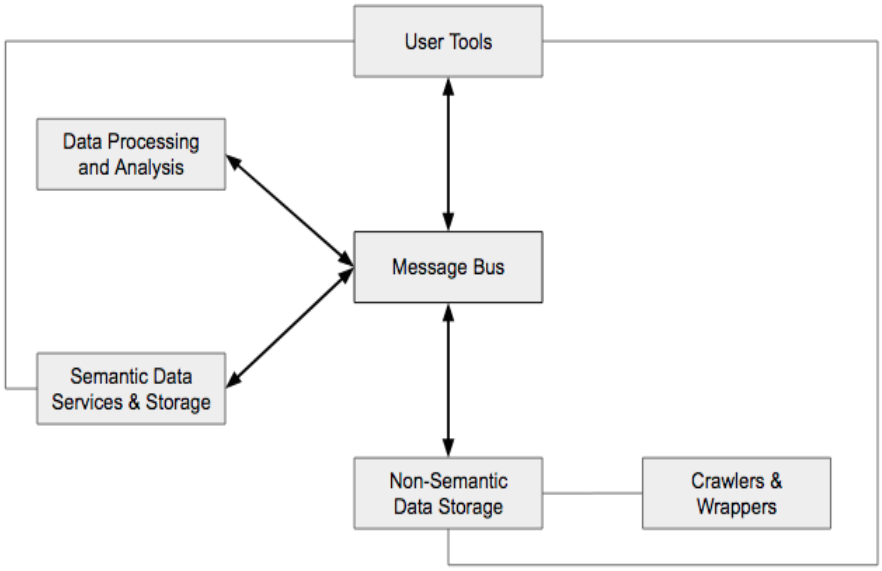

Figure 3: Module-based abstraction of the architecture design

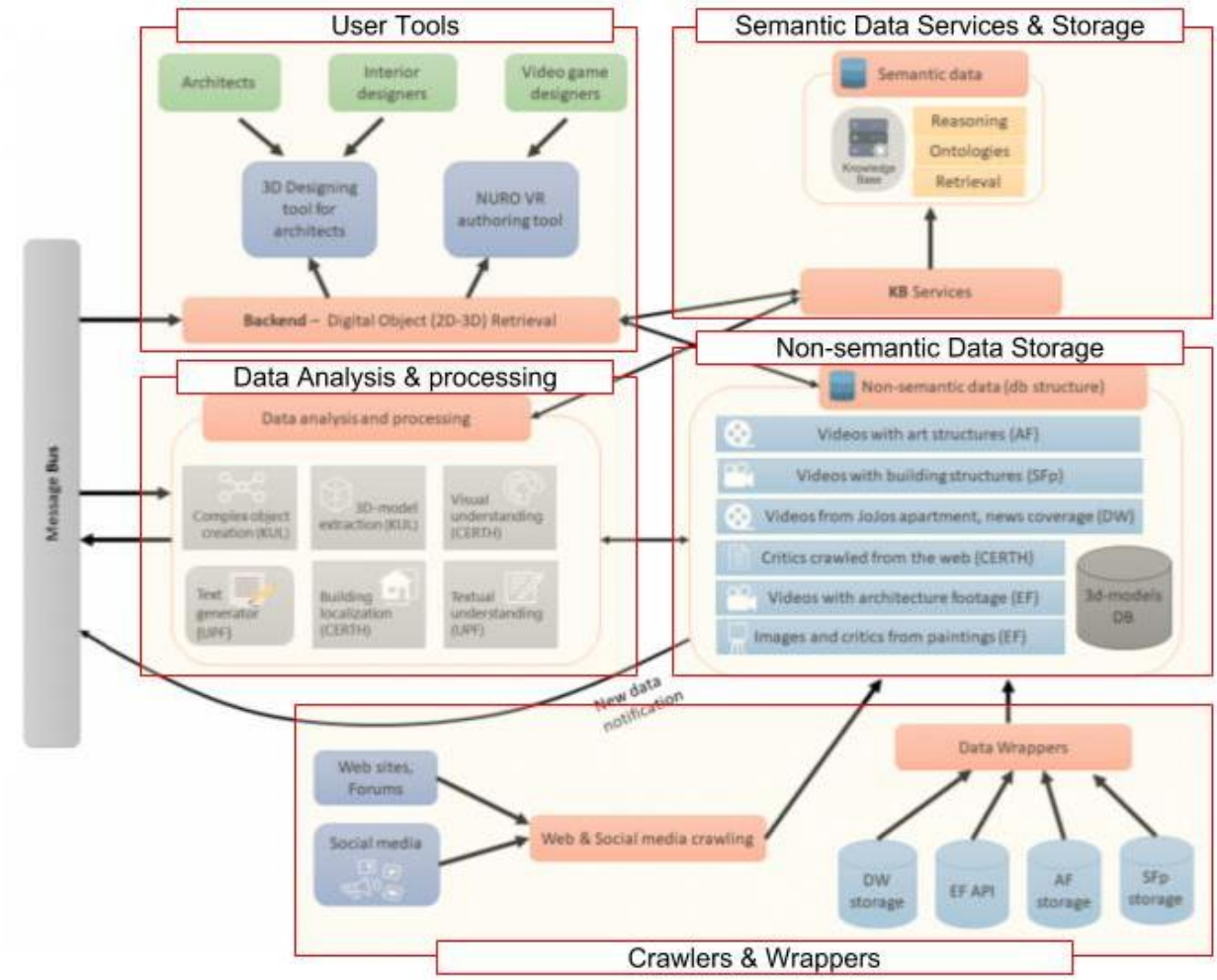

Figure 4: Encapsulation of the architecture components

Based on this approach, we refined the original design of the application architecture to define more abstract modules. This is an abstraction of the early design concept as shown in Figure 4.

In Table 3, we describe how each of these modules interacts with the message bus according to its overall role in the integrated architecture and the technological objectives of the project.

Module	Message bus integration
User tools	Sends petitions and queries according to the user interaction, receives responses to queries, including data retrieval responses with URIs.
Data Processing and Analysis	Receives petitions for analysis (upon new data arrival), and user-related requests for data processing.
Non-semantic data storage	Receives user queries for retrieval. Responds with data URIs. Notifies the system when new data is imported via crawlers or wrappers.
Semantic data services and storage	Receives petitions for semantic analysis, and queries for semantic information stored in the knowledge base. Responds with data URIs, and light semantic information can be integrated in the message content.
Web & Social Media Crawlers and DB wrappers	Will not be integrated with the Message Bus. The non-semantic data storage will notify the message bus when new data becomes available.

Table 3: Relationships between modules and the message bus

In the following chapter, we discuss each architecture module in more details, defining and describing its elemental services and the manner by which they integrate and operate in the system architecture. This definition is based on a detailed inquiry conducted to define these components as “black boxes”, integrated within the overall architecture. The architecture design is consequently revised and refined afterwards.

4 DESCRIPTIONS AND FUNCTIONALITIES OF THE V4DESIGN PLATFORM COMPONENTS

The following components constitute the main modules of the V4Design architecture. Each component describes an independent service with a specialized function in the overall system. Some acquire data, others extract features, some cast or format data objects, some provide support for user-oriented functions, and some are user-oriented tools.

We describe each component according to its overall role in the V4Design platform, focusing on the functionalities that it supports and the capacities it brings forth. In addition, we list the expected development milestones and discuss shortly the relationship between the component and the rest of the components from an integration perspective.

4.1 The V4Design user tools

The user tools are a set of applications designed to service different user profiles and provide functionalities to professionals based on the value generated through the analysis of data. They are mainly concerned in retrieving processed data and assets from the system. They manage the system users independently from the rest of the platform

4.1.1 The Rhino Design Tool for architects

General description: The V4Design for Rhino project is a plugin for the Rhinoceros 3D 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 Non-uniform rational basis spline (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.

Rhino users can extend its default functionality by creating plugins with the various public APIs provided by Robert McNeel & Associates. A plugin is a compiled dynamic link library (DLL) that is loaded by Rhino. Plugin functionalities can be presented to the user in various formats, including as an additional command to be entered through Rhino's command line, or as a graphical user interface (GUI) hosted in a window dockable to the Rhino interface.

Robert McNeel & Associates develops public APIs in several languages (http://developer.rhino3d.com/api/) including C++, .NET, Python, VBScript, RhinoScript, the .NET RhinoCommon API will be used to develop the V4Design for Rhino. RhinoCommon has been selected as the API due to: team expertise and general adoption of .NET in developer communities. The latter is an important point, as the plugin will be developed as an open source initiative and community contributions should not be limited by the learning curve of the language.

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.

Within the Rhino interface (Figure 5), the V4Design for Rhino plugin will be presented as an interface panel alongside other existing panels, which ship with Rhino, such as the Layers and Properties panels on the right side of the interface. Within this panel, a GUI will be developed in HTML5, JavaScript, and CSS to ensure portability to other platforms.

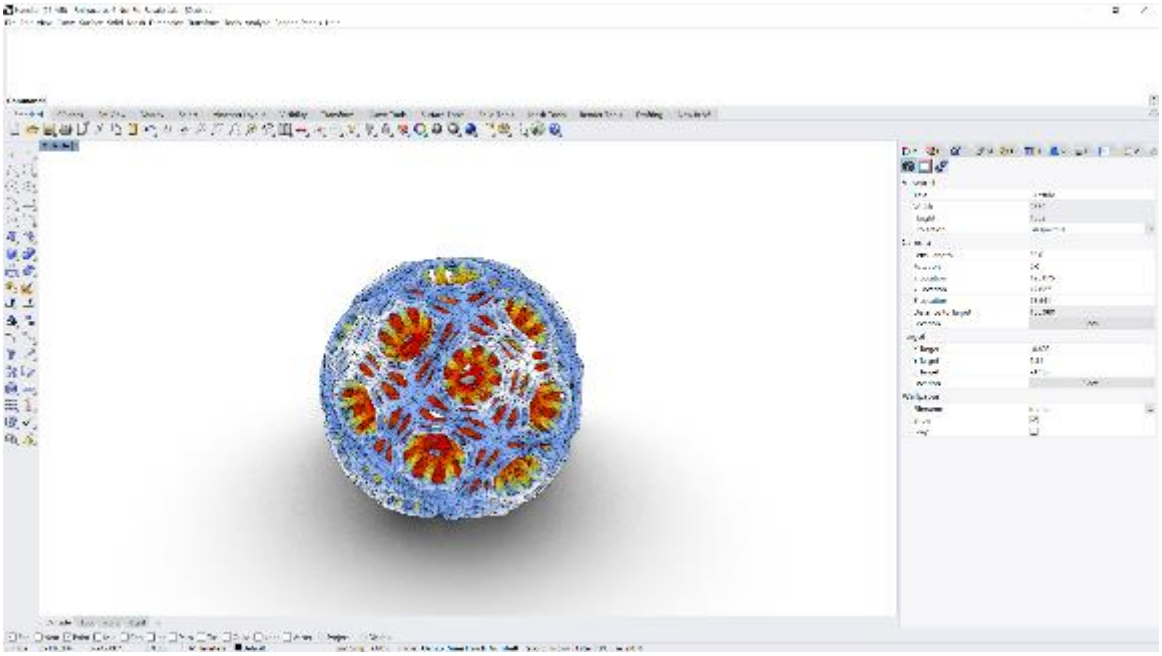

Figure 5: Rhinoceros 6 User Interface

Input: Search query via keywords, location, tags, and other relevant search filters.

Output: Relevant assets from the V4Design Asset Repository

Development milestones: The V4Design for Rhino plugin is currently in-development. Its expected milestones are the following:

- [M12] First version: Basic operational version with ability to query the V4D Asset Database or dummy database with keyword queries. Results are displayed to the user as thumbnails of the retrieved assets and some basic information. User can introduce the retrieved asset into the authoring tool.
- [M18] Second version: Development of query capabilities, including the addition of filtering by tags, location, asset type, etc. App queries the real V4Design Asset Repository (instead of a dummy repository).
- [M26] Third version: Development of review capability, allowing the user to be able to add a small review of the asset. This data will be attached to the asset in the V4Design Asset Repository.
- [M34] Final version: Fully developed query capabilities, review and further personalization based on user preferences (keeping a catalogue of retrieved assets).

Integration: Since the V4Design for Rhino plugin needs to query the V4Design asset repository, it will access the asset metadata through the V4Design REST API. The V4Design for Rhino plugin therefore communicates exclusively with a backend service (the V4Design REST API) in order to query and retrieve assets from the V4Design asset repository. Since the V4Design for Rhino plugin is only interested in asset querying and retrieval, it need not intervene in the asset production facilities of the V4Design platform, and thus, does not need to communicate with the message bus.

4.1.2 The NURO VR Authoring tool

General description: 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.

The NURO VR Authoring tool will use the native functionalities of Unity but will add various new functionalities to author VR games and use of V4Design repositories to extract assets and 3D models for the games. Acting as a plug-in of Unity, a new tab will be added in the Unity, which will be connected to the V4Design Backend tool.

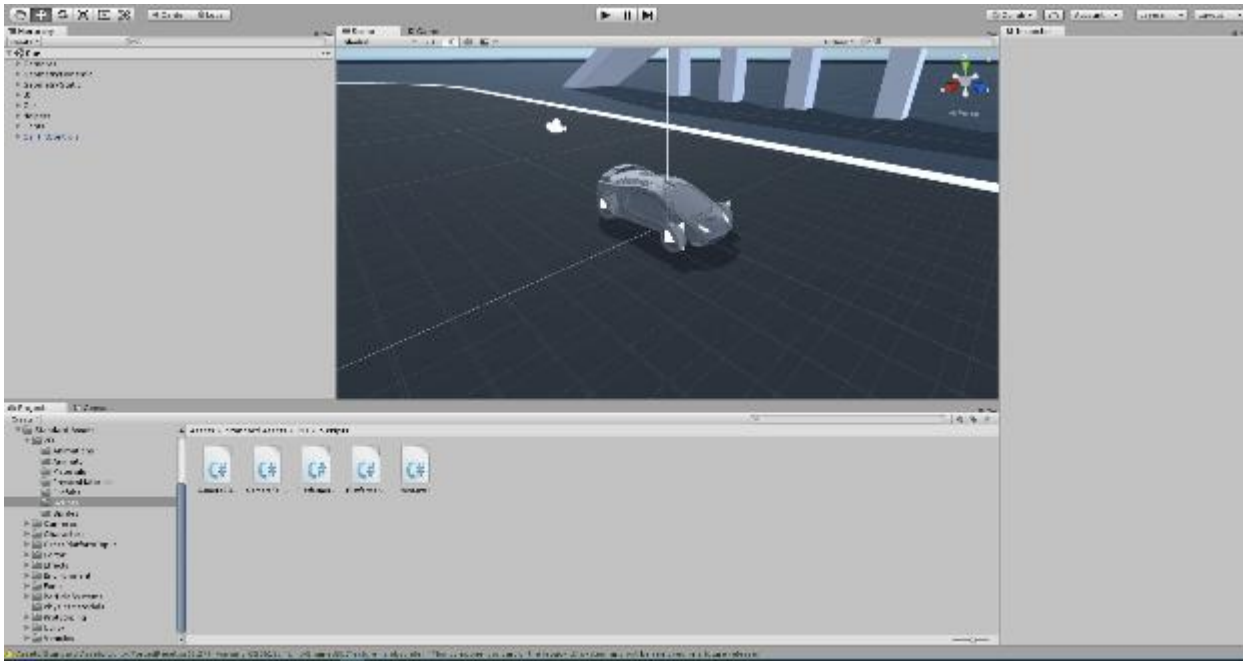

Figure 6: Unity3D Editor screenshot

Figure 6 shows a sample editor screen of Unity, which has tabs of the Game, Scene, Hierarchy, Inspector, Project and Console. Similar to this, another tab will be included, which will use the interface of the V4Design Backend Tool to show the repository of assets available for the users. The tool will be developed on C# and will use the APIs provided by Unity to integrate into the editor. Apart from this, the tool will provide new functionalities to

the users to easily create VR environments, test them and produce gaming elements with pre-defined designs.

Input: Search query via keywords, location, tags, and other relevant search filters.

Output: Relevant assets from the V4Design Asset Repository

Development milestones: The Authoring tool is currently in-development. Its expected milestones are the following:

- [M12] First version: This version will include the integration with Unity, ability to query from V4Design repository and import 3D model into the scene. Moreover, this will include the easy ability to change the year of a particular model and change the years inside the VR application.
- [M18] Second version: The second version will include the ability to export VR game using a specific environment along with various game features and interactions.
- [M26] Third version: The third version of the VR authoring tool will include all the requirements from the users, results of the evaluations from V1 and V2 and will be a pre-final version.
- [M34] Final version: This version will include all the bug fixes and the feedback from the users and packaged along with the V4Design product.

Integration: Follows exactly the same paradigm described for the Designer tool.

4.1.3 The V4Design REST API for user tools

General description: 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.

The V4Design REST API will be designed to meet Open API Specification [OAS] (https://github.com/OAI/OpenAPI-Specification) criteria, including utilizing specific query and response formats for the different calls and responses. To facilitate this, the API will utilize Swagger (https://swagger.io/), Apiary (https://apiary.io/), or similar as a framework for designing, developing, documenting, and deploying the API.

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.

Input: User petitions and queries

Output: V4Design system queries and data retrieval requests

Development milestones: This component is currently a concept, its expected development milestones:

- [M12] - Basic operational prototype that allows querying the V4Design Asset Repository and receiving results based on the available content.
- [M18] - Query and Result calls to be further developed. Starting the documentation process.
- [M26] - Query and Result calls to be further developed.
- [M34] - Final functionality and documentation complete.

Integration: This service will communicate directly with the database system in charge of managing the V4Design Asset Repository via a RESTful API, adhering to the Open API Specification. Its communication with the message bus is minimal (authentication of services).

4.2 Data processing and analysis tools

The set of data processing and analysis tools mainly deals with non-semantic analysis of acquired data, focusing on data formatting and data extraction, and data classification mechanisms that implement the V4Design approach.

4.2.1 The Spatio-Temporal building and object localization (STBOL) service

General description: Spatio-Temporal building and object localization in images and video frames service aims to detect buildings and some of their basic elements (i.e. type of window, door, roof, decoration, facade, etc.) from images or video frames. The end-user will give to the system images or videos and it will return masks of frames having bounding boxes that include buildings and some of their basic elements.

Building and interior elements localization will be applied on art and architecture-related movies, documentaries and multiple art-images, aiming to localize the exteriors of buildings, and outdoors spaces. For these purposes, scene recognition and image segmentation algorithms will be applied in history or modern city architecture movies and documentaries so as to localize modern and historic cityscape environments. In this manner, sub-sequences containing building exteriors and cityscape environments of interest to designers (video game creators, architects), can be extracted from collected video content, repurposed and reused in a meaningful and innovative way.

This service will store and update the localization models that will be used to understand the objects and architecture elements that exist in the provided videos and images. More specifically, we foresee the construction of two localization models: (i) one for the identification of buildings and other outdoors elements, (ii) one for the detection of architecture objects and other elements that exist in interior spaces.

Input: Images, video frames and other assets.

Output: Object and building boundaries in images and video frames.

Development milestones: This service is currently under development and will be delivered according to the following milestones.

- [M6]: Initial version of the basic STBOL component is released
- [M12]: Service will be integrated in the platform and message bus
- [M16]: The basic version of the algorithm will be delivered
- [M20]: The component will be integrated in V4Design system 1st prototype purposes
- [M26]: Integration with the second V4Design prototype.
- [M34]: Advanced version of STBOL component will be deployed.

Integration: This service will communicate exclusively with the database of the system. However, it may connect to the message bus for authentication purposes and to receive broadcast messages.

4.2.2 **The Aesthetic Extraction and Texture Proposals (AE&TP) service**

General description: The Aesthetic Extraction (AE) from paintings component aims to extract and categorize the aesthetics of paintings based on their style (i.e. impressionism, cubism and expressionism), creators and emotion that they evoke to the viewer. This service will be used to categorize the acquired paintings and create an aesthetic gallery from where an end-user could choose or be inspired from, in order to create novel architecture structures or other artworks. The V4Design API will provide to the end-users the capability to use keywords in order to search the knowledge base and acquire the aesthetics, style and emotion of paintings, images and pictures of artworks that they want.

This component will analyse the acquired visual content from paintings and images of other kind of artwork in order to extract geometry, style and other aesthetics aspects concerning specific artwork collections. It will then provide it as metadata to the V4Design platform, and artwork features so as be specified in measurable attributes, such as colour (RGB, HSV, etc.), texture, image bumps, gradients, palettes, and patterns.

Texture Proposals (TP) from paintings and other artwork images will use the categorized paintings by AE component so as to create novel aesthetics and style textures which will be provided to architects and video game designers, within the aesthetics gallery, so that they can create novel artwork and structures based on past observations and styles. For the interface, the end-user will also have the possibility to give images to the V4Design platform and the service will return a texture extracted from the paintings, images and other pictures of artworks.

This service will store and update the aesthetics model that will be used to build the aesthetics V4Design gallery. More specifically, we foresee the construction of three galleries based on the aesthetics of the image (i.e. cubism, abstract, renaissance, etc.), painter (i.e. Bernini, Picasso, etc.) and emotion induced in the viewer (i.e. fear, sadness, neutral, etc.).

Input: acquired visual content from paintings and images.

Output: Not yet defined.

Development milestones: This service is currently under development, with expected delivery according to the following milestones:

- [M6]: 1st version of the basic aesthetics component is released
- [M12]: service integrated with the platform and message bus
- [M15]: The basic version of the algorithm will be delivered
- [M20]: The component will be integrated in V4Design system 1st prototype purposes
- [M26]: Integration with the second V4Design prototype.
- [M33]: Advanced version deployed.

Integration: The service does not communicate with other services since it takes and returns messages from and to the database respectively.

4.2.3 The 3D-Reconstruction Service

General description: The 3D-Reconstruction service will be used to convert initial input data into 3D point clouds and meshing. Input data will be initially analysed to determine reconstruction suitability. The service will support multiple output formats. The entire reconstruction pipeline is heavily resource demanding. Therefore, intermediate results will be available in order to cancel a reconstruction process if they are not up to certain standards. 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, being the most resource intensive, will be the one providing intermediate results.

If a set of input imagery is determined suitable for 3D-Reconstruction a new reconstruction object will be allocated. Any further command towards this service (e.g. 'mesh calculation') will always reference a specific reconstruction object (by for example a unique ID).

This service will implement well-established 3D-Reconstruction algorithms based of several available tools (colmap, theiasfm, visualsfm, opencv, etc.). The V4Design project will also implement new research opportunities towards reconstruction of data unsuited for traditional multiple-view reconstruction and these will be implemented in this service as well.

The core of this service is expected to be implemented using the C# language (likely .net framework 4.5+ or possibly .net core). Specific 3D-Reconstruction algorithms can be implemented in several programming environments and languages like C++ and Python (depending on prior research and available open source projects).

Input: Initially grouped images or video files. Data from visual understanding tool.

Output: 3D point clouds. 3D Mesh. BIM objects. Reusable texture.

Development milestones:

- [M12] Initial prototype. The service accepts suitable data and runs the initial photogrammetry pipeline. Initial integration with message bus.

- [M20] Checks regarding reconstruction feasibility. Improved reconstruction pipeline specifically regarding the multiple input of images and videos to a single reconstruction. Output may be requested in various quality (vertex count) and formats. Initial capabilities for alternative output like facades and reusable textures.
- [M28] Further enhancement and segmentation of output. Implementation of linked data and initial acquisition of BIM objects.
- [M33] Final version of the reconstruction service. Improvements regarding performance and scalability of the service.

Integration: The service works independently from other services and is concerned with the availability of new data to process in the system. It will receive broadcast messages from the message bus, and can access data directly.

4.3 Semantic data services and tools

The semantic data services centre on the extraction, classification and analysis of semantic information that is related to the acquired visual and graphical assets. The complement performs the analysis of these assets that aims to extract and process sharable or reusable elements.

4.3.1 The KB Population service

General description: The Knowledge Base (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. This involves the development of vocabularies for capturing:

- The aesthetics extracted from visual content (i.e. images and videos) and the textures generated in order to retrieve relevant artwork attributes (Aesthetics and Texture extraction modules in WP3)
- The semantic relations (e.g. named entities, concepts and relations) extracted from textual analysis, along with various properties, such as artists, year etc. (Language analysis module in WP3)
- Buildings, interior objects and other content-specific attributes (e.g. landscapes, architectural styles, etc.) that will be extracted from video and image analysis modules (Object Localisation module in WP4).

The underlying knowledge structures will also provide all the necessary semantics needed to generate textual descriptions and summaries for each asset (Language Generation module in (WP5). 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.

Input: Analysis results of visual content (e.g. tags, building, objects, textures, aesthetics from images and videos), concepts, entities and relations extracted from textual content.

Output: RDF-based representation format

Development milestones: The KB Population service is currently a concept, and will be developed and delivered according to the following milestones.

- [M6]: The skeleton of the service will be available, e.g. a dummy service able to receive and send messages to the bus, without any mapping functionality.
- [M12]: Basic mapping functionality will be available towards v1. This involves the delivery of the mapping algorithms able to populate the KB with real results generated by the current version of the V4Design components. The interaction with the bus will be also implemented and tested, aligning the subscription mechanisms to the events published by the analysis modules.
- [M20]: Fully fledged mapping service, supporting the full structure and content of the outputs generated by the V4Design modules for v1. The mapping algorithms in M20 will extend the ones developed in M12, taking into account updates and refinements made in the V4Design modules to address the technical and user requirements.
- [M28]: Necessary updates for v2, in line with the updated structure and content provided by the analysis modules. This involves the update of the mapping algorithms to support the richer inputs that will be provided by the components, as well as to update the publishing and subscription mechanisms to the bus in order to realise the communication with the other modules of the framework. Special focus will be given on the semantic enrichment of the incoming information, e.g. by including additional references to Linked Data resources.
- [M36]: Necessary improvements on the final system, according to the updates made on the output (structure and content) provided by the other components. In the final version the focus will be also given on the scalability of the mapping algorithms, as well as on developing fall-back strategies when the incoming information is incomplete. The possibility of a tighter interaction with the Reasoning service will be also investigated, according to the need to incorporate some sort of reasoning in the mapping process (this depends on the semantics of the input that will be provided).

Integration: This service will communicate with other components through the message bus.

4.3.2 Semantic Integration and Reasoning

General description: The reasoning service (WP5) will be responsible for further analysing the knowledge captured in the Knowledge Base (KB). More precisely, the module will try to build a unified representation of the available assets, taking into account information relevant to texture and aesthetics (Aesthetics and Texture extraction modules in WP3), named entities, concepts and relations extracted from textual analysis (Language analysis module in WP3), as well as buildings, interior objects and other content-specific attributes (Object Localization module in WP4). 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 (WP5), supporting the decision-support aspects of the V4Design platform, according to the use case requirements that will be defined. Overall, 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.

Input: The information in the KB.

Output: Additional inferences in the RDF-based representation format.

Development milestones: This component is currently a concept, and its implementation has not been started yet. It will be developed and delivered according to the following milestones:

- [M6]: The skeleton of the service will be available, e.g. a dummy service able to receive and send messages to the bus, without any reasoning functionality.
- [M12]: Basic reasoning functionality will be available towards V1. This involves the development of the rule-based reasoning framework able to combine existing tags and generate high-level concepts, semantically enriching the captured context.
- [M20]: Reasoning functionality aiming to address the V1 requirements. This involves the extension of the reasoning framework developed in M12 with advanced multimodal information fusion and content aggregation techniques to generate higher level conceptualizations for content repurposing. A hybrid reasoning scheme of Description Logics and rule-based reasoning will be investigated.
- [M28]: Necessary updates for V2, based on the evaluation of V1 and the new input provided by the other modules. In addition, the reasoning framework will be further enriched with non-monitoring capabilities, addressing challenges relevant to content disambiguation and handling of conflicts, e.g. in the case when conflicting information is received from different modules.
- [M36]: Necessary updates for the final system. Improvements on the scalability will be investigated, while similarity measures will be implemented for advanced Linked Data resource linking and approximate reasoning (e.g. to define clusters of relevant assets)

Integration: The Semantic Integration and Reasoning service will communicate with other services, namely the KB population, via the message bus.

4.3.3 The TALN Language Generation service

General description: 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. Starting from the repository that contains the extracted contents from the documents processed by the analysis modules, the following sequence of actions is performed:

- text planning identifies contents related to the queried entity, assesses their relevance relative to this entity, and produces an ordered sequence of linguistic predicate argument;
- linguistic generation starts by transferring the lexemes associated to the semantics structures to the desired target language;

- the structure of the sentence is determined;
- grammatical words are introduced;
- all morphological agreements between the words are resolved;
- the words are ordered and punctuation signs are introduced.

As for analysis, we follow a pipeline approach based on a theoretical model being the Meaning-Text Theory.

Input: data extracted from text, webpages, and/or visual analytic

Output: textual reports, descriptions, or summaries

Development milestones: This component is currently in development and will be delivered according to the following milestones:

- [M12]: Operational prototype. Generation of a few sentences from ontological representations will be supported in English. Some basic summarization techniques (e.g. extractive summarization) will be implemented for handling possible textual inputs.
- [M16]: Basic summarization techniques. The generator starting from ontological structures will be adapted to one or two more languages, and its coverage will be increased (all depending on the UC requirements). A first version of the ontology-based text planning will be setup and connected with the generator.
- [M34]: Final summarization techniques. The ontological generator will handle all V4Design languages (English, Spanish, Greek, and German) and cover all UCs, and will include statistical sub-modules when needed. The advanced version of the text planning module will be released, which will aim at optimizing the relevance and coherence of the summaries. Efforts will be dedicated to ensure the reusability of the developed tools outside of V4Design.

Integration: The TALN Language Generation service will communicate with the other platform components through the message bus.

4.3.4 The TALN Language Analysis service

General description: 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. The analysis pipeline comprises the following sub-modules:

- Tokenization: identify the token boundaries;
- Part-of-speech tagging: assign grammatical categories to tokens (*noun*, *verb*, etc.);
- Lemmatization: determine base form of tokens (*built* -> *build*);

- Word Sense Disambiguation, Entity linking: assign particular senses or referents to tokens;
- Surface-syntactic parsing: assign grammatical relations between tokens
- Deep-syntactic parsing: assign predicate-argument relations between meaning-bearing tokens (first argument, second argument, etc.)
- Conceptual relation extraction: map to language-independent abstract structures.

The V4Design text analysis framework adheres to a multi-layer paradigm: starting from the input texts, representations of higher abstraction are successively obtained, until the underlying semantics are distilled in a formal and language-independent manner that allows for automated reasoning and interpretation.

Input: Textual material from reviews, image captions, etc.

Output: JSON file with syntactic and semantic annotations on top of the sentences.

Development milestones: This component is currently in development, its expected milestones are the following:

- [M12]: Operational prototype. The language analysis pipeline, with all the aforementioned components, will be able to output language-independent representations starting at least from English, for a limited set of input sentences.
- [M18]: Basic version of multilingual text analysis. The analysis pipeline will be operational for at least three languages, and its coverage will be improved according to the specifications of the different UCs. The quality of the outputs will be evaluated.
- [M33]: Final version of multilingual text analysis. The analysis pipeline will have an improved coverage and will be able to handle all the V4Design languages (English, Spanish, Greek, and German). Efforts will be dedicated to ensure the reusability of the developed tools outside of V4Design.

Integration: This component will also utilize the message bus in its communication with the rest of the system.

4.4 The data storage and retrieval tools

The data storage and retrieval system of the V4Design platform groups all databases, data wrappers, connectors, and other low-level data manipulation modules.

Visual content will be acquired from renowned film producers and distributors like DW, EF, AF, and SLRS, and be complemented with visual artwork that will be crawled and scrapped from public digital archives. Textual content related to the visual content of interest will be also retrieved from public databases in order to enhance the digital information that will be provided to the users.

Crawling and scraping and data wrapping modules will extract freely-available textual and visual content from open resources on the web, including social media. This data will be repurposed in the context of V4Design. Data wrappers will be developed based on existing 3rd-party APIs (e.g. twitter, Facebook) to collect content based on queries automatically formulated by machine learning techniques.

Data acquisition will be complemented with collections of paintings, pictures of other artwork and the relevant metadata from the Europeana's repository of records. This will be done mainly by accessing Europeana Collections and the Europeana API. Data selection and extraction will take into account the associated licenses (open data) in order to ensure compliance with the legal frameworks of the project.

Storage modules will store all the structured and unstructured data of the system, including raw data that has been extracted by crawling or scrapping, and provides search and retrieval capacities to the entire system. Storage modules currently include a storage for non-semantic data (3D objects, styles, textures, etc.) and another storage for the knowledge generated by semantic analysis (mainly from the semantic integration and reasoning, and the TALN language analysis services). These two storage modules are virtually independent, and each communicates with a distinct set of services. They can be hosted as two separate and independent components. For the first phase of the project, and until these modules are developed and become available as experimental prototypes, we group them under the data storage and retrieval system in order to standardize their relationship with the rest of the platform in parallel.

Data should start to be available in the system from M5 onwards.

Storage modules will send a notification to the message bus when new data is available. This broadcast will entail the activation of the analytical services that will process this new data accordingly.

5 TECHNICAL SPECIFICATIONS OF SERVICES AND COMPONENTS

The design of distributed systems is a complicated endeavour because it entails many decisions that could impact the architecture scalability and performance on the long run. Therefore, understanding the specifications of the intended architecture design is critical before engaging the design and development of its integrated components. This starts by defining component-level specifications, and then addressing specifications on the level of the platform in which components are considered and treated as black boxes.

Name	Acronym	Owner	Status	Function
V4Design REST API	REST API	NURO	Concept	Query the V4Design Asset Repository with a user defined list of filters.
KB Population service	KBPopulation	CERTH	Concept	Mapping incoming information from different components to the RDF-based representation format.
Semantic integration and Reasoning	Reasoning	CERTH	Concept	analysing the data generated by the various V4Design components
3D-Reconstruction service	3D-Reconstruction	KU	In development	Convert initial input data into 3D point clouds and meshing
TALN Language Generation	TALN-LG	TALN	In development	Generating textual reports, descriptions, or summaries, starting from extracted data.
TALN Language Analysis service	TALN-LA	TALN	In development	Analysis and capture of the natural language textual material into structured, ontological representations
NURO VR Tool	NURO VR	NURO	In development	Tool used by game designers to create virtual environments by reusing assets.
V4Design for Rhino	Rhino	McNeel	In development	Used by design professionals to produce 3d models of objects, spaces, buildings, urban environments, etc.
Spatio-Temporal building and object localization	STBOL	CERTH	In development	detect buildings and basic elements of them (i.e. type of window, door, roof, decoration, facade, etc.) from images or video frames

Aesthetic extraction and texture proposals service	AE&TP	CERTH	In development	Extract and categorize aesthetics of paintings based on their style, creators and emotion.
Data storage and retrieval system	DS	CERTH	In development	Stores all data objects pertinent to the system and is accessible from other components for querying and storage.

Table 4: Summary of V4Design architecture components

Before delving into the details of the technical specifications, we summarize the components of the architecture as previously defined in this document in Table 4.

Arguably, the most important aspect of technical specifications for these architectures is the manner by how its components will communicate and share data. In the case of V4Design, the fact that most architecture components are considered as standalone applications and that a distributed cloud-based architecture is contemplated, simplifies the requirements analysis for the system. It no longer needs to address the specific functional and non-functional specifications of the components leaving these concerns to be addressed at the level of component development, integration or adaptation. This will be explored in depth in the context of D6.2, which will detail the technical requirements of each module as well as those pertaining to the implementation of the V4Design platform. In D6.2, we will expand our definition of the functionalities that will be supported by the platform and will finalize the system architecture design that will be used for the implementation of the platform prototypes.

In the context of the implementation roadmap, we centre our analysis mainly on the following concerns:

- How will the architecture be distributed so as to accommodate all the envisioned system components?
- How the architecture will support communication between these components?
- How will data be stored, managed and shared amongst the components?
- How to define and address more generic concerns, such as performance, scalability and flexibility of the architecture?

Not all of these concerns are currently tackled, especially for components that are still in early stage, having flexible specifications that loosely depend on the outcomes of experimental prototypes and use cases. Nonetheless, the information collected so far has provided a meaningful overview of the intended components and identified several concerns that need to be addressed prior to the implementation and deployment of the first prototype.

In the following section, we will summarize the basic and advanced specifications of the components as described by their owners.

5.1 Basic specifications of platform components

The basic specifications of the modules describe in a general sense the deployment environment, the development framework, the expected capacity and the expected availability and reliability of each component are summarized in Table 5.

Component	Deployment environment	Development Framework	Expected capacity	Expected availability and reliability
3D-Reconstruction	Windows 10, additional hardware requirements expected after testing	Visual studio 2017, C++ , C#	Very susceptible towards request image/video size and needs further experimenting	100%. If service saturates, then new requests will be put on hold
Reasoning	windows 10, 5-6GB	Java	Linear, can only process one request at a time	The service will be available at any time
KBPopulation	windows 10, 2-3GB	Java	Linear, can only process one request at a time	The service will be available at any time
AE&TP	Linux, GPU (more than 8G memory)	Python, Tensorflow, Keras	Linear, can only process one request at a time	Depending on status, is not available while executing a request
STBOL	Linux, GPU	Python, Tensorflow, Keras	Linear, can only process one request at a time	Depending on status, is not available while executing a request
REST API	Linux or Windows Server	MacOS / Windows, node.js, Swagger or Apiary	Largely dependent on the server resources.	The service will be available at any time
Rhino	Windows 10, MacOS 10.13.x	Windows 10, MacOS 10.13.x, Visual Studio 17, .NET Framework 4.5, C#, ECMAScript, CefSharp,Vue.js	Determined during functional testing, no bottleneck expected.	The service will be available at any time
NURO VR	Windows 10, MacOS 10.13.x	Windows 10, MacOS 10.13.x,	Determined during functional testing, no	The service will be available at any time

			bottleneck expected.	
TALN-LG	Docker (preferably on Linux), 5GB RAM (estimation)	Java, C++, Python, Graph-transduction grammars	Determined during functional testing, no bottleneck expected.	The service will be available at any time
TALN-LA	Docker (preferably on Linux), 10GB RAM	Java, C++, Python, Graph-transduction grammars	Determined during functional testing, no bottleneck expected.	The service will be available at any time
DS	Not yet determined	Not yet determined	Determined during functional testing, no bottleneck expected.	The service will be available at any time

Table 5: Basic specifications of components

5.2 Advanced specifications of platform components

The advanced specifications of the components address their relationship with other platform components and how they manage data. In particular, we discuss the local data storage policy, the interoperability requirements, the security model and the scalability of each component.

Component	Local Data storage	Interoperability requirements	Security model	Scalability
3D-Reconstruction	Needs agile access to data, and may store instances of data locally for processing.	N/A	Access to 3D reconstruction service is limited to only V4D services	Horizontal scaling possible if service is deployed on multiple machines
Reasoning	The service stores new knowledge in the KB (RDF triple store).	RDF model (reads and stores data in this format)	Basic authentication (username/password) can be supported	Not yet determined
KBPopulation	The service stores data in the KB (RDF triple store)	The data to be stored in the KB needs to follow the RDF data model	Basic authentication	Not yet determined

AE&TP	Stores locally and updates the aesthetics model used to build the aesthetics V4Design gallery.	N/A	No access protocol is currently envisioned.	The number of requests and the processing time are linearly dependent
STBOL	Stores locally and updates the segmentation model used to build V4Design gallery.	N/A	No access protocol is currently envisioned	The number of requests and the processing time are linearly dependent
REST API	stores authentication keys locally	Specifications for the database system and V4Design Asset Repository	Not yet determined	This service would scale horizontally and vertically
Rhino	stores retrieved assets locally if the user has implemented the asset into a Rhino model	N/A	Depends on the REST API authentication paradigm	Horizontal and vertical scaling possible
NURO VR	stores retrieved assets locally if the user has implemented the asset into a Rhino model	N/A	Depends on the REST API authentication paradigm	Horizontal and vertical scaling possible
TALN-LG	No local storage	N/A	Relies on the message bus security for communication.	Horizontal and vertical scaling possible
TALN-LA	No local storage	N/A	Relies on the message bus security for communication.	Horizontal and vertical scaling possible
DS	N/A	Data format and request format	Not yet determined	Horizontal and vertical scaling possible

Table 6: Advanced specifications of components

5.3 Platform-level technical specifications

The way the defined services are integrated into a coherent platform that provides and supports the functionalities envisioned in V4Design requires further investigation. In fact, some related concerns cannot be resolved before the development of current concepts into experimental prototypes, as shown in the previous sections. This is expected to happen

between M6 and M12, and therefore the roadmap for the development of the V4Design platform identifies current ambiguities and provides a strategy for their resolution, taking into account the major technical milestones of the project, and its implementation plan. This is discussed in the following.

Ideally, the internal organization of the V4Design architecture follows a well-defined pattern that separates between user-oriented tools, middleware components, services, and data storage and retrieval. This organization helps to standardize the way the platform is integrated and expanded in the future to accommodate more services or replace services with more advanced ones. This standardization helps to guarantee a healthy evolution of the platform towards a market-ready enterprise application.

In the following diagram (Figure 7), we show an envisioned conceptual design that implements this standardization.

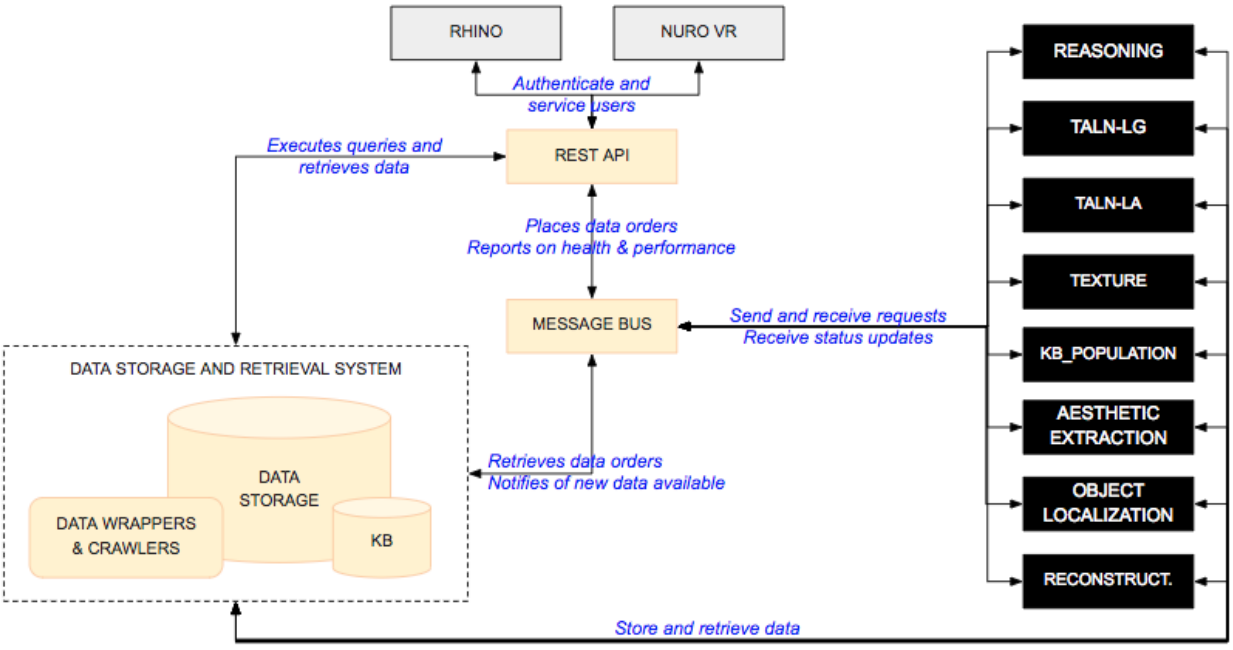

Figure 7: V4Design envisioned conceptual design

5.3.1 Data management and storage concerns

Data is a critical aspect of the V4Design platform, which aims to support and process different types of data objects in different degrees of construction, from the basic raw data to more complex and composite objects that respond to the needs of the envisioned user experience.

Therefore, we will accord a special attention to the data management and storage policy in the platform, taking into account the expected level of maturity required at the end of the project. For experimental purposes and in the context of early prototypes, direct integration between services on the one side and data storage and retrieval components on the other would be expected.

As the platform development progresses, we expect to migrate into a more centralized architecture model in which local storage of data is reduced, and communication between services and global data storage is standardized and mediated by the message bus. In this mature model, components and services will maintain direct access to the data (as data will not be channelled through the platform message bus), but will do so in an organized way that allows monitoring and control to the extent required.

5.3.2 Local storage policy

Components and services of the V4Design platform can store data locally according to the following requirements:

- **Local Data**: data only relevant to the component and not required nor accessed by other platform components and services.
- **Data Queues**: services and components can implement data queues if needed to streamline the data streams locally.
- **Control Data**: including logs, buffers or any metadata storage to support local functionalities and improve performance.

The platform components and services should take care not to safeguard any data pertaining to the platform locally, longer than needed, for data security and access concerns, as well as to support an overall centralized model of data storage and retrieval for the platform.

5.3.3 Data access and querying policy

The data storage and retrieval system will be accessible to all authenticated platform components, which can execute queries on the data, and store the output of their local processes without intervention from other components.

For instance, the responsibilities of the V4Design REST API component include orchestrating data queries and retrieval according to the users' input. It connects with the data storage and retrieval system, and acquires the data pertinent to the user query. This data may be distributed among different components of the data storage and retrieval system, which will not support composite queries. Instead, the V4Design REST API will query each of the data storage components separately (e.g. storage of 3D objects, and Knowledge Base).

5.3.4 Availability and scalability of services

The services of the V4Design platform are designed to work independently of the user-oriented functions. These are encapsulated by Rhino and NURO VR and mainly access and retrieve data from the system. They do not trigger any service explicitly, instead the services work in an ad-hoc manner, preparing the data and analysing it as it becomes available. Therefore, in principle, there is a disjunction between the services and the user tools. However, as the platform scales and its users multiply, we expect the data needs to follow a similar trend, and in this context the availability of services becomes relevant. In addition, we do not wish to disregard a tighter relation between the user tools and the services, since it is possible to consider a dynamic processing of data (under user request) in the future.

For these reasons, and due to the fact that several services have limited capacity as envisioned (can only process one request at a time, and are unavailable during processing), we recommend the following items as part of the platform specifications for availability:

- Each service with limited availability, or with a low saturation threshold, should implement a queue to receive and store incoming requests in order to process them sequentially.
- The scalability of these services should be addressed as part of the development efforts for its final version. This could include the introduction of (or at least support for) parallel processing or the deployment of several instances of the service.

5.3.5 **Platform security concerns**

Security concerns are an important aspect of enterprise applications, especially in distributed architecture. The platform has to insure the security of the data it acquires and generates, in addition to protecting the user data. There are several levels in which security should be addressed. First, on the level of individual components; second, on the level of the servers that constitute the distributed architecture; third, on the level of the architecture middleware, namely the message bus; and fourth, on the level of the user tools.

- **Component-level security**: the security policy adopted by each component and service should be defined and included in its requirements and specifications, and documented in D6.2 for reference. The objectives of such policy would centre on preventing data leaks, and unwanted and uncertified access to the component. Special care should be accorded when integrating or connecting with third-party applications, and in choosing the adequate development framework. In addition, security should be one of the determining factors in choosing libraries and external components for the development.
- **Server-level security**: The security of the servers integrated in the platform architecture should be addressed early on in the development process. Access to the servers should be controlled, and the servers protected according to common practices. This includes addressing risks of hacks, denial-of-service attacks, and other common security hurdles. Earlier versions of the servers could address security lightly, but final versions should contemplate security in a more comprehensive manner.
- **Architecture-level security**: in addition to dealing with the security concerns on the component and server levels, the architecture middleware should authenticate the services and monitor their health in a bid to add an extra layer of security to the entire platform. Platform security can be centralized to a large extend, alleviating service authentication and trust concerns.
- **User-tools security**: the user tools represent the user interface of the platform, and therefore should be responsible for addressing user-oriented security concerns. These mainly include the authentication of users and the prevention of malicious behaviour, which will be addressed by these tools.

These security aspects will be addressed in the development and integration of the V4Design platform in order to insure the level of service quality, stability and performance expected at TRL6-7.

5.4 Communication and messaging

As early concepts, most of the services do not yet envision organizing their interaction with the rest of the platform component through the platform middleware in a complete manner. The integration between all of the platform's components and the message bus will be considered in order to achieve the required level of security, consistency, stability, and service quality associated with TRL6-7 levels, implying a healthy performance in a real-world environment without customization according to the environment's characteristics (or implemented use case).

In Table 7, we compile the current message interaction of the platform, listing the messages defined that the services send and receive through the message bus. These will be expanded further and completed before the integration of the first prototype architecture.

Sender	Event name	Description	Message type / topic	Message receiver(s)
3D-Reconstruction	Photogrammetry Update	Update towards photogrammetry object (new, pointcloud completed, mesh completed, etc.)	Photogrammetry Update, Any service requiring 3D models	Broadcast
	setData	remote method to upload data to the KB	analysis results from various modules	KBPopulation
DS	New data available	When new data is stored in the DB, the message bus notify Localization to update the outdoors and indoors models that it already contains.	New data available	Localization
KBPopulation	startReasoning	Data Uploaded from the KB Population service, and request sent to Reasoning to process it. This notification is sent when the service finishes the mapping and uploading of data in the KB.	Task request	Reasoning

Reasoning	Reasoning Finished	This notification is sent when the service finishes the reasoning task	Task completed	*Broadcast*
	Photogrammetry Verify	Reference to input data (format to be discussed -> Json, xml, etc.)	Unsuitable data -> "unsuitable" message Suitable data -> handle/path to 1 or more Photogrammetry Object	Reconstruction
	Process PointCloud	Photogrammetry Object	Progress report (Json, xml, etc.)	Reconstruction
	ProcessMesh	Photogrammetry Object (with PointCloud)	Progress report (Json, xml, etc.)	Reconstruction
	Photogrammetry Request	Photogrammetry Object		Reconstruction
	Single Image Reconstruction	Input image data	Output Summary with regards to extracted material (Json)	Reconstruction
Reasoning	Generation requested	Request for summary/report/… generation	Task request	TALN-LG
TALN-LG	Generate text request	Request for summary/report/… generation	Task completed	Reasoning
Reasoning	Analyse text request	New content available for analysis	Task request	TALN-LA
TALN-LA	Analyse text request	Text analysis is completed, results are stored in the database	Task completed	Reasoning
DS	Data Storage	Stores all types of data in the system, composed of different specialized modules, each addressing a specific type of data	New data available	*Broadcast*

Table 7: Main exchanged messages in the system

5.5 Development timeline

The development timeline of these services and components is detailed in the project implementation plan. We compile this information as a reference in the following diagram (Figure 8) that reveals the development track of each component and service.

							MS2								MS3							MS4							MS5
	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
Rhino							V1						V2								V3							V4			
Nuro							V1						V2								V3							V4			
V4Design REST API							V1						V2								V3							V4			
KB Population	prototype						Basic mapping								V1								V2							V3	
Reasoning	prototype						Basic reasoning								V1								V2							V3	
Localization											V1																	V2			
Aesthetic	prototype									V1																	V2				
Texture									V1																V2						
Reconstruction															V1																
TALN-LG							prototype				V1																	V2			
TALN-LA							prototype						V1														V2				
Data storage & retrieval	prototype						V1						V2								V3						V4				

Figure 8: Development timeline

In addition, the project contemplates the development and execution of four pilot use cases (PUCs). The roadmap for the implementation of the V4Design platform does not address the peculiarities of the PUCs directly, instead their requirements and planification is addressed in the project plan and in the context of WP7 (user requirements and evaluation). Instead, we use these four PUCs are a reference, taking into account the development cycles in addressing their needs and requirements. The following are the four PUCs accounted for in V4Design:

- PUC1: Architectural design, related to existing or historical buildings and their environments
- PUC2: Architectural design, related to artworks, historic or stylistic elements
- PUC3: Design of virtual environments, related to TV series and VR video games
- PUC4: Design of virtual environments, related to actual news for VR (re-) living the date

On the level of V4Design platform development and integration, there are the following four major milestones to consider, as they correlate with the deployment and delivery of the PUCs:

- **MS2**: Operational prototype (M12) - Setup of the operational infrastructure or the first version of the architecture. No PUC is related to this milestone.
- **MS3**: 1st prototype and evaluation (M20) - Completion of the first development cycle, which will be utilized for PUC1, PUC2, and PUC3.
- **MS4**: 2nd prototype (M28) - Completing of the second development cycle, which will be utilized for PUC2, PUC3, and PUC4.
- **MS5**: Final system (M36) - Completion of the third and final development cycle, which will be utilized for all PUCs.

5.6 Required resources

During our analysis of requirements and of the architecture and integration models, we have derived several concerns related to the resources necessary for the development of the V4Design platform. These include data resources, servers, encoding and decoding libraries, and code repository. They are discussed in the following.

Data: non-semantic and semantic data sets to be used as benchmark and as a material to support development, testing and evaluation of the first prototypes. The availability of data is essential for the successful development of all services, and therefore special care should be taken into account so as to provide the required data in time. In addition, the datasets provided should be sufficiently large and diversified as to cover different scenarios and situations, representing accurately the type of data that the services will process when deployed in real-world environments.

Servers: each component requires its own server to run independently from the other components, given the distributed aspect of the architecture. However, given the technical specifications and relationship between different services, several services can be deployed on a single server, if this improves the system efficiency. For instance, the Reasoning and KBPopulation services, or TALN-LA and TALN-LG, or the Texture, Aesthetics and Localization services. Overall, the minimum number of servers required for the platform is eight servers. This is explained in the following table (Table 8: Overview of server requirements):

Message bus and logging mechanisms	Constitutes the middleware server and acts as the central authority of the system. Would contain component authentication functions and logging mechanisms, in addition to the message bus.
TALN-LG and TALN-LA	Both services can be hosted on a single server since they share the same system requirements and work with the same data.
AE&TP and STBOL	3D analysis services can be deployed on a single server. Each service can process a single request at a time; therefore deploying them on a single server should not affect their performance.
Rhino, NURO VR and REST API	Deployed on a single server designed to provide access to users and support their interaction with the system. Standalone versions of the tools could be considered, and would connect remotely to the REST API server.
Crawlers and data wrappers	Kept on a separate server from the data warehouse and storage for performance considerations
Data storage server	The semantic and non-semantic data storages can be hosted on a single “data service” server. The server architecture can be chosen in a way that takes into account requirements in performance and scalability of

	data-related functionalities.
Reasoning and KBPopulation	Both services can be hosted on a single server since they share the same system requirements and work with the same data.
3D-Reconstruction service	The reconstruction service's technical requirements and the type of data it processes imply that it should be hosted on its own server, since it is not fully compatible with other configurations.

Table 8: Overview of server requirements

Encoding and decoding Libraries: encoding and decoding messages from the message bus is a task pertinent to the system components that use the message bus to communicate with other components. Content encoding follows the CAP protocol including the descriptions of data objects associated to the message and the communication and routing encoding AMQP protocol. While many libraries for encoding in AMQP are available for different development languages and frameworks (e.g. see activemq.apache.org/cross-language-clients.html), few tools are available for CAP, and those available may not be sufficiently mature or compatible with some of the development languages used. Therefore, in the context of V4Design, developers of services and platform component would assume the development of such encoding-decoding mechanisms. Guidelines for this task will be provided in writing by the developers of the message bus and the coordinators of the platform integration (McNeel).

Shared code repository: a code repository is required to facilitate the sharing of code, libraries, and programs that are common to several implemented components in order to facilitate development and standardize the way common technical topics are addressed. This could include queues and buffers applications, data encoding and decoding (e.g. supporting RDF format encoding), component authentication, and message encoding and decoding libraries. This will be provided by the project coordinator.

It is worth noting that at this stage in the project we do not expect any deviations in human resources requirements. However, if this were to occur, it will be managed by the project coordinator under WP1.

6 AN ENTERPRISE BUS SOLUTION FOR V4DESIGN

In technical and system architecture literature the expression “message bus” is used to describe different things albeit all related in supporting messaging among the system components. Essentially, a message bus is a message broker that translates messages written in a formal messaging protocol from architecture components to others, they are one fundamental building block for message-oriented middleware in distributed architectures.

Beyond the fundamental nature of message brokers by which a message bus is a system that supports message exchange patterns such as publish-subscribe, the message bus is also responsible for routing the messages to their relevant components. In fact, according to Hohpe [2], a bus contains a router that manages message distribution and uses a canonical data model to which other applications or architecture components can use through adapters, unlike simple message brokers.

When message buses are deployed as part of an enterprise application integration by which different autonomous and distributed services are bound into a single application, the message bus becomes better known as Enterprise Service Bus, and assumes even greater responsibilities in establishing and maintaining communication and information exchange among the system components. Enterprise Application Integration (EAI) models vary largely among systems, and can be classified according to their architecture, from the most vertical to the most horizontal. Vertical approaches rely heavily on middleware solutions, including protocols and APIs, and are often custom-tailored to befit a specific application. Vertical solutions tend to be time-consuming to implement and difficult to scale, edit, update or debug, especially for large complex systems [3, 4]. In these cases, another more horizontal architecture design, the “Enterprise Service Bus (ESB)” offers a better alternative as discussed in section 1.2.

In effect, the ESB bus is a service-oriented solution that can integrate different applications developed with disjoint or incompatible technologies, having different (even incompatible) formats and communication protocols. Therefore, it can assume the roles of traducing from one protocol to another, resolving contention issues between services, controlling deployment and actualization of services, and sometimes extending to aspects of data management such as data transformation and mapping, among others.

That is why a growing number of applications are built on the basis of architectural design integration bus ESB. ESB bus is a service-oriented platform for connecting applications created basing on various technologies, incompatible formats, data resources, and communication protocols. The advantage of this solution is primarily its dynamic conversion and data transformation (dynamic data transformation and conversion), distributed communication and intelligent routing services.

Hohpe et al. have surveyed and studied enterprise integration patterns extensively and created a list of the most common patterns are a reference point for new enterprise application integration projects [2]. They catalogued and described 65 integration patterns (see Figure 9) to provide a technology-independent design guidance and align similar underlying concepts from different frameworks. We will use these patterns as a general guideline for isolating the concerns related to V4Design architecture from those that are

irrelevant, simplifying the task of implementing the integration, and planning the different versions of the architecture.

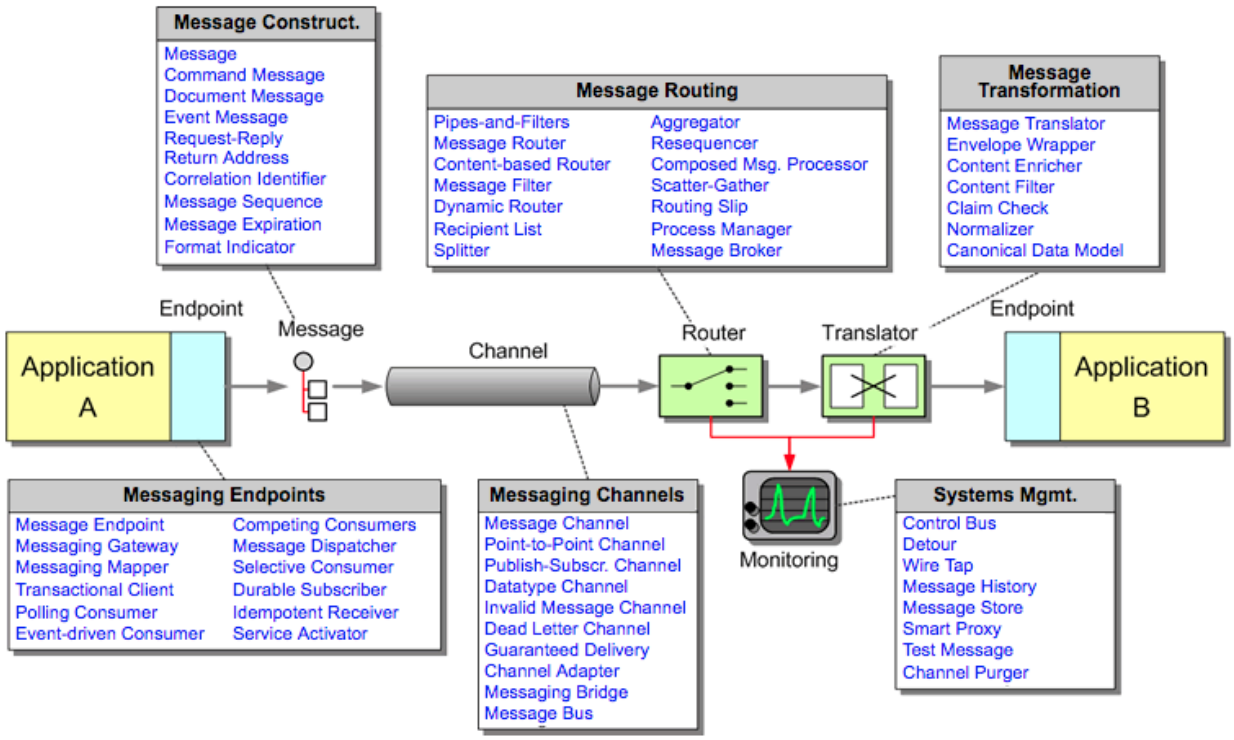

Figure 9: Most common/popular classes of enterprise application integration (EAI) patterns [2]

6.1 Functionalities of message bus solutions

In chapter 1, we introduced Enterprise Application Integration as a field of practice, and discussed its relevance to V4Design, which requires a middleware solution to connect all its components. The role of this middleware was discussed in details in chapter 2, where V4Design architecture model was introduced. In the current chapter, we discuss specific solutions for message bus middleware. We first discuss the general functionalities of the message buses and distinguish the functionalities relevant to V4Design from those that are irrelevant. Next, we introduce and compare existing solutions. Overall, there are many types of message bus solutions; some are domain-specific and other generic. Some are oriented to complex architectures others to more simple application integration cases. Additionally, some are open-source and others are legacy solutions, and some implement specific protocols while others offer more flexibility. Commercial solutions tend to integrate as many features as possible, whereas freely available solutions under non-commercial licenses tend to be simpler. Therefore, we conduct a walkthrough of the different aspects that may differ between message bus solutions, contrasting this analysis with the V4Design overall requirements and case specifics.

Message bus functionalities or responsibilities can be grouped into the following four major concerns:

Connectivity: establishing and standardizing secure communication between the application services or components. This includes the choice/definition of the supported communication protocols, including the messaging protocol, and the messaging paradigm (synchronous or

asynchronous). Synchronous messaging involves a client that waits for the application to respond to a message. In other words, in synchronous messaging, messages flow in both directions as a two-way communication, i.e. sender sends a message to receiver and receiver receives this message and gives reply to the sender. Sender will not send another message until it gets a reply from receiver. By contrast, clients in asynchronous messaging do not wait for a message response, and use a one-way communication paradigm.

It is well known that asynchronous flows are a preferred solution in many applications integration cases, including in the case of V4Design architecture. There are several advantages to asynchronous messaging systems e.g., they are more flexible and support a much higher availability of services. In addition, they are more stable since a failure of shutdown in a given component does not affect the overall system. When using such paradigm, a special attention is accorded to performance, especially in the case of user-driven processes. This is achieved by understanding the response time of services and engineering communication and platform-level processes accordingly, and by taking advantage of the parallelism in the messaging that asynchronous messaging systems support. In addition, message tracking and routing is key since messages may not be delivered or can get lost without notification.

Routing messages: Most of the message routing patterns are fundamental aspects of any integration solution, for instance the message broker, message router, and message filter patterns. However, some are complex, such as the composed message processor or the scatter-gather patterns. These are usually not common, and reserved for highly complex application integrations that include a large number of modules and services, each with its own complexities. From a requirements viewpoint, it is important to isolate the patterns relevant to the integration in order to facilitate the choice of the messaging solution to implement. In V4Design, especially for the first version of the integrated architecture, complex patterns will be avoided as much as possible, and the message bus solution will concentrate on the most fundamental aspects of messaging unless explicitly specified by the requirements analysis. In all cases, no exception or requirement is expected to entail the use of complex messaging and routing patterns given the scope of the V4Design architecture.

Data transformation: consists of the concerns related to filtering, wrapping, translating, and conducting other types of data operations with the data associated to the messages on the level of the message bus. Traditionally, these concerns are too related to the specifics of the implementation case and cannot be easily abstracted. They typically depend on the manner that the integration deals with data exchange, and the data models and the data management policies of each integrated service or component. The more these services share similarities in terms of data definition and schema, the less is the need for centralized data transformation capacities. One direct manner to deal with data transformation, especially for small to midsize applications like the V4Design platform, is to deal with such transformations on the client side instead of the middleware side.

One important decision related to data transformation in the context of Application Enterprise Integration is the use of canonical data models. These models thrive to contain all the data from the connecting data models, and therefore ensure that transforming data from a given model to the canonical data model format is always possible, allowing to streamline data exchange between different services. Therefore, the use of a canonical data model will reduce the need for a centralized data translation service and would facilitate

such translation on the level of the services instead, which would in turn facilitate the integration, increase the performance and decrease the complexity of the adopted middleware solution.

Orchestration: the orchestration role that message bus plays in the architecture is a concern that groups a higher level of functionalities than those usually associated with simple message bus solutions (connectivity, routing, and data transformation). In fact, we distinguish between technical orchestration that attends to the technical concerns of complex routing and errors management, and the enterprise orchestration that deals with the top-down implementation of complex business processes that the enterprise application implements. In case of technical orchestration, patterns such as pipes and filters and content-based routing are relevant, and in the case of enterprise orchestration, process manager patterns and composite message processors are more relevant.

The V4Design architecture aims to ultimately support user-oriented processes and to develop tools that provide new functionalities related to facilitating the reuse and repurposing of graphic resources by designers and architects. The centralized aspect of these user-oriented tools implies that no complex enterprise processes would be defined, and this is explicitly noted in the early concept of the architecture. In fact, the way components and services are integrated minimizes the role of the message bus in orchestrating the system processes. These processes are either autonomous or automated, or mostly based on notifications, data queries, and service-driven data transformations (including data extraction and classification). Therefore, in terms of integration of the V4Design architecture, we will centre solely on technical orchestration requirements.

Besides these four aforementioned concerns, there are several more concerns related to Enterprise Applications that are usually associated with message bus applications, such as business engines and application monitoring. These concerns are considered outside the scope of V4Design architecture because they address aspects of integration pertinent to application of much larger size and complexity. On this level, concerns of meta-integration and the overall correlation of the system processes with the business processes become more relevant.

In general, the complexity of the message bus solution tends to correlate with how close the requirements and specifications of the integrated components to standard practices and protocols are: the less distant they are the more likely we are to use a lightweight proprietary or open-source ESB solution; the more distant they are the more we need to consider custom-built ESB that itself integrates different modules stacked and connected according to the integrated application's requirements. Based in this reasoning, and in the light of the previous discussion on major message bus concerns, the types of message bus mostly compatible with the architecture of V4Design are generally simple and light, or what otherwise is known as lightweight ESBs. Compared to traditional ESBs, lightweight ESBs as their name suggests, are simplified integration solutions that focus on common needs and centre on efficiency and productivity instead of encapsulating features and functionalities.

6.2 Functional aspects supported by the V4Design message bus

Based on the early concept of the V4Design architecture and the description of the services that it will integrate, we have defined a set of functionalities that the message bus should

support. In addition, we have disregarded explicitly some functionality that are either are delegated to the services or are deemed outside the scope of the V4Design platform, and platform integration. Both are discussed in the following.

6.2.1 Supported functional aspects

In connectivity:

A) **Security and exception handling**: The message bus centralized role (use of a single centralized bus) in establishing and maintaining communication among the platform components reduces security risks but does not eliminate them completely. The message bus will implement a protocol-specific authentication and authorization scheme for the components to reduce further security risks. In addition, logging messages coming through the message and other practices will account for the same objective. Given than security will also be managed on a component level, there is no explicit need for using Security Decision and Security Enforcement services (SDS and SES).

In routing messages:

A) **Routing messages between components**: The message bus will take charge or routing all messages between components of the platform architecture.

B) **Monitoring and control of message routing**: The bus traffic will be logged and the routing system monitored passively [5]. Passive monitoring does not inject messages into the bus or modify on-going traffic; it collects information about transient messages as they reach the bus. The concerns addressed by this monitoring are: routing errors, protocol mixes, and message rates. Low-level concerns such as timings and transmission accuracy/errors are not addressed, as they are only relevant to the network architecture where the platform will be deployed, and therefore will be addressed in this context rather than on the level of the message bus. To analyse the requirements of the message bus, we will assume a separation of concerns with network performance issues. We note that the information collected for the aforementioned purposes will not be analysed in real time.

C) **Sequencing and queuing of messages:** Sequencing and time stamping are standard features of any message bus system. In some basic cases, the sequence number and the timestamp are the same parameter, but in most elaborate cases they differ and are employed for different purposes. The sequence number is generally used as a unique identifier for the message, and is assigned by the bus, while the timestamp is generally assigned by the component that generates the message. By default, the message bus will apply a FIFO (First-In-First-Out) queuing protocol, but will be able to support different policies in the case where it is required by the functional requirements.

In orchestration:

A) **Resolving competition between communicating components**: the message bus will take charge of resolving competitive requests or potential resource contention conflicts. The message bus will utilize a queuing strategy to respond to situations in which a component is experiencing on-going contention (oversubscribed).

6.2.2 Unsupported or irrelevant functional aspects for V4Design

Taking into account the envisioned design of the platform architecture, the scalability of the architecture is not a direct concern for integration. In other words, we will not account for multiple instances of the same service, and we assume that each service will be hosted on a single given server or end-point. In case parallel instances of a given service are contemplated, then the load-balancing between these instances is not a concern of the message bus, which will treat the service as a single end-point, integrating with the load balancer. In addition, and based on the same design, security risks will mainly be mitigated at service-level. Using a centralized message bus reduces security risks, and the message bus will implement protocols for authentication and authorization. However, each service should account for its own security policy and protocols.

In data transformation:

A) **Data transformation, mapping and transmission**: data will not be routed through the message bus, nor will the message bus monitor data read/write operations or the correct execution of queries. The message bus will propagate messages announcing the availability/unavailability of data on the behalf of the data service, but will not channel data. Data transmission will be address through public URI addressed associated to data objects by the data storage and retrieval systems.

B) **Protocol conversion and protocol validation**: the selected protocol for message communication among the platform components is the CAP V1.2, which will dictate the format and type of messages implemented in the system. The message bus will respond with an error message when the message header is not formatted according to the protocol definitions, and otherwise will route the message to its destination.

In orchestration:

A) **Marshal use of redundant services**: since the components of the platform will be deployed as online independent server architectures, each component will be responsible for providing a scalable service that can accommodate the expected load. The scalability of the platform therefore rests on the ability of its service components to scale.

B) **Controlled deployment and versioning of services**: As independent services, the architecture components will be responsible for their own deployment control and versioning of service. Upon deployment and activation they authenticate and register with the message bus, which will follow up on service availability.

In the following, we discuss some of the popular message bus solutions that are currently available in the market, and choose the most adequate solution for V4Design, taking into account the general requirements of the V4Design architecture and the specifications of the services it integrates.

6.3 Popular and relevant message bus solutions

Many tools are currently available to bind data and link systems together. Open source message bus solutions differ from classic ESB software. Open source solutions are essentially more modern solutions, lightweight and flexible (deployable in different configurations)

mainly because they delegate more tasks to the endpoints (e.g. message transformation), reliable transportation, and long-running orchestration between endpoints. In this way, the system does not cede control to a centralized bus, but rather uses the message bus to transport opaque data between endpoints. This allows the message bus to become more efficient at ingesting and routing data (and large quantities of data), and the latest versions are becoming very good at it.

We have conducted a comparative analysis of the different solutions for message bus currently available, and meeting a general criterion for stability, reliability, availability, ease of implementation, customization, scalability, extensibility, compatibility with messaging protocols, among other concerns. The aim was to isolate potential candidate solutions compatible with our understanding of the overall requirements of the V4Design platform. Consequently, the identified candidates were compared in more details and evaluated according to the scope of integration in V4Design.

In the following, we discuss these identified solutions (Table 9). Each described in terms of its license, development language, client language, required operating system, and compatibility with messaging protocols. We start by a detailed overview of open-source solutions for their high relevance to V4Design framework, and then follow with an overview of legacy-based (proprietary) solutions.

Solution	License	Dev. Language	Client Languages	OS	Protocol Compatibility
Apache Kafka	Apache license 2.0	Scala	Kafka script	Cross-platform	Kafka protocol
Apache ActiveMQ	Apache License 2.0	Java with JMS, REST and WebSocket interfaces	many, including C/C++, Python, NodeJS, Java, etc.	Cross-platform	AMQP, MQTT, OpenWire, STOMP
Apache Distributed Log	Apache License 2.0	Java	DistributedLog core library	Cross-platform	Flexible
RabbitMQ	Mozilla Public License	Erlang	mainly Java, .NET Framework, and Erlang	Cross-platform	AMQP, STOMP, MQTT, HTTP
NATS	MIT	Go	major programming languages	Cross-platform	NATS protocol
GoogleRPC	Open Source License	3 implementations (C99, Java, Go). Uses Protocol Buffers as IDL	many, including C/C++, Python, NodeJS, Java, etc.	Cross-platform	Flexible

Table 9: Selected examples of message bus solutions

6.3.1 **Open-source message bus solutions**

There are many advantages for using open-source solutions in enterprise applications, including integration. For a start, open-source solutions are generally more reliable from a product perspective because they liberate the product from dependence on external providers, vendors, or legacy technological frameworks, which in case of innovative projects can sometimes become a real barrier for progress. Open-source solutions are highly flexible and agile because they offer many alternatives to solve problems and generally allow casting complex problems into a composition of smaller more manageable problems. Furthermore, a great advantage of open-source solutions is the community-produced and validated a set of pre-existing solutions for typical and common problems, allowing for a speedy implementation. Finally, open-source solutions usually allows to build small basic versions and scale the project from there in an iterative manner, and allowing iterative testing and evaluation to achieve a highly reliable outcome. The following five message bus solutions are among the most popular in the market today, and among the more ubiquitous.

Apache Kafka: Originally designed by LinkedIn and now is part of the Apache project, Kafka was developed under the influence of persistent requirements in Enterprise Application Integration. Kafka stores the messages in flat files that are queries by offset, or in other words, Kafka clusters the publishing of messages and requires the consumer to track subscription. This simple approach and agnostic attitude towards its clients allows Kafka to perform very efficiently while maintaining a low consumption of resources. Its message management system doubles as a message queue and a log that can store messages for prolonged periods of time without affecting performance. On the other hand, Kafka lacks many features in comparison with other message bus solutions.

Apache ActiveMQ: it is the most popular open source message broker with the largest distribution network. It implements the Java Message Service specification and offers numerous features, including an excellent support for many of the Enterprise Integration Patterns. A highlight for ActiveMQ is its flexibility in supporting clustering and distribution by which several brokers can be connected into a network of brokers.

Kafka and ActiveMQ may share many features but they were originally designed for different purposes. Kafka is a distributed streaming platform with a very good horizontal scaling capability and ideal for big data processing of small messages. It allows applications to process and re-process streamed data on disk with a high throughput, and therefore it is commonly used for real-time data streaming. By contrast, ActiveMQ is a general-purpose message broker that supports several messaging protocols such as AMQP, STOMP, and MQTT. In general, it is mainly used for integration between applications/services especially in a Service Oriented Architecture, and is more ideal for enterprise messaging.

DistributedLog: it is essentially a replicated log stream stores solution. Its key abstract element is a continuous replicated log stream. All the records of a log stream are sequenced by the stream owner, and the reader can read the stream starting from any sequence number. DistributedLog is currently being merged with BookKeeper, an enterprise-grade storage system designed to provide durability, consistency, and low latency. It was originally developed at Yahoo! Research as a high availability (HA) solution for the Hadoop Distributed File System (HDFS). BookKeeper is widely adopted by enterprises like Twitter, Yahoo and Salesforce to store and serve mission-critical data and supporting different use cases. The

merger of DistributedLog with BookKeeper implies that support and improvement of DistributedLog as a standalone application is improbable. However, the functionalities of BookKeeper could be relevant to the V4Design platform on the long term, albeit outside of the scope of the project. A comparison between Kafka and DistributedLog can be seen in Figure 10.

RabbitMQ: The RabbitMQ broker was created by the functional language Erlang, which is especially suited for distributed applications because of its high-level support for concurrency and availability. The core implementation of RabbitMQ is fully compatible with the AMQP protocol, but additional protocols can be supported by integrating RabbitMQ plugins. It has an appealing web-enabled management console that facilitates the system administration, and the monitoring of the overall bus health and activities through a series of implemented indicators (e.g. number of messages per second, resource consumption, etc.).

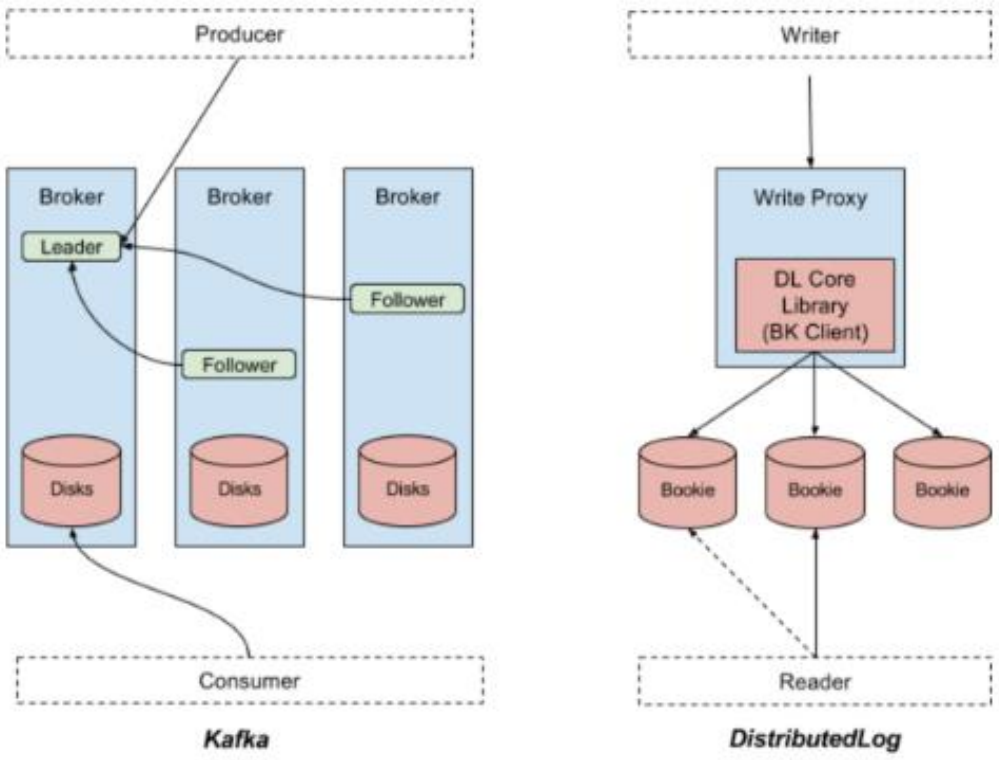

Figure 10: Comparison between Kafka and DistributedLog approaches [6]

NATS: it is a simple (even simplistic), high-performance open-source messaging solution oriented towards cloud native applications, IoT messaging, and microservices architectures. It is an example of modern messaging systems that are born from the evolution of simple service-oriented web architectures as they grew rapidly and sometimes exponentially in number of users. Instead of cramming different workflows and business processes, the approach of this type of message bus solutions centres on the performance of ingesting and routing large quantities of data. The basic solution is light, but also configurable for more complex applications.

gRPC: is a framework for implementing HTTP-based remote procedure call (RPC) services. It is designed to support the creation of highly efficient and scalable APIs and micro-services.

From a message bus perspective, it is more oriented toward technical orchestration and less so towards messaging in a classic sense. Its use of the HTTP2 standards allows it to support bidirectional streaming and to implement flow control over TCP connections, making it ideal for streaming applications and environments where synchronization between end-points is essential. TCP provides apps a way to deliver (and receive) an ordered and error-checked stream of information packets over the network. The User Datagram Protocol (UDP) is used by apps to deliver a faster stream of information by doing away with error-checking. When configuring some network hardware or software, you may need to know the difference. Finally, gRPC has its own Protocol Buffers library that facilitates the definition and integration of services.

6.3.2 **Legacy-based message bus solutions**

Proprietary message bus solutions are available almost from all vendors of Enterprise Software solutions, including Microsoft, Oracle, IBM, Amazon, and others. Almost by definition, these solutions adhere to the specifications of their legacy framework, and are primarily designed to work within a legacy ecosystem composed of several Enterprise Applications often sold as individual products. Therefore, the use of proprietary message bus solutions conditions the overall system architecture with legacy requirements, which may include the need to develop and deploy the entire platform in accordance with the chosen legacy framework.

The advantages of such solutions lay in their loyalty to industrial standards of performance, quality, and reliability. However, they are easily comparable to other open-source solutions when it comes to applications where the enterprise coordination is not required, or when only the basic integration patterns are concerned.

In the following table (Table 10), we show popular examples of legacy-based message bus solutions that are popular on the market. We choose not to include them in selecting the appropriate message bus solution for V4Design, and instead centre on open-source solutions.

Solution	License	Dev. Language	Client Languages	OS	Protocol Compatibility
Azure Service Bus	Commercial	.NET Framework Java	Java with JMS, C, PHP, Python	Azure-Cloud	AMQP
Amazon Simple Queue Service (SQS)	Commercial	Java	AWS Management Console & Java with JML	AWS-Cloud	Flexible
IBM Integration Bus	Commercial	Java, ESQL, C++, Visual Basic, and .NET	ESQL, Java, PHP, .NET	Cross-platform (AIX, HP-Itanium, Linux, Solaris, Windows, z/OS)	Flexible

Table 10: Selected examples of commercial message bus solutions

Function	Relevance	Description
Invocation	*Relevant*	support for synchronous and asynchronous transport protocols, service mapping (locating and binding)
Routing	*Slightly relevant*	addressability, static/deterministic routing, content-based routing, rules-based routing, policy-based routing
Service orchestration	*Not Relevant*	coordination of multiple services exposed as a single, aggregate service
Event processing	*Not Relevant*	event-interpretation, correlation, pattern-matching
Quality of service	*Relevant*	reliable delivery, transaction management
Adapters	*Slightly Relevant*	adapters for supporting integration with external and legacy systems
Security	*Relevant*	standardized security-model to authorize, authenticate & audit use of the ESB
Validation	*Relevant*	validation against CAP schema for sending and receiving messages
Enrichment	*Not Relevant*	enriching messages from other sources/services
Commodity Services	*Not Relevant*	commonly used functionality as shared services depending on context

Table 11: Relevance of general message bus functionalities to V4Design

In Table 11, we describe the most popular functionalities of message bus solutions as defined by their developers and distributors. Based on our previous description of the expected functionalities of V4Design message bus, we value the relevance of each aspect as a reference to facilitate the selection of the adequate solution for V4Design.

6.4 Selecting a message bus solution for V4Design

Before selecting a message bus solution, we set the general selection criteria in order to evaluate the different candidates, based on the concerns related to V4Design and the implementation of its platform.

Availability: the message bus should be constantly available and therefore the way it is deployed and connected should be as stable as possible. This includes network connectivity concerns, but also the way the message bus responds to requests in different situations.

Resilience: the capacity of the message bus to recover from difficulties is also an important factor since the services it integrates have different maturity levels (some are early prototypes), and due to the experimental aspect of the V4Design platform.

Throughput: the V4Design application will handle different types of data in different manners, creating a diversity of situations that is more typical of complex enterprise applications than small-to-midsize applications. These include heavy 3D models, movies and high-resolution images, but also light metadata and semantic information. For this reason, the throughput of the message bus is an important factor.

Lightweight: the number of Enterprise Integration Patterns relevant to V4Design is generally low and focusing on simple and basic requirements. There is no apparent need to support complex routing or processing on the level of the middleware. These functionalities are better delegated to the services. Therefore, we opt for a lightweight message bus solution that facilitates the implementation of the architecture and its maintenance, and the future expansion of its features and services.

Security: V4Design will process data and media resources, some possibly under licensing agreements; therefore security is basic concern for the entire architecture. Due to the experimental aspects of its implementation, and the integration of prototypical services that will mature along the project implementation plan, the message bus will implement some security protocols. In addition, security will also be addressed on the service level.

Ease of implementation: The optimization of available resources is important to allow flexibility in responding to changing requirements and evolving concepts in the context of the project, therefore it is important to choose a solution that facilitates the implementation of the message bus and allows flexibility in adapting it to the V4Design development environment.

Flexibility: as new requirements come to light and other requirements change in accordance with the evolving implementation of the services, and the emergence of new use scenarios and use cases, the flexibility of the message bus becomes an important factor. One way to consider flexibility is through the availability of modules, add-ons, and components that can be easily integrated in the message bus to support these requirements. Another aspect of flexibility is the adaptation of the message bus solution to the known requirements, which include the use the CAP messaging protocol.

A first evaluation of the available open-source message bus solutions shows that ActiveMQ is the most adequate solution or the most compatible with the V4Design general requirements and functional concerns. RabbitMQ is also an excellent choice, but ActiveMQ is simpler and better befitting to our requirements. RabbitMQ is easy to use and deploy but also is geared towards advanced or complex scenarios like routing, load balancing or persistent message queuing. This makes RabbitMQ less scalable and slower because the central node adds latency and message envelopes are usually quite large in comparison to ActiveMQ. In addition, RabbitMQ needs Erlang runtime environment, which is not necessary for ActiveMQ.

This choice implies that the message bus will centre on routing concerns rather than workflow and process execution, and will typically be lightweight and deployed in a configuration that supports high availability and reliance. This also implies that the integrated services will have a significant responsibility, and will not delegate control to a centralized bus, but rather use the bus as a content-agnostic transport system for data across the architecture.

6.5 Deploying the message bus in the cloud

Nowadays, with the advances in cloud-based architectures and the maturing of cloud services, Cloud Integration for distributed applications seems to be the best generic option. In fact, this approach takes advantages of Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) paradigms that facilitate the integration to a great extent, and offer additional

technical services that alleviate maintenance, management, and monitoring costs of the application.

Based on the concept of the V4Design platform, the description of its components and services, and the overall requirements of its architecture, we consider a cloud-based integration model for the following reasons:

- Using PaaS and IaaS models will reduce the overhead needed for managing low-level technical concerns, such as the management, allocation, and scalability of resources, load balancing, backup functions, security management, portability, among others.
- In the case of the message bus, third-party implementations of ActiveMQ are available and can alleviate some of the development costs without compromising the independence of the platform, as later version may emancipate by deploying locally the exact same versions of the software provided by third-party. Using such solutions will notably reduce the overhead for development, and facilitates the implementation, deployment and management of the server architecture.
- The long-term servicing and support required for the platform can be assumed by the service provider.

Despite recent advances in cloud-based architecture, security remains a concern for cloud users and is complicated by the challenge of integration. A cloud integration solution must be capable of authenticating and authorizing access to resources and services, and able to encrypt and store data. In addition, there are some concerns related to flexibility and scalability in the cloud environment for distributed applications, but this is mitigated largely by modern PaaS services. These services offer convenience and ease of use while shifting the burdens such as maintenance and upgrades to the provider. Here, the trade-off is a loss of visibility and control over some technical aspects, and in some cases a reduced ability to debug the application. Some providers include rich monitoring capabilities in order to provide the visibility and control over information flows and other performance attributes.

There are several available cloud-based solutions that implement ActiveMQ message bus, namely Amazon Web Services' Amazon MQ [7] that makes it easy to set up and operate message brokers based on ActiveMQ in the cloud. It allows direct access to the ActiveMQ console and supports most messaging protocols such as JMS, AMQP, STOMP and MQTT. In addition, it provides several add-ons and modules to support complex message bus functionalities (including transcription between messaging protocols).

6.6 Implementing the CAP messaging protocol

According to the project's plan, and in consensus with the partner owners and developers of services that will be integrated in the V4Design architecture, the messaging protocol according to which the exchanged messages will be formatted is the Common Alerting Protocol Version 1.2 [8], CAP in short hereafter.

The CAP has been originally developed as a generic format for exchanging emergency alerts and public warnings over various early-warning, disaster prevention and relief, and emergency management networks, which are complex meshes of urban and infrastructure systems, data and intelligence centres, command centres, NGOs, and others [9, 10]. The CAP protocol provides an open, non-proprietary digital message format for all types of alerts and

notifications, and it is useful beyond its application is emergency alerts and public warnings. The CAP helps to standardize event content so different and independent networked components and services can send and receive events in a common format using common and flexible conventions. The CAP standard defines the mandatory and optional fields for this type of events, and the acceptable values for each field. In short, the CAP is a consistent, complete, multilingual, and interoperable global emergency communication protocol that offers standard guidelines for developing emergency information and messages. It is adopted by states and organizations for integrating services to exchange emergency information.

Therefore, the central focus of the CAP protocol is on event format. In Enterprise Application Integration that use the CAP protocol, a layer of event processing and management usually mediates or transforms messages between the CAP standardized format and other legacy formats, which are selected according to each architecture's requirements and preferences. By nature, the CAP can also be extended to handle more operative aspects of the enterprise applications that use it as an event-encoding standard.

In practice, in emergency alerts and public warnings systems, each of the participating agencies that integrate or interface with the system usually implements its own CAP broker and event processing middleware for constructing, issuing, and processing incoming CAP messages. To further facilitate the process of adopting CAP and integrating with CAP compliant systems, a set of tools called "Common alerting protocol (CAP) alert origination tools" has been developed [11]. In addition, the CAP protocol has been extended to support flexible geographic targeting by enabling geospatial boxing using lat/lon, 3D representation, and facilities for digital images, video and audio data.

Currently there are several tools that can be useful for V4Design development in the context of the CAP protocol, including a CAP PHP library [12], a Java library from Google [13] and an online format validation tool (also from Google) [14].

6.6.1 **Formal definition of CAP standard format**

As we can see from Figure 11, the basic CAP document object model is composed of four main segments called Alert, Info, Resource and Area. The Alert segment is the main message segment and provides basic information about the message and references to other related messages. The Alert segment can be used alone for system functions such as acknowledgements and cancellations.

Usually, a CAP message includes at least one Info segment, which describes the event in details with occasional instructions for expected response. Multiple Info segments can be used in a single message to describe differing parameters, which is an interesting aspect that can be exploited in applications that work with objects having evolving states, such as animation and virtualization applications, from a V4Design perspective.

The Resource segment describes a digital asset that is related to the event information, and several Resource segments can be associated with a single Info segment. The Area segment describes the geographic area or boundaries of the event. This is usually represented by geospatial shapes (polygons and circles) and an altitude range, expressed in standard latitude / longitude / altitude terms. An event may contain more than more Area segment, similarly to Resource segments.

Being designed for alert applications, the CAP protocol is ideal for broadcasting messages to different systems, and coordinating responses from different systems nearly in real time. It is designed to normalize and aggregate data from various sources to generate graphical or visually-oriented representations, which aid in situational awareness and pattern detection/extraction applications. In the following, we show the formal CAP document object model representation.

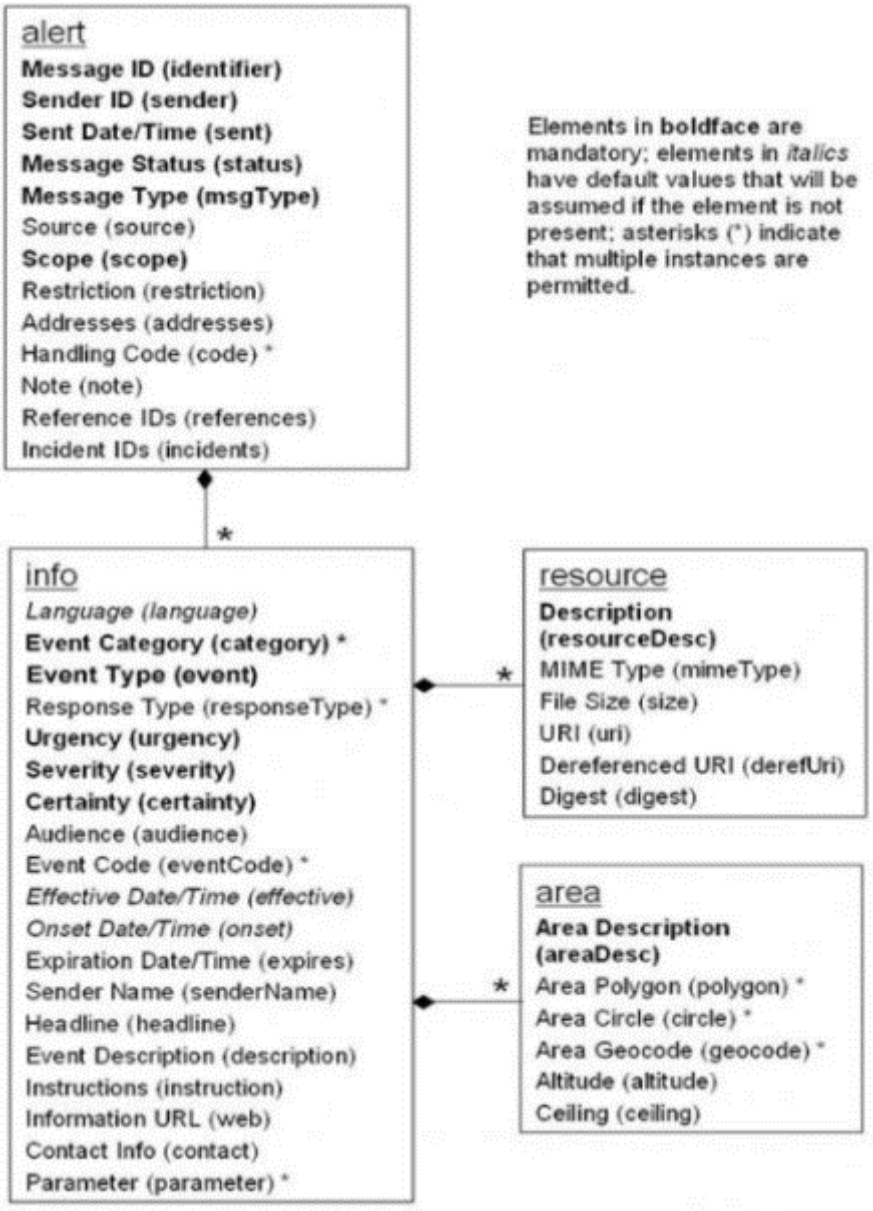

Figure 11: Formal CAP document object model representation

6.6.2 **Using CAP protocol in V4Design applications**

In essence, and from a V4Design viewpoint, the CAP protocol focuses heavily on formatting the message's content, structuring it in a flexible way according to the protocol's specifications. The CAP protocol focuses less on routing, queuing, security and other more technical concerns from an Enterprise Application Integration viewpoint. The real added value of the CAP protocol for V4Design applications is its ability to define geospatial

information in a standard way, and to associate series of digital resources to the message, making it ideal to integrate the services conceived for the platform.

Overall, and to the extent of our knowledge, there are no service-oriented enterprise-level architecture similar to that conceived under V4Design that uses the CAP protocol as its primary messaging protocol in its messaging middleware, although it is extensively used in disaster alert systems and similar applications. Even in these cases, the CAP protocol is used with complementary protocols on the level of local platforms and systems to ensure other messaging requirements not specifically covered by the CAP protocol. In addition, there is no mainstream message bus that supports the CAP protocol or lists it as one of its main supported protocols, with the exception of IBM's IBM Intelligent Operations Centre **[15]**, which is a generic software solution designed to facilitate effective supervision and coordination of operations. Existing examples of operational CAP brokers are small-scale ad-hoc projects (e.g. SAMBRO: https://sahanafoundation.org/sambro/), which do not seem adequate for reuse in the context of V4Design integration.

By contrast, other mainstream protocols used in message bus focus far more on technical aspects rather than content: for instance, the Advanced Message Queuing Protocol (AMQP) supports a wider range of functionalities that are more adequate for Enterprise Application Integration, such as flow control and message-delivery guarantees, authentication and data encryption. This wire-level protocol describes the format of data sent across the network as a stream of bytes, and therefore any system that interprets messages in this format can interoperate with any other system.

Therefore, a good option to combine the advantages of the CAP protocol with those of mainstream protocols is to use a legacy protocol for messaging and routing and other technical concerns, and encode its message content as a CAP event. Under this approach, the message bus will not decode the message content but rather treat it as a black box, and will only log message transactions but not the messages themselves. Encoding and decoding the message content will be dealt with on service-level: each service should be able to achieve this through its own message bus interface.

7 CONCLUSIONS

In order to mitigate the risks associated with the failure of scientific and technical integration of the platform components, a roadmap for the development of the V4Design platform is presented in this document. It situates the platform development and integration within the context of Enterprise Application Integration, and draws on existing models of application architecture, practices, and relevant approaches in this field.

At this early phase of the development process, nearly all components and services that would be integrated in the platform are either concepts or in the early stage of development. Therefore this roadmap provides a reference document with guidelines to insure that platform-level concerns are addressed coherently in the development of these components and services.

The roadmap presents an architecture model for the V4Design platform that is based on distributed and composite applications, in which similar components are grouped and standardized, and the communication protocols and mechanisms established and described.

As part of its analysis of high-level specifications, the roadmap includes a comparative analysis of existing middleware solutions for messaging and communication, and makes the decision to use an industrial solution that befit closely the case of V4Design and the nature of its architecture. This concludes by the selection of ActiveMQ message bus as the most adequate middleware solution for establishing and maintaining communication in a way that facilitates the construction of the platform's middleware.

The roadmap also includes a high-level description of the platform components and services as defined by their owners, and an early assessment of their requirements. These concerns will be addressed more in details in the context of deliverable D6.2 that describes the technical requirements that will be taken into account during the implementation of the platform, and describes the functionalities that will be supported, alongside the system architecture used for the implementation of the defined platform prototypes. In the context of the roadmap, we identified the main specifications of each module and discussed how to integrate them conceptually in a coherent way.

In addition, the roadmap identifies and discusses several concerns for the development of the V4Design platform, including data management, security, hosting, communication, queuing, among others. It proposes a more advanced conceptual model by which the definition of modules are standardized and grouped into four essential categories, according to their role in the platform: user tools, middleware components, services, and data storage and retrieval.

Finally, the timeline for the development of the platform is integrated based on the individual milestones of the components, and the milestones that govern the platform development cycles. The main required resources for this development are identified and discussed from a platform point of view.

8 REFERENCES

[1] GORTON, Ian; THURMAN, Dave; THOMSON, Judi. Next generation application integration: challenges and new approaches. En Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual Internat.

[2] HOHPE, Gregor; WOOLF, Bobby. Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional, 2004.

[3] Orłowski C., Ziółkowski A., Czarnecki A., Validation of an agent and ontology-based information technology assessment system, Cybernetics and Systems: An International Journal, 41 (1), 62–74, 2011.

[4] Orłowski C., Rule-based model for selecting integration technologies for Smart Cities systems // Cybernetics and Systems, 2014 – in press.

[5] Curtis, James "Passive Measurement", Jan 17, 2000. http://wand.cs.waikato.ac.nz/old/wand/publications/jamie_420/final/node9.html

[6] A technical review of Kafka and DistributedLog, Apache Foundation 2016 http://bookkeeper.apache.org/distributedlog/technical-review/2016/09/19/kafka-vs-distributedlog.html

[7] Amazon Web Services' implementation of ActiveMQ, known as Amazon MQ: https://aws.amazon.com/amazon-mq/

[8] Common Alerting Protocol Version 1.2, publish specifications. https://docs.oasis-open.org/emergency/cap/v1.2/pr03/CAP-v1.2-PR03.pdf

[9] Video "Introduction to CAP", Eliot Christian (WMO): http://www.youtube.com/watch?v=n0iKp60jjtY

[10] Video on the use of CAP in real-time biosurveillance pilot, Nuwan Waidyanatha (LIRNEasia): http://www.youtube.com/watch?v=G7WOq5giddI

[11] Description of the Common Alerting Protocol Alert Origination Tools, by the U.S. department of homeland security. https://www.dhs.gov/publication/common-alerting-protocol-alert-origination-tools

[12] A PHP Library for CAP protocol message encoding/decoding: https://github.com/AT-backbone/Cap-PHP-library

[13] A Java Library from Google for CAP protocol message encoding/decoding: https://github.com/google/cap-library

[14] An online CAP validation tool from Google: http://cap-validator.appspot.com/

[15] www.ibm.com/support/knowledgecenter/en/SS3NGB_5.1.0/ioc/kc_welcome.html