Period Covered 2014/03/01–2014/06/30

<table>
<thead>
<tr>
<th>Project ref. no.</th>
<th>FP7-ICT-610691</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project acronym</td>
<td>BRIDGET</td>
</tr>
<tr>
<td>Start date of project (duration)</td>
<td>1 November, 2013 (36 months)</td>
</tr>
<tr>
<td>Document due Date:</td>
<td>30 June 2014</td>
</tr>
<tr>
<td>Actual date of delivery</td>
<td>30 June 2014</td>
</tr>
<tr>
<td>Leader of this document</td>
<td>Diego Gibellino (Telecom Italia)</td>
</tr>
<tr>
<td>Reply to</td>
<td><a href="mailto:diego.gibellino@telecomitalia.it">diego.gibellino@telecomitalia.it</a></td>
</tr>
<tr>
<td>Document status</td>
<td>Final</td>
</tr>
</tbody>
</table>
Abstract
This document describes the initial version of the BRIDGET system architecture as well as the initial infrastructure of the system according to the list of functional requirements organised and presented in D2.2 Version A. The architecture is based on a standard component-based model for digital media services and applications: ISO/IEC MPEG-M.

Keywords
Bridget, Engine, Components, Architecture, Hybrid, Interface, API, MPEG-M, Data Model, Infrastructure

Sent to peer reviewer
24 June 2014

Peer review completed
26 June 2014

Circulated to partners
26 June 2014

Read by partners
30 June 2014

Mgt. Board approval
30 June 2014
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2014-03-12</td>
<td>Diego Gibellino – First table of contents</td>
</tr>
<tr>
<td>0.2</td>
<td>2014-04-17</td>
<td>Diego Gibellino, Gianmatteo Perrone – Introduction, Architecture design</td>
</tr>
<tr>
<td>0.3</td>
<td>2014-05-10</td>
<td>Gianmatteo Perrone, Davide Bertola, Leonardo Chiariglione - Methodologies, MPEG-M</td>
</tr>
<tr>
<td>0.4</td>
<td>2014-05-19</td>
<td>All – Review and logical architecture</td>
</tr>
<tr>
<td>0.41</td>
<td>2014-05-27</td>
<td>All – Engine descriptions</td>
</tr>
<tr>
<td>0.42</td>
<td>2014-06-10</td>
<td>All – Review, Infrastructure, Data model</td>
</tr>
<tr>
<td>0.43</td>
<td>2014-06-12</td>
<td>All – Editorial changes</td>
</tr>
<tr>
<td>0.44</td>
<td>2014-06-14</td>
<td>All – Review, updates</td>
</tr>
<tr>
<td>0.50</td>
<td>2014-06-16</td>
<td>All – Review, updates</td>
</tr>
<tr>
<td>0.6</td>
<td>2014-06-17</td>
<td>All – Editorial changes</td>
</tr>
<tr>
<td>0.7</td>
<td>2014-06-18</td>
<td>All – Engines revision</td>
</tr>
<tr>
<td>0.8</td>
<td>2014-06-21</td>
<td>Final version for peer review</td>
</tr>
<tr>
<td>0.9</td>
<td>2014-06-23</td>
<td>Typos</td>
</tr>
<tr>
<td>1.0</td>
<td>2014-06-26</td>
<td>Final review by IMT</td>
</tr>
<tr>
<td>1.1</td>
<td>2014-06-30</td>
<td>Helen Cooper – Final formatting and grammar/spelling/typos check before submission.</td>
</tr>
</tbody>
</table>
Summary

1 Executive Summary 6
2 Introduction 7
3 Glossary 8
4 Architecture Design 9
   4.1 Methodology 9
      4.1.1 Service Oriented Architectures for Media 11
   4.2 MPEG-M 12
      4.2.1 Structure of MPEG-M 13
      4.2.2 Applications 14
      4.2.3 Middleware 14
      4.2.4 Computing Platform 15
   4.3 Design Tools and APIs documentation 15
5 High Level Architecture 17
6 Broadcast/Broadband Hybrid Architectural Model 19
7 Detailed Architecture 21
   7.1 Professional Authoring Peer Logical Architecture 22
   7.2 End User Peer Logical Architecture 23
   7.3 Physical Model 25
   7.4 Data Model 25
      7.4.1 Logical Data Model 25
      7.4.2 Authoring tool Database Model 35
      7.4.3 ARAF Data Model 36
   7.5 Engine descriptions 38
      7.5.1 Media Analysis Proto Engine 38
      7.5.2 Visual Search Proto Engine 40
      7.5.3 3D reconstruction Proto Engine 42
      7.5.4 Fingerprint Extractor Engine 44
      7.5.5 BRIDGET Description Engine 47
      7.5.6 3D Compression Engine 47
      7.5.7 Synchroniser Engine 48
      7.5.8 MediaFramework Engine 50
8 BRIDGET Infrastructure 51
   8.1 Servers and Applications 51
   8.2 User Devices 55
   8.3 Network 56
9 Conclusions 58
9 References 59
Figures

Figure 1 - Architecture design process in BRIDGET ....................................................................................................... 10
Figure 2 - Architecture of an MXM Peer................................................................................................................................... 13
Figure 3 - Aggregation and Orchestration ........................................................................................................................... 15
Figure 4 - BRIDGET Architectural model .......................................................................................................................... 17
Figure 5 - BRIDGET Peer architectural layers ................................................................................................................... 18
Figure 6 - BRIDGET High Level Architecture .................................................................................................................. 18
Figure 7 – Hybrid content consumption flow ................................................................................................................... 19
Figure 8 – Synchronisation mechanism for hybrid broadcast/broadband content ......................................................... 21
Figure 9 - BRIDGET Professional Authoring Peer Logical Architecture ............................................................................... 22
Figure 10 - BRIDGET End User Peer Logical Architecture ............................................................................................... 24
Figure 11 - Bridget Description Data Model .................................................................................................................... 26
Figure 12 - Content Description Data Model .................................................................................................................... 27
Figure 13 - Content Types Data Model ............................................................................................................................... 30
Figure 14 - Bridget Presentation Data Model .................................................................................................................... 31
Figure 15 - Query By Content Data Model .......................................................................................................................... 33
Figure 16 - 3D Model Reconstruction Request Data Model ............................................................................................... 35
Figure 17 - Authoring tools database structure................................................................................................................ 36
Figure 18 - The ARAF context ................................................................................................................................................. 37
Figure 19 - Media Analysis Proto Engine Component Diagram ............................................................................................ 38
Figure 20 - Activity diagram for Media Analysis Proto Engine ............................................................................................ 39
Figure 21 - Fingerprint Extractor Engine Component Diagram ........................................................................................... 45
Figure 22 - Fingerprint Extraction Activity Diagram ........................................................................................................ 46
Figure 23 - Synchroniser Engine Component Diagram .................................................................................................. 48
Figure 24 - Activity Diagram for the FindTrigger procedure ........................................................................................................ 49
Figure 25 - Platform Layer based structure ............................................................................................................................ 51
Figure 26 - Hybrid Virtual Cluster Schematic .......................................................................................................................... 52
Figure 27 - Virtual Cluster Network Configuration .................................................................................................................. 53
Figure 28 - Software package deployment structure ........................................................................................................ 54
Figure 29 - Virtual Cluster - Nodes network visibility ........................................................................................................ 55
Figure 30 - A picture of Huawei Media Pad M1 tablet ......................................................................................................... 56
Figure 31 - BRIDGET Network Architecture ...................................................................................................................... 56
Executive Summary

The BRIDGET architecture and infrastructure are designed to support the scenarios and use cases described in Deliverable 2.1 – "First BRIDGET workshop and use scenarios – version A", and the list of functional requirements organised and presented in Deliverable 2.2 – "Requirements and Functionalities - Version A". This version of the document will drive the developments planned for the first year of the project.

The project partners selected common methodologies to contribute to the design of the high level and detailed BRIDGET architecture and main interfaces, including UML-based design tools and API documentation styles. To maximise the flexibility and reuse of the technology developed in BRIDGET, an ISO/IEC standard component-based model specifically designed to support digital media applications and services, namely MPEG-M, has been selected and adopted as a reference model. BRIDGET services and applications can be implemented using technology modules defined as MPEG-M Middleware components.

The adopted hybrid broadcast/broadband model consists of main content displayed on the main screen and delivered through a broadcast channel (e.g. DVB-T/S/C). This content is the driver for accessing and consuming related additional or enhanced content. The second screen, using a broadband connection, can retrieve and present available bridgets, and when the user selects one of them, it displays the related target content. Availability and presentation of bridgets are dependent on synchronisation points identified through audio fingerprint technologies by the second screen.

The components identified and defined by BRIDGET to support scenarios and demonstrators for Version A are:

- Media Analysis: analyses a media content producing a hierarchical temporal structure representation of the content, as well as low-level descriptors (e.g. visual content-based), high-level annotations (e.g. scene classification), and quality measures (e.g. visual or audio quality based);
- Visual Search: supports search for visual objects in large image and video libraries: analyses images and videos in the content libraries, extracts compact descriptors, builds descriptor databases and indexing schemes, ranks results;
- 3D Reconstruction: creates a 3D model from input images and/or videos;
- Synchroniser: Monitors an audio stream to detect synchronisation points;
- Fingerprint Extractor: extracts an Audio Fingerprint (AFP) from an uncompressed chunk of audio data;
- 3D Compression: produces a compressed 3D graphics data structure from a 3D model that can be efficiently transmitted, and later decompressed and rendered to screen;
- Bridget Description: provides access to bridget data structures and the associated metadata;
- MediaFramework: provides media demultiplexing and decoding functionalities according to ISO/IEC 23006 Part 2

The different components are used in the two different applications implemented for Version A: the Professional Authoring tool and the End User application.

At least a subset of these components are expected to be submitted to ISO/IEC MPEG and standardised in a possible amendment of ISO/IEC 23006 Part 2.

The BRIDGET platform infrastructure is composed of two major components: a Processing Platform and a Software Platform. The BRIDGET Processing Platform contains the processing units (physical or virtual), operating systems, network and infrastructure. It is designed to allow flexibility, interoperability, scalability and ease of deployment. In this context, we propose a cloud structure based on virtual clusters. The BRIDGET Software Platform can be defined as a Software as a Service (SaaS) and contains the software packages with the corresponding deployment and configuration rules.
2 Introduction

This document describes the initial version of the BRIDGET system architecture as well as the initial infrastructure of the system according to the list of functional requirements organised and presented in D2.2 Version A [1].

The document is organised as follows: Chapter 3 provides a list of terms used in the rest of the document, Chapter 4 provides guidance on which methodologies and tools were selected and used to design and define the architectural elements in BRIDGET and how they are documented especially focussing on the MPEG-M architecture paradigm and on the UML design tools. Chapter 5 describes the BRIDGET high level architecture, providing an overview of the architectural model implemented. Chapter 6 presents the hybrid broadcast/broadband model to deliver additional bridget content. The BRIDGET detailed architecture is presented in Chapter 7. This Chapter defines the logical and physical architecture, the data model, and the main engines of the system, including interactions with other engines and related data flows; in Chapter 8, the BRIDGET infrastructure elements are described, focusing on client-side and server-side aspects.

Chapter 9 draws the conclusions from this version of the document and describes the plans for the next steps (Version B), while Chapter 10 provides a list of references mentioned in the document.
### 3 Glossary

**Table 1 - Terms and definitions**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Compression TE</td>
<td>An MXM Technology Engine that produces a compressed 3D graphics data structure from a 3D model that can be efficiently transmitted, and later decompressed and rendered to screen</td>
</tr>
<tr>
<td>3D Reconstruction TE</td>
<td>An MXM Technology Engine that creates a 3D model from input images and/or videos</td>
</tr>
<tr>
<td>Application</td>
<td>Software implementing functionalities using the High Level API</td>
</tr>
<tr>
<td>Application Programming Interface</td>
<td>A software interface specifying how a program module interacts with other program modules</td>
</tr>
<tr>
<td>BRIDGET Description TE</td>
<td>An MXM Technology Engine that provides access to bridget data structures and the associated metadata</td>
</tr>
<tr>
<td>Computing Platform</td>
<td>The component of an End System that executes software and provides such functionalities as execution and network access</td>
</tr>
<tr>
<td>Content</td>
<td>A Resource or the corresponding Resource Information</td>
</tr>
<tr>
<td>Engine</td>
<td>An organised set of technologies communicating with other Engines and Applications via API</td>
</tr>
<tr>
<td>Proto Engine</td>
<td>A model representing a set of technologies that may be decomposed in different Engines communicating with other Engines and Applications via API. It is used during the system architecture design process.</td>
</tr>
<tr>
<td>Fingerprint Extractor TE</td>
<td>An MXM Technology Engine that extracts an Audio Fingerprint (AFP) from an uncompressed chunk of audio data</td>
</tr>
<tr>
<td>High Level API</td>
<td>Programming interfaces that offer application developers easy access to Peer and Computing Platform functionalities</td>
</tr>
<tr>
<td>Low Level API</td>
<td>Programming interfaces that offer middleware developers easy access to Computing platform including trusted execution environment and network resources</td>
</tr>
<tr>
<td>MediaFramework TE</td>
<td>An MXM Technology Engine that provides media demultiplexing and decoding functionalities according to ISO/IEC 23006 Part 2</td>
</tr>
<tr>
<td>Middleware</td>
<td>A collection of engines that support the execution of Applications via High Level API</td>
</tr>
<tr>
<td>MXM</td>
<td>MPEG Extensible Middleware</td>
</tr>
</tbody>
</table>
4 Architecture Design

This section describes the approach followed to design the BRIDGET architecture and infrastructure. Four main aspects are considered:

- The selection of a common methodology adopted by all the project partners to contribute to the design of the high level and detailed BRIDGET architecture and main interfaces
- The identification of a set of key components or technology enablers (i.e. components that alone or associated with other related components provide support for new classes of applications or services) developed within the project
- The adoption of a component-based model (i.e. the ISO/IEC MPEG-M standard) to maximise the flexibility of the system and the reuse of functionalities developed in BRIDGET
- The selection of suitable and common design tools and API documentation styles supporting both the design and the development phases

4.1 Methodology

The design of the BRIDGET architecture and infrastructure is based on principles defined by the Unified Process (UP) [2], which is based on UML.

This process is based on the following principles:

- Use case driven: the design and development is planned and organised over a list of use cases and scenarios (detailed in D2.1 [3])
- Architecture centred: the development process leads to the construction of a system architecture that allows the implementation of the functional requirements (detailed in D2.2 Version A [1])
- Iterative and incremental: the development is divided into development cycles, each adding/correcting/refining new features of the system architecture

The BRIDGET architecture and infrastructure are designed to support the scenarios and use cases described in [3] and the list of functional requirements organised and presented in [1]. This version of the document will drive the developments planned for the first year of the project.

According to the functional requirements in [1], it is critical to select an architectural design pattern that can ensure both flexibility and modularity: BRIDGET developments, though aimed at providing integrated demonstrators and applications for professional and end users, have the ultimate goal of defining an “enabling platform” useful to create new systems and services, beyond those identified in this project. Moreover some requirements may change or may be added at a later stage based on the achievements and findings of the related research activities within the project.

The general structure of the project includes a set of “technology focused” work packages running in parallel, with the mission to investigate, develop and provide algorithms and modules to be integrated in a distributed system. Key aspects are therefore, in a top-bottom approach, the definition of the main domains, the identification of the main elements, associations and interfaces and a clear definition of the logical, loosely-coupled components implementing the required functionalities.

The approach depicted in Figure 1 was followed to define the BRIDGET architecture: WP3 collected, analysed and reviewed the outputs of D2.1, D2.2 and D8.1 and asked WP4, WP5, WP6 and WP7 to provide the list of exposed functionalities and expected software modules supporting the envisaged scenarios and functional requirements. The outcome of this process was a preliminary high level architecture with the definition of main elements and interfaces of the systems and their associations. The developments in WP4, WP5, WP6 and WP7 continued in parallel providing feedback and refinements to the overall design and detailing the architecture of each module. WP8 contributed to this process by ensuring the consistency of the architectural choices with the constraints defined for demonstrators and user trials.
BRIDGET embraces the Service Oriented Architecture (SOA) architecture design pattern. A system which follows SOA principles is based on discrete pieces of software providing application functionality to other applications in form of “services”. Functionalities are exposed through stable and well defined interfaces which isolate the implementation of the functionalities from their usage by external applications. Through the definition of common interfaces, interoperability among multi-vendor components or components based on heterogeneous technologies is ensured. The internal implementation details are left to the responsibility of each group implementing the different components. In BRIDGET each component implements part of the system's functionalities that combined together allow the implementation of one or more functional requirements. Interface definitions should allow the reuse of components in many different applications and contexts, and may be extended to provide new functionalities as long as the original functions are maintained.

The BRIDGET system is designed to interact with systems already in place and used by broadcasters (e.g. the Digital Asset Management system). Interfaces between BRIDGET components and external systems may require some level of customisation depending on the systems involved. In this case adaptation layers will be defined to harmonise the BRIDGET interface messages and the custom interface exposed by the systems.

In BRIDGET, the different components can interact locally or as distributed components connected typically through the Internet.

Usually, SOA is implemented at the level of interfaces between systems or servers in distributed environments, because typically each vendor customises their internal implementations at the level of components (even if in some cases common software component frameworks such as the Component Object Model - COM - are adopted). In BRIDGET, we extend this concept, taking advantage of a standard component model defined by MPEG specifically conceived to support multimedia technologies: the Multimedia Service Platform Technologies (MPEG-M) [4]. Each technology enabler is defined as a component of the MPEG-M Middleware, maximising its reuse in services and products implemented also by third parties.

Functionalities and components defined and developed in BRIDGET may be also successfully and relatively easily included in a Software as a Service (SaaS) or even Platform as a Service (PaaS) software delivery models. These approaches, quite popular today, rely on the availability of hardware and software environments in the cloud, made available to customers over the Internet. The design of the overall architecture should support possible deployments along these models.
4.1.1 Service Oriented Architectures for Media

Modern media production and distribution naturally follow a service orientated paradigm, due to the intrinsic distributed workflows and to the trend towards flexible integration of highly re-usable processing blocks as opposed to the development of dedicated solutions. Market products are nowadays increasingly developed as integrated distributed environments, although normally conceived as vertical proprietary systems.

Several attempts have been produced recently to address the problem of applying SOA to professional media processing through an open and standardised approach. Considering the latest most meaningful results, in media generation and production the FIMS initiative (Framework for Interoperable Media Services) – a joint collaborative effort between EBU (European Broadcasting Union) and AMWA (Advanced Media Workflow Association), has recently produced a set of open specifications addressing basic blocks of media management and processing in professional environments [18].

FIMS specifications are based on a common data model and are structured as specific service definitions on top of this common model. New services are demanded through a business-orientated process, governed by an international business board. Many technology providers have now started to develop FIMS-compliant services and it is expected that the market will be heavily impacted by these new developments in the very near future.

On the experimental side, although with a clear plan to address concrete business developments, the recently concluded EC project TOSCA-MP has designed, produced and tested a full-fledged SOA environment in which automated information extraction services (such as, speech to text engines, face detectors, picture quality analysers and many others) are orchestrated by a central engine (the Metadata Production and Management Framework) and rely on a distributed storage infrastructure accessible through RESTful APIs (the Distributed Repository Framework) [19].

This final result of TOSCA-MP has been constructed following a structured approach at requirements and scenario analysis, based on the S-Cube methodology [21]. Following this first analysis phase, service descriptions have been defined through a rigorous method, starting from the formalisation of relevant media production tasks [22]. This work has been complemented with analysis of the impacts of introducing automated information extraction methods in terms of costs in typical media production settings [20]. Towards delivery, exchange of content and content distribution platform integration, MPEG-M, also known as Multimedia Service Platform Technologies (MSPT), facilitates a collection of multimedia middleware APIs and elementary services and service aggregation through which service providers can offer users innovative services by extending their technology toward the seamless integration of personal content creation and distribution, e-commerce, social networks and Internet distribution of digital media [1].

The MPEG-M suite of standards is aimed at facilitating the creation and provisioning of vastly enhanced IPTV services. This is achieved by establishing a fully-fledged service ecosystem in which developers can offer service components to the professional market, manufacturers can offer MPEG-M devices to the global consumer market and service providers can set up and launch new MPEG-M services because of the ease to design and implement MPEG-M value chains. Users can also seamlessly create, offer, search, access, pay/cash and consume MPEG-M services.

The MPEG-M suite of standards extends device capabilities with advanced features such as content generation, processing and distribution by a large number of users; easy creation of new services by combining service components of their choice; global, seamless and transparent use of services regardless of geo-location, service provider, network provider, device manufacturer and provider of payment and cashing services; diversity of user experience through easy download and installation of applications produced by a global community of developers (since all applications share the same middleware APIs); and innovative business models because of the ease to design and implement media-handling value chains whose devices interoperate because they are all based on the same set of technologies, especially MPEG technologies.
Several platforms, services, and applications are built using MPEG-M for both commercial and research purposes. Some relevant related developments are:

- **Open Connected TV (OCTV)** has been set up by the Digital Media Project (DMP). The outcome of OCTV is not a complete product or a running service, but a commercial-grade implementation of MPEG-M software that may be used direct by DMP members who implement commercial products and services. Available at: [http://octv.dmpf.org/](http://octv.dmpf.org/)

- **WimTV (Web/Internet/Mobile TV)** is a digital media ecosystem enabling diffuse trading and distribution of video content. Media moves from one user to the next with associated terms of use and payment conditions. The WimTV server implements the MPEG-M suite of standards and exposes a set of application-oriented API. Available at: [http://www.wim.tv/](http://www.wim.tv/)

- Another MPEG-M compliant implementation for secure management and distribution of multimedia content, known as MIPAMS, has been developed by DMAG-UPC [11]. This platform provides several MPEG-M Elementary Services that are accessible by third parties to further develop their own Aggregated Services. Available at: [http://dmag.ac.upc.edu/mipams](http://dmag.ac.upc.edu/mipams)

- The EC funded project ALICANTE (FP7-248652), deploys MPEG-M Elementary Services on the Service Provider side of the media handling chain to manage the distribution of content to end-user devices. It also defines additional Elementary Services for managing multimedia service offerings. Available at: [http://ict-alicante.eu/](http://ict-alicante.eu/)

- The EC funded project CONVERGENCE (FP7-257123), offers an MPEG-M based platform with additional Aggregated Services supporting publish, control, search, and content usage, where users are able to define their own policies. Thus, enabling new business models for content usage. Available at: [http://www.ict-convergence.eu/](http://www.ict-convergence.eu/)

- A collaborative music analysis, visualisation, annotation and repurposing service is also currently under development based on Sonic Visualizer integrated into MPEG-M digital media trading platform. Sonic Visualizer supports multi-track audio with volume sliders for DJ mixing and lyrics for Karaoke applications, thanks to MPEG-A: Interactive Music Application Format (IM AF), as well as chords and melody extraction, automatic audio tracks alignment and audio effects. Available at: [http://www.isophonics.net/SonicVisualiser](http://www.isophonics.net/SonicVisualiser)

### 4.2 MPEG-M

As a result of current market and standardisation trends it is envisaged that real systems based on BRIDGET technologies operating in the future media production workflows will be integrated in an ecosystem in which MPEG-M based and FIMS-based services will co-exist. However, while the latter are to be seen as mechanisms mostly enabling system integration on the professional media generation and management side, MPEG-M is orientated at construction and launch of innovative distribution and publication services, the area in which BRIDGET technologies are foreseen to be mostly influential. For both standards the threshold to utilisation is lower than for other solutions because they are international standards that can be used without discrimination by small and large companies alike.

For these reasons, and to facilitate the development of BRIDGET Players and Authoring Tools, the project has taken MPEG-M as a reference for its developments. Furthermore MXM, a part of MPEG-M, natively references a range of standards designed to support media technologies that are continuously enriched with new functionalities as well as including the possibility of specifying “southward” API e.g. for Environment (Sensors and Actuators) and Security.
4.2.1 Structure of MPEG-M

The following five standards are included in MPEG-M:

2. MPEG Extensible Middleware (MXM) API [5]
4. Elementary Services [7]
5. Service Aggregation [8]

In particular, MXM is composed of:

a. Engines that expose engine-specific API
b. Technology Engines that can be variously orchestrated to achieve specific complex functionalities
c. Protocol Engines that can be used to access functionalities not available in a devices from another device and aggregated to provide complex services
d. Different Engines can be variously reused

Figure 2 is a pictorial representation of the adopted BRIDGET Application Reference Architecture, an extension of the MPEG-M Architecture.

![Figure 2 - Architecture of an MXM Peer](image)

At the top there are Applications that access the Middleware functionalities via High Level API. HL API are a subset of the individual Engines’ API exposed to Applications.

In the middle sits the Middleware which is populated by Protocol Engines (PE) and Technology Engines (TE) each of which exposes its own API. PEs are mostly used to communicate with the Middleware of other Peers, while TEs execute locally available functions. Engine API are accessible to Applications but more commonly Applications utilise the Aggregation (of Protocol Engines) functionality and an Orchestration (of Technology Engines) Engine.

Most Engines and their APIs are standard and defined in [5]. Some BRIDGET-defined Engines will be defined within the project and submitted for standardisation (the process has already started, see D3 1 - BRIDGET System Architecture and Interfaces - Version A.docx 13/59
e.g.[9]) and the reference software will be released as Open Source Software with an industry-friendly BSD licence [10].

Middleware communicates with the Computing Platform via Low Level API (LL API). These are grouped as follows

1. **Local resources API**: platform (device) specific
2. **Network API**: to connect to a network
3. **Security API**: to access hardware-based security functionalities
4. **Environment API**: to access/control sensors/actuators

It is clear that there can be additional Computing Platform functionalities that are needed by the burgeoning field of mobile devices, an important instance of second screen devices. MPEG is already addressing some sensors and actuators Engine(s) [11].

The following sections provide additional information on the architecture and its components.

**4.2.2 Applications**

Applications offer the main functionalities of converting:

1. User actions into input to the Middleware via HL API or with the Computing Platform via LL API

In general a Peer has several Applications running on it. Some applications may be resident (i.e., pre-loaded by the Peer manufacturer) while some may be temporary (e.g. downloaded for a specific purpose).

**4.2.3 Middleware**

Middleware acts on user input received from applications via HL API, possibly hands it over to the Computing Platform layer via LL API, and returns any result from internal processing and/or the Computing Platform to the application via HL API.

The Middleware has a software architecture specified in [5] that includes a number of “Engines”. As mentioned above Engines are of two types, namely

1. **Technology Engines**, i.e. software (or hardware) modules capable of performing specific functions such as encoding or decoding a video, encrypting or decrypting a file etc. in response to a call by an Application
2. **Protocol Engines**, i.e. software modules that are called by a local or remote Application via the Middleware needing an Elementary Service such as creation of a licence and description of a video. Such Services are called elementary because they cannot be meaningfully subdivided into more atomic services.

A call to a Protocol Engine typically triggers one or more calls to a number of Technology Engines. For instance, the invocation of the Create Content Protocol Engine triggers a sequence of calls to the Metadata, Licence and Event Report Technology Engines. This functionality – called Orchestration – is the creation of chains of Technology Engines to provide a higher level functionality, typically executed by a particular engine called Orchestrator.

A similar functionality exists at the Protocol Engine level. While PEs are implementations of Elementary Services, i.e. Services at an atomic level [7], PE Aggregation is the functionality of combining Protocol Engines to provide a higher level (i.e. non-elementary) service [8]. An Application may call a local or remote, Elementary or Aggregated Service via the Middleware.

Currently BRIDGET does not envisage using the service aggregation functionality.

Figure 3 illustrates the combination of Aggregation and Orchestration functionalities. Each Protocol Engine in an aggregation of three Elementary Services calls a number of Technology Engines. PE, calls
just one, PE_b calls a sequence of three and PE_c calls two. In the latter two cases an Orchestration Technology Engine is needed.

![Figure 3 - Aggregation and Orchestration](image)

### 4.2.4 Computing Platform

The Computing Platform includes the following components:

1. **Local Resources** that performs storage and processing of resources;
2. **Network** that processes requests from Middleware via Low Level API and vice versa;
3. **Security** that supports security functions using a hardware component that is specific of the platform
4. **Environment** that accesses/controls sensors/actuators

### 4.3 Design Tools and APIs documentation

The BRIDGET architecture is documented using the Unified Modelling Language (UML). UML defines a set of diagram types to describe different views of a system:

- **Structure diagrams**, including class, package, object, composite structure, component, profile, and deployment diagrams
- **Behaviour diagrams**, including use case, activity, and state machine diagrams
- **Interaction diagrams**, including sequence, communication, timing, and interaction overview diagrams.

In BRIDGET, systems are described using class, use case, and sequence diagrams.

**Class diagrams** describe the relation between different system modules and the provided external interfaces. They define what must be implemented in the system in terms of components, and are useful to specify the part of the system architecture that is time independent.

**Use case diagrams** show how many different kind of users are involved in the system and how they interact with the system itself.

**Sequence diagrams**, as a subset of behaviour diagrams, define how the interfaces (APIs) of system modules are used and the control flow between modules interacting with each other.

It’s important to remark the great importance of developing these Application Programming Interfaces (APIs) according to the MPEG Extensible Middleware (MXM) API, thus providing access to the BRIDGET middleware and its technology engines.

These APIs can be divided in four categories:

1. **Creation APIs**, used to create data structures, files and elementary streams conforming to the respective standards;
2. **Editing APIs**, used to modify an existing data structure, file, elementary stream in order to obtain a derived object still conforming to the respective standard;
3. **Access APIs**, used to parse data structures, files, decode elementary streams in order to retrieve the information contained within;
4. Engine-specific APIs, used to accomplish a specific functionality of the developed technology engine, e.g. APIs for Visual Search engine.

Finally, every API involved both in internal system interaction, e.g. between system modules, and external interaction, e.g. between users interacting with system modules, will be appropriately documented, fulfilling the needs of keeping a modular approach and providing independent developing of different part of the system (e.g. developing different technology engines).
5 High Level Architecture

This section describes the BRIDGET architecture from a high level view. Figure 4 presents the basic model that BRIDGET would like to enable. A number of actors (Broadcasters, Application or Service Providers but also users) can create BRIDGET applications and complex services coordinating multiple BRIDGET applications. Applications use functionalities provided by a set of defined BRIDGET components and exposed through a consistent API by a middleware layer. This layer may provide additional components beyond those developed in BRIDGET (e.g. video decoding capabilities) to create particular applications. The applications can access these APIs locally or remotely in a distributed environment. Applications are locally installed or delivered through a network to the actual user devices and then executed. Depending on the context, an application may be developed to interact with a Professional User (e.g. in a broadcaster’s post production environment) or an end user.

![Figure 4 - BRIDGET Architectural model](image)

According to the MPEG-M architecture model, a device executing a BRIDGET application is called a “BRIDGET Peer” and is made up of an application layer, a middleware layer and a computing platform/infrastructure layer as described in Figure 5. Each layer has its own structure and can communicate with other layers via standard APIs (High Level APIs and Low Level APIs).
Examples of BRIDGET Peers are represented in the architecture diagram in Figure 6. The diagram describes the main components and interfaces of the system developed by the project in the first year to implement the scenario of interest selected in [3].

The BRIDGET Professional Authoring Peer and the BRIDGET End User Peer include the application logic as well as the MXM middleware components required.
6 Broadcast/Broadband Hybrid Architectural Model

A BRIDGET service integrates technologies and tools required to create and consume bridgets (i.e., active links), allowing to navigate from a media content to another media content in a transparent and natural way. This functionality is considered a key enabler to enrich the experience of users watching broadcast TV taking advantage of personal devices able to access content through the Internet.

Access and consumption of content by the user through the BRIDGET system follows a hierarchical flow, described in the Figure below:

The content displayed on the main screen and delivered through a broadcast channel (e.g., DVB-T/S/C) is the main driver for accessing and consuming related additional or enhanced content. The second screen, using a broadband connection, can retrieve and present available bridgets, and when the user selects one of them, it displays the related target content.

According to requirements collected from Broadcasters in the early phases of the project, in this version of the BRIDGET architecture we define a hybrid broadcast/broadband model for the system that does not make assumptions on the availability of an interactive middleware able to run applications or services on the main screen (like a GEM-based or HbbTV stack) nor do we define particular broadcast signalling methods (like EIT events for DVB-T) to convey additional information in the programme stream.

The following Table lists the class of content and information retrieved by the main screen and by the second screen specifying if a broadcast or broadband connectivity is used:
A critical functional requirement of the system is the ability to announce, present and consume content on the second screen that is related to a particular time interval of the programme displayed on the main screen. In this scenario, the main content is delivered via broadcast, while the related linked content and information are delivered via broadband. This hybrid model, within the home network environment therefore requires two functionalities: the first is the identification of the content currently displayed on the main screen, the second is the ability to announce and present a suitable representation of bridgets on the second screen synchronised with the playback of the programme on the main screen.

The identification of the content is typically dependent on asset information created and handled by the broadcaster for a specific programme. Different broadcast platforms may use different identification systems and, in general, identification data transmitted with the broadcast stream is kept to a minimum in order to save bandwidth. In BRIDGET, the content identification is required mainly to retrieve the related bridget assets in MP4 scenes and the synchronisation information needed to present them. In Version A, this is done directly by the second screen application, retrieving EPG information through the Internet. Other mechanisms may be possible, including automatic detection of the programme using video search capabilities or using information available on the main screen (when an interactive application is available), however these features are not targeted within the project lifetime.

Once a programme is identified, synchronisation of bridget announcement and presentation is performed through an audio-based fingerprint solution. The second screen application analyses continuously (but with a low frequency) the audio of the main programme, extracting a fingerprint for a short window of audio track and comparing it with a list of synchronisation points defined for that programme. When a matching synchronisation point is found, the second screen scene time line is aligned with the main screen time line. This time line is furthermore used to create the presentation of the bridgets available in the application as depicted in Figure 8.

It should be noted that a continuous synchronisation of the timelines of the main programme and of the target additional content is not required.
7 Detailed Architecture

This chapter describes the BRIDGET architecture in terms of logical components and main interfaces. Each component provides a distinct functionality and is associated to one of the following logical layers:

- Application Layer
- Middleware Layer
- Data Layer

The application layer typically includes a Graphical User Interface to interact with a user and an application logic component providing the custom business logic associated to the application. The application logic component, receives commands from the user through the GUI, and modifies its state by executing the required operation. It is also responsible for handling communications with data sources available in the data layer. In an MVC (Model-View-Controller) architectural pattern, the GUI will implement the View functionalities whereas the application logic will provide the Model and Controller functionalities.

Following the MPEG-M model, the application layer interacts with a middleware layer, providing BRIDGET-specific media functionalities exposed as MPEG-M Technology Engines through a BRIDGET High Level API. Depending on the complexity of the operation, the application logic may need to coordinate and orchestrate the different Engines. This logic may be embedded in the application logic or, if generic enough, encapsulated in a Protocol Engine responsible to orchestrate multiple Technology Engines at the middleware level. In this version of the architecture, since the developments are currently on-going, we assume that the orchestration functionalities are provided by the application logic. Complex functionalities such as Media Analysis, Visual Search, and 3D Reconstruction are modelled and defined as “proto engines”, because it is anticipated that each of them will be decomposed in a set of MPEG-M engines in the next version of the architecture, as a result of the research and development activities in WP4, WP5, and WP6.

The third layer, the Data layer, logically represents the components acting as Data sources or destinations (including media content and metadata) for the system. No assumptions are made at this level on where these components are physically located or with what kind of infrastructure or protocols the application and middleware components can access them.

It should be noted that an MPEG-M infrastructure layer is always implicitly present in the architecture (although not included in the diagrams for the sake of simplicity), providing generic computing and network resources.

In the next sections we provide the logical architecture diagrams for both the Professional Authoring Peer and the End User Peer. The diagrams are UML Component Diagrams describing the different main layers, interfaces, and components for each peer. The components are organised in different colours, each corresponding to the work package that will be in charge of the related development. It should be noted that a few components are white coloured, meaning that the component will not be entirely developed within the project but will be reused or adapted. One important example is the
**MediaFramework engine** that provides media decoding functionalities. This engine, already available in the MPEG-M middleware will be reused by the project. Other components in the data layer are usually available in any real production environment (e.g. a digital asset management system - DAM) of a service provider or broadcaster. The project will not develop its own DAM but will focus on the integration with actual systems. For demonstration purposes some of these components may be replaced by software simulating interface and behaviour of those systems.

### 7.1 Professional Authoring Peer Logical Architecture

The Professional Authoring Peer (Version A) is a system providing, to a Professional User, the following high level functionalities:

- Enrichment of broadcast programmes with additional on demand content, according to the broadcast planning;
- Association and synchronisation of a bridge to a defined point (or portion) in the timeline of the programme, thanks to the Fingerprint Extractor TE;
- Consolidation and rendering of editorial decisions concerning bridges;
- Media content analysis on a set of selected media items, via the Media Analysis TE;
- Content search service to select content to be linked, via the Visual Search TE;
- 3D reconstruction on a set of selected 2D media items, via the 3D Reconstruction TE.

Figure 9 describes the component, main interfaces and the interactions between components for the Professional Authoring Peer.

![Figure 9 - BRIDGET Professional Authoring Peer Logical Architecture](image-url)
Table 3 provides the list of engines used by the Professional Authoring Peer that will be developed for Version A. For each engine or proto engine a short description is provided as well as a link to the chapter of this document describing it and the work package responsible for the related developments.

<table>
<thead>
<tr>
<th>Proto Engine</th>
<th>Description</th>
<th>Chapter</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Analysis</td>
<td>analyses a media content producing a hierarchical temporal structure representation of the content, as well as low-level descriptors (e.g. visual content-based), high-level annotations (e.g. scene classification), and quality measures (e.g. visual or audio quality based).</td>
<td>7.5.1</td>
<td>WP4</td>
</tr>
<tr>
<td>Visual Search</td>
<td>supports search for visual objects in large image and video libraries: analyses images and videos in the content libraries, extracts compact descriptors, builds descriptor databases and indexing schemes, ranks results</td>
<td>7.5.2</td>
<td>WP5</td>
</tr>
<tr>
<td>3D Reconstruction</td>
<td>creates a 3D model from input images and/or videos</td>
<td>7.5.3</td>
<td>WP6</td>
</tr>
<tr>
<td>Fingerprint Extractor</td>
<td>extracts an Audio Fingerprint (AFP) from an uncompressed chunk of audio data</td>
<td>7.5.4</td>
<td>WP7</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>provides access to bridget data structures and the associated metadata</td>
<td>7.5.5</td>
<td>WP7</td>
</tr>
<tr>
<td>3D Compression</td>
<td>produces a compressed 3D graphics data structure from a 3D model that can be efficiently transmitted, and later decompressed and rendered to screen</td>
<td>7.5.6</td>
<td>WP6</td>
</tr>
<tr>
<td>MediaFramework</td>
<td>The MediaFramework is a high level MPEG-M Engine, grouping together several media specific engines such as Video, Image, Audio, and File Format Engine. It also implements common functionalities (independent on the media type) such as resource loading and saving.</td>
<td>7.5.8</td>
<td>MPEG-M Part 2</td>
</tr>
</tbody>
</table>

7.2 End User Peer Logical Architecture

The End User Peer (Version A) is a system which provides an End User with the following high level functionalities:

- Additional related content retrieval based on the identification of the media item and of the media time point being watched on the main screen;
- Access to bridgets synchronised with the programme displayed on the main screen, independently from the broadcast network (e.g. DVB-T/S/C) used to receive the programme, thanks to the Synchroniser TE and to the MediaFramework TE;
- Media aligned and global bridgets bookmarking for future use;
Figure 10 describes the component, main interfaces and the interactions between components for the End User Peer (Version A).

![Figure 10 - BRIDGET End User Peer Logical Architecture](image)

Table 33 provides the list of engines used by the End User Peer that will be developed for Version A. For each engine a short description is provided as well as a link to the chapter of this document describing it and the work package responsible for the related developments.

**Table 4 - End User Peer Engines**

<table>
<thead>
<tr>
<th>Engine</th>
<th>Description</th>
<th>Chapter</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Compression</td>
<td>Compress and decompress 3D graphics data structure representing a 3D model that can be efficiently rendered to screen</td>
<td>7.5.6</td>
<td>WP6</td>
</tr>
<tr>
<td>Synchroniser</td>
<td>Monitors an audio stream to detect synchronisation points</td>
<td>7.5.7</td>
<td>WP7</td>
</tr>
<tr>
<td>Fingerprint Extractor</td>
<td>extracts an Audio Fingerprint (AFP) from an uncompressed chunk of audio data</td>
<td>7.5.4</td>
<td>WP7</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>provides access to bridget data structures and the associated metadata</td>
<td>7.5.5</td>
<td>WP7</td>
</tr>
<tr>
<td>MediaFramework</td>
<td>The MediaFramework is a high level MPEG-M Engine, grouping together several media specific engines such as Video, Image, Audio, and File Format Engine. It also implements common functionalities (independent on the media type) such as resource loading and saving.</td>
<td>7.5.8</td>
<td>MPEG-M Part 2</td>
</tr>
</tbody>
</table>
7.3 Physical Model

The proposed approach is based on virtual clusters, thus any component can be deployed on any of the nodes (with the corresponding middleware). It is the scheduler’s job to choose which request goes to which node. If all the components (Engines) are deployed on every node, the scheduler should choose based only on load status.

The virtual cluster allows the coexistence of all the components authoring and others in the same cluster. Also, a separation of services is possible: 1 cluster for authoring, 1 cluster for fingerprint matching, and 1 cluster to serve packages (the clusters can also exchange data if necessary).

7.4 Data Model

7.4.1 Logical Data Model

This section includes the logical data model of the BRIDGET system, which has been derived by analysing requirements defined in [1]. Such a model is a fundamental component needed to perform a solid integration of the different components of the system. While individual components (engines) may implement a different internal physical model to store and manage the information about the objects pertinent to them, when interacting with other components it is essential to follow a well-established and shared logical model, with a corresponding well defined physical implementation (e.g., following an XML Schema). As a consequence, each engine will have to build an adaptation layer able to parse and produce instances of the physical model (e.g., XML documents compliant to the defined XML Schemas) in order to interact with the other components.

The physical implementation of the model (XML Schemas), will be fully specified in Deliverable D3.2 – Content Transport Specification.

7.4.1.1 Bridget Description

This section describes the Bridget Description package of the BRIDGET logical data model. The corresponding physical implementation of this part will be used by the Bridget Description Engine to store and manage bridget descriptions in a structured way. The same description will be used by other engines interacting with the Bridget Description Engine when invoking its APIs and by other components of the system to store bridget related information (e.g., the Authoring tool). Specifically, on the End User Peer side, the model will be used to acquire knowledge about which bridgets are associated to the currently identified media item and which Time Aligned Bridget are in scope of the current time instant. On the Professional Authoring Peer side, the model will be used to retrieve information about stored bridgets and to save information about the newly created bridgets from/on the Bridget DB.
### Class Semantics

#### Table 5

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridget Description</td>
<td>GenericBridget</td>
<td>A generic abstract class representing all types of bridgets</td>
<td>Specialised classes: GlobalBridget, TimeAlignedBridget</td>
<td>Related classes: ProductionMetadata, ContentItem</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>ProductionMetadata</td>
<td>A class containing production metadata of a bridget (e.g., author)</td>
<td>Related classes: GenericBridget</td>
<td></td>
</tr>
<tr>
<td>Bridget Description</td>
<td>TimeAlignedBridget</td>
<td>A class representing time aligned bridgets</td>
<td>Parent class: GenericBridget</td>
<td>Related classes: SpatioTemporalScope</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>GlobalBridget</td>
<td>A class representing global bridgets</td>
<td>Parent class: GenericBridget</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>ContentItem</td>
<td>A class representing a generic content item</td>
<td>Related classes: GenericBridget</td>
<td>See Section 7.4.1.2</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>hasSourceContentItem</td>
<td>Association class linking a generic bridget to its source content item</td>
<td>Related classes: GenericBridget, ContentItem</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 11 - Bridget Description Data Model](image-url)
<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridget Description</td>
<td>hasDestinationContentItem</td>
<td>Association class linking a generic bridget to its destination content items</td>
<td>Related classes: GenericBridget, ContentItem</td>
<td></td>
</tr>
<tr>
<td>Bridget Description</td>
<td>SpatioTemporalScope</td>
<td>A relation class specifying spatiotemporal coordinates of the source content item of a time aligned bridget in which each destination content item is associated</td>
<td>Related classes: TimeAlignedBridget, ContentItem</td>
<td></td>
</tr>
</tbody>
</table>

### 7.4.1.2 Content Description

This section describes the Content Description Package of the BRIDGET logical data model. This part represents the core descriptive scheme for content items (i.e., audiovisual media items, images, text documents and 3D reconstruction media objects). Though the full specification of the physical implementation of this model shall be object of D2.3, it is anticipated that it will be heavily based on MPEG-7 AVDP [16]. This model shall be used and implemented by all engines dealing with managing the creation, consumption and enrichment of content items. Specifically, the Media Analysis engine will provide its output according to this model and the Visual Search engine shall be able to parse the information provided by the former engine to process and store features useful for visual search purposes.

![Figure 12 - Content Description Data Model](image)

### Class Semantics

**Table 6**

D3 1 - BRIDGET System Architecture and Interfaces - Version A.docx  
27/59
<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Description</td>
<td>ContentItem</td>
<td>A generic class representing a content item (e.g., a picture, a video)</td>
<td>Related classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContentMetadata, ContentIdentifier, Descriptor</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>ContentMetadata</td>
<td>A structured class containing production and descriptive metadata of a content item</td>
<td>Related classes:</td>
<td>The structure of the properties of this class can be conforming to EBU Core Metadata set [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>ContentIdentifier</td>
<td>A structured class containing identification properties of a content item</td>
<td>Related classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>Descriptor</td>
<td>A generic class representing descriptors of a content item, i.e. data structures collecting descriptive, semantic, structural or low-level audiovisual features of a content item</td>
<td>Related classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>ContentSegment</td>
<td>A class representing information related to the temporal validity of a descriptor, as related to the timeline of the associated content item</td>
<td>Related classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptor</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>StructuralDescriptor</td>
<td>A generic class representing descriptors about structural aspects of a content item (e.g., shot segmentation, shot grouping)</td>
<td>Parent classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specialised classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CandidateBridgePointsDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EditorialSegmentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ShotSegmentation</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>SematicDescriptor</td>
<td>A generic class representing descriptors about semantic aspects of a content item</td>
<td>Parent classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptor</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>VisualDescriptor</td>
<td>A generic class representing descriptors about visual aspects of a content item</td>
<td>Parent classes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descriptor</td>
<td></td>
</tr>
<tr>
<td>Namespace</td>
<td>Class Name</td>
<td>Definition</td>
<td>Related To</td>
<td>Note</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Content Description</td>
<td>AuralDescriptor</td>
<td>A generic class representing descriptors about aural aspects of a content item</td>
<td>Parent classes: Descriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specialised classes: AuralDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CandidateBridgetPoints</td>
<td>A class representing descriptors about segments of the content item which can be candidate of a bridget insertion</td>
<td>Parent classes: StructuralDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditorialSegmentation</td>
<td>A class representing descriptors of editorial parts of a content item (e.g., a scene, an interview, a jingle)</td>
<td>Parent classes: StructuralDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ShotSegmentation</td>
<td>A class representing descriptors of visual shots</td>
<td>Parent class: StructuralDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDVSDescriptor</td>
<td>A class representing Compact Descriptors for Visual Search</td>
<td>Parent class: VisualDescriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AudioFingerprint</td>
<td>A class representing audio fingerprints</td>
<td>Parent class: AuralDescriptor</td>
<td></td>
</tr>
</tbody>
</table>

7.4.1.3 Content Types

This section describes the basic sub-types of content items managed in the BRIDGET system. Each specific content type may carry specialised production-related or descriptive metadata and may be described using different sets of structural, semantics, visual or aural descriptors.
Figure 13 - Content Types Data Model

Class Semantics

Table 7

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Description</td>
<td>ContentItem</td>
<td>A generic class representing a content item (e.g., a picture, a video)</td>
<td>Specialised classes: VideoContentItem, AudioContentItem, TextContentItem, ImageContentItem, 3DModelContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>VideoContentItem</td>
<td>A class representing video content items</td>
<td>Parent class: ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>AudioContentItem</td>
<td>A class representing audio content items</td>
<td>Parent class: ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>TextContentItem</td>
<td>A class representing text content items</td>
<td>Parent class: ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>ImageContentItem</td>
<td>A class representing still pictures content item</td>
<td>Parent class: ContentItem</td>
<td></td>
</tr>
<tr>
<td>Content Description</td>
<td>3DModelContentItem</td>
<td>A class representing 3D model content items</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.4.1.4 Bridget Presentation

This section describes the Bridget Presentation package of the overall BRIDGET logical data model. The corresponding physical implementation of this part of the logical data model shall be used by the End User Peer to retrieve and manage information about how to present bridget information, e.g. during which specific time segment of the media timeline, or using which representative icon, in the case in which such information is obtained from the Bridget DB. A simplified version of the physical representation of the model may also be stored on the local device to manage information about bookmarked bridgets.

![Figure 14 - Bridget Presentation Data Model]

Class Semantics

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridget Presentation</td>
<td>BridgetPresentation</td>
<td>An abstract class representing global bridget views and time aligned bridges presentation</td>
<td>Specialised classes: BridgetPresentationTimeline, GlobalBridgetView</td>
<td></td>
</tr>
<tr>
<td>Bridget Presentation</td>
<td>BridgetPresentationTimeline</td>
<td>A class representing presentation parameters for time aligned bridges</td>
<td>Parent class: BridgetPresentation, Related classes: TimeAlignedBridget</td>
<td></td>
</tr>
<tr>
<td>Bridget Presentation</td>
<td>GlobalBridgetView</td>
<td>A class representing presentation parameters for global bridgets</td>
<td>Parent class: BridgetPresentation, Related classes: GlobalBridget</td>
<td></td>
</tr>
<tr>
<td>Namespace</td>
<td>Class Name</td>
<td>Definition</td>
<td>Related To</td>
<td>Note</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Bridget Presentation</td>
<td>TimeAlignedBridgetRepresentationParameters</td>
<td>An association class representing parameters used to present time aligned bridgets</td>
<td>Related classes: BridgetPresentationTimeline, TimeAlignedBridget</td>
<td></td>
</tr>
<tr>
<td>Bridget Presentation</td>
<td>GlobalBridgetRepresentationParameters</td>
<td>An association class representing parameters used to present global bridgets</td>
<td>Related classes: GlobalBridgetView, GlobalBridget</td>
<td></td>
</tr>
<tr>
<td>Bridget Description</td>
<td>TimeAlignedBridget</td>
<td>A class representing time aligned bridgets</td>
<td>Related classes: TimeAlignedBridgetRepresentationParameters</td>
<td>See Section 7.4.1.1</td>
</tr>
<tr>
<td>Bridget Description</td>
<td>GlobalBridget</td>
<td>A class representing global bridgets</td>
<td>Related classes: GlobalBridgetRepresentationParameters</td>
<td>See Section 7.4.1.1</td>
</tr>
</tbody>
</table>

### 7.4.1.5 Content – Based Query Description

This section describes the Content- based Query package of the overall BRIDGET logical data model. The corresponding physical implementation of this part of the logical data model shall be used by the End User peer and the Professional Authoring peer to invoke generic metadata- and descriptor- based queries. The model is represented by two main cases; a query launched through the specification of a descriptor and a query launched through the specification of a reference to a content item. In the former case, depending on the specific type of descriptor used in the query, the engine invoked to satisfy the query will be different. For example, if a visual descriptor is used, this will be the Visual Search engine, while if an aural descriptor is used (an audio fingerprint), this will be the Synchroniser engine. When a reference to a content item is used (e.g., a picture), the corresponding engine in charge of satisfying the query shall autonomously extract the useful information from the referenced content item to perform its retrieval process.
Class Semantics

Table 9

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryByContent</td>
<td>QueryByContent</td>
<td>An abstract class representing queries by content descriptions and queries by content items</td>
<td>Specialised classes: QueryByContentDescription, QueryByContentItem</td>
<td></td>
</tr>
<tr>
<td>Namespace</td>
<td>Class Name</td>
<td>Definition</td>
<td>Related To</td>
<td>Note</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>QueryByContent</td>
<td>QueryByContentDescription</td>
<td>A class representing queries specified through a descriptor (e.g., a SemanticDescriptor)</td>
<td>Parent class: QueryByContent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryByContentItem</td>
<td>A class representing queries specified through a content item (e.g., a still picture)</td>
<td>Parent class: QueryByContent</td>
<td></td>
</tr>
<tr>
<td>QueryByContent</td>
<td>QueryByVisualDescriptor</td>
<td>A class representing queries specified through a descriptor (e.g., a CDVDescriptor)</td>
<td>Parent class: QueryByContentDescription</td>
<td>Related classes: VisualDescriptor</td>
</tr>
<tr>
<td>QueryByContent</td>
<td>QueryByAuralDescriptor</td>
<td>A class representing queries specified through an audio descriptor (e.g., a AudioFingerprint)</td>
<td>Parent class: QueryByContentDescription</td>
<td>Related classes: AuralDescriptor</td>
</tr>
<tr>
<td>ContentDescription</td>
<td>Descriptor</td>
<td>A generic class representing descriptors of a content item, i.e. data structures collecting descriptive, semantic, structural or low-level audiovisual features of a content item</td>
<td>Related classes: QueryByContentDescription</td>
<td></td>
</tr>
<tr>
<td>ContentDescription</td>
<td>VisualDescriptor</td>
<td>A generic class representing descriptors about visual aspects of a content item</td>
<td>Related classes: QueryByVisualDescriptor</td>
<td></td>
</tr>
<tr>
<td>ContentDescription</td>
<td>AuralDescriptor</td>
<td>A generic class representing descriptors about aural aspects of a content item</td>
<td>Related classes: QueryByVisualDescriptor</td>
<td></td>
</tr>
</tbody>
</table>
7.4.1.6 3D Reconstruction Request Description

This section describes the Content-based Query package of the overall BRIDGET logical data model. The corresponding physical implementation of this part of the logical data model shall be used by the End User peer and the Professional Authoring peer to invoke 3D reconstructions from the 3D reconstruction engine, and to classify the results in terms of levels of refinements. Correspondingly, the 3D reconstruction engine may use the model to locally store information about the received requests.

![Figure 16 - 3D Model Reconstruction Request Data Model](image)

**Class Semantics**

**Table 10**

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Class Name</th>
<th>Definition</th>
<th>Related To</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Reconstruction Request</td>
<td>3DReconstructionRequest</td>
<td>A class representing a generic 3D reconstruction request</td>
<td>Related classes: ContentDescription, 3DModelContentItem</td>
<td></td>
</tr>
<tr>
<td>3D Reconstruction Request</td>
<td>3DmodelFeatures</td>
<td>An association class representing parameters qualifying the 3D reconstruction</td>
<td>Related classes: 3DreconstructionRequest, 3DmodelContentItem</td>
<td></td>
</tr>
<tr>
<td>ContentDescription</td>
<td>ImageContentItem</td>
<td>A class representing still pictures content item</td>
<td>Related classes: 3DReconstructionRequest</td>
<td>See Section 7.4.1.2</td>
</tr>
<tr>
<td>ContentDescription</td>
<td>3DModelContentItem</td>
<td>A class representing 3D model content items</td>
<td>Related classes: 3DReconstructionRequest</td>
<td>See Section 7.4.1.2</td>
</tr>
</tbody>
</table>

7.4.2 Authoring tool Database Model

The authoring tool uses an internal data model, mapped to a distributed database. During the authoring process, data will be extracted from the database and exported in a format interpreted by the Bridget player. The data model used by the Bridget player is MPEG ARAF and is introduced in the next section. At the authoring tool level the following information is part of the data model:

- the entire layout of the bridget presentation (styles and formatting, media and visual elements)
- the region of interest (the selection used in Visual Search)
- program metadata
- program fingerprints
• the selected set of results obtained by running the engines from the middleware layer and the association with the bridget bridget metadata and timeline

The database behind the authoring tool is structured, at a conceptual level, as depicted in Figure 17.

![Figure 17 - Authoring tools database structure](image)

7.4.3 ARAF Data Model

The Augmented Reality Application Format (ARAF) is an extension of a subset of the MPEG-4 part 11 Scene Description and Application Engine standard, combined with other relevant MPEG standards (MPEG-4, MPEG-V), designed to enable the consumption of 2D/3D multimedia content as depicted in Figure 18.

An ARAF, available as a file or stream, is interpreted by a device, called an **ARAF device** (End user’s BRIDGET Peer in the figure 18). The nodes of the ARAF scene point to different sources of multimedia content such as 2D/3D image, 2D/3D audio, 2D/3D video, 2D/3D graphics and sensor/sensory information sources/sinks that are either remote or/and local.
The description of the multimedia scene ARAF is based on the standard MPEG-4 Part 11 BIFS [12] which at its turn is based on VRML97 [13]. About two hundred nodes are standardised in MPEG-4 BIFS and VRML, allowing various kinds of scenes to be constructed. ARAF is referring to a subset of MPEG-4 BIFS nodes for scene description. The nodes and prototypes included in the end user's BRIDGET Peer and proved by MPEG as part of ARAF are presented below:

- **Elementary media:**
  - Audio
  - Image and video
  - Textual information
  - Graphics

- **Programming information**
- **User interactivity**
- **Scene related information (spatial and temporal relationships)**
- **Dynamic and animated scene**
- **Communication and compression:**
  - Maps

**Terminal**

MPEG-4 Part 11 describes a scene with a hierarchical structure that can be represented as a graph. Nodes of the graph build up various types of objects, such as audio video, image, graphic, text, etc. A specific method, called PROTO, can be used to add user-defined types of node.

In general, nodes expose a set of parameters, through which, aspects of their appearance and behaviour can be controlled. By setting these values, scene designers have a tool to force a scene-reconstruction at clients’ terminals to adhere to their intention in a predefined manner. In a more complicated scenario, the structure of BIFS nodes is not necessarily static; nodes can be added or removed from the scene graph arbitrarily.

Certain types of nodes called **sensors**, such as TimeSensor, TouchSensor, can interact with users and generate appropriate triggers, which are transmitted to other nodes by routing mechanisms, causing...
changes in state of the receiving nodes. They are bases for the dynamic behaviour of a multimedia content supported by MPEG-4.

The maximum flexibility in the programmable features of MPEG-4 scene is carried out with the Script node. By routing mechanism to the Event In `valueIn` attribute of Script node, the associated function (defined in its URL attribute) with the same name Event In `valueIn` () will be triggered. The behaviour of this function is user-defined, i.e. the scene-designer can freely process some computations, and set the values for every Event Out `valueOut` attribute, which consecutively affects the states of other nodes linked to them.

Direct manipulation of nodes’ states is also possible in MPEG-4 Part 11: the `field` attribute can refer to any node in the scene; through this link, all attributes of the contacted node will be exposed to direct setting and modifying operators within the Script node. The syntax of the language used to implement the function of Script node is ECMAScript [12].

ARAF supports the definition and reusability of complex objects by using the MPEG-4 PROTO mechanism. The PROTO statement creates its own nodes by defining a configurable object prototype; it can integrate any other node from the scene graph.

7.5 Engine descriptions

The following sub-sections describe the different engines identified for Version A. Since research, development and integration activities are currently on going in WP4, WP5, WP6, and WP7 it is anticipated that some modifications or extensions may be required after the finalisation of this deliverable. Any changes will be duly reported and integrated in the next version of this deliverable (Version B). As mentioned in section 7, Media Analysis, Visual Search, and 3D Compression are defined as "proto engines" because it is expected that the research and development activities in WP4, WP5, and WP6 will result in the definition of a set of suitable MPEG-M engines for each functionality.

7.5.1 Media Analysis Proto Engine

7.5.1.1 Description

The purpose of the Media Analysis Proto Engine is to provide a hierarchical temporal structure analysis of media content, as well as deliver low-level descriptors (e.g. visual content-based), high-level annotations (e.g. scene classification), and quality measures (e.g. visual or audio quality based) for media content. These will, in turn, facilitate the manipulation of media content, and bridget generation, in the authoring tools, as well as aid the operation of the Visual Search Proto Engine and 3D Reconstruction Proto Engine, by pre-filtering their input and/or post-processing their results.
7.5.1.2 Interfaces and API

The envisaged interfaces and API between the Media Analysis Proto Engine and other parts of the BRIDGET system are simple. At the most basic level, the various modules of the engine will take a media file as input and produce standardised output files. MPEG-7 AVDP [16] is XML-based and is specifically designed to carry the type of results that the Media Analysis Proto Engine produces, such as a hierarchical temporal segmentation, or media segment descriptors, making it suitable for communication with other parts of the BRIDGET system in a standardised fashion. More specifically, MPEG-7 AVDP is a profile (i.e. subset) of the MPEG-7 standard targeting applications in media production and archiving. The profile can be used to describe the results of various kinds of media analysis such as shot/scene detection, face recognition/tracking, speech recognition, copy detection and summarisation, etc. in a way that these data can be usefully integrated in media production processes. The AVDP profile supports temporal and spatial analysis of audio-visual material, including low-level audio and video descriptions.

Thus, we aim to utilise MPEG-7 AVDP, possibly with extensions as required for BRIDGET. Some analysis and annotation functionalities will require media decoding functionalities that may be provided within the engine or, through a proxy component, directly by the external MediaFramework engine.

The following activity diagram describes a possible workflow to perform a media analysis task that includes an initial structure analysis and segmentation of the media content and a following annotation of a subset of media content frames:

![Figure 20 - Activity diagram for Media Analysis Proto Engine](image-url)
The external API will consist in a generic method that, based on the media input file and the configuration description will create instances of internal components required and will create the required workflow.

The following table shows the expected inputs and outputs for each internal component of the proto engine:

<table>
<thead>
<tr>
<th>Internal Component</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Structure Analysis</td>
<td>• Media file</td>
<td>Standard MPEG-7 AVDP/XML description</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td>Media Annotation</td>
<td>• Media file</td>
<td>Standard MPEG-7 AVDP/XML description</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Output of Media Structure Analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Image local descriptors from Visual Search Proto Engine</td>
<td></td>
</tr>
<tr>
<td>Media Quality Assessment</td>
<td>• Media file</td>
<td>Standard MPEG-7 AVDP/XML description</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Output of Media Structure Analysis / Media Annotation</td>
<td></td>
</tr>
</tbody>
</table>

7.5.1.3 Interactions with other Engines

The results of the Media Analysis Proto Engine will be used in the authoring tools, as well as in the Visual Search Proto Engine and 3D Reconstruction Proto Engine. These operations will be coordinated typically by the application logic component.

Media parsing and decoding functionalities may reuse functionalities provided by the MediaFramework Engine.

Additional APIs may be defined between the Media Analysis Proto Engine and other engines or proto engines, for direct access to the results of the analysis, as needed.

7.5.2 Visual Search Proto Engine

7.5.2.1 Description

Visual Search Proto Engine will support fast and robust visual based search and access to large archives of image and video content in order to facilitate creation of the content links in the authoring tool and also during the multi-screen playback. The engine will work in two modes – analysis mode and search mode. In the analysis mode it will extract local image descriptors and construct aggregated (global) representations at various spatio-temporal resolutions, such as at image level and at a temporal segment levels (for video). The hierarchical temporal structure of video content provided by the Media Analysis Proto Engine will inform the aggregation process. In the search mode, the engine will use a query image or a selected region in an image or video to initiate search for visually similar object in the media library. This will support finding of relevant content for bridget generation in the authoring tools and for 3D reconstruction. It is also expected that low-level local descriptors extracted by the Visual Search Proto Engine will be used for scene classification tasks by the Media Analysis Proto Engine.
7.5.2.2 Interfaces and API

The interfaces and API for Visual Search Proto Engine will support two modes (analysis, search) and interface to other parts of the BRIDGET system, specifically to the Authoring Tools, Media Analysis Proto Engine and 3D Reconstruction Proto Engine.

In the analysis mode, the engine will take a media file (image, video) as input and will produce a standardised descriptor representation for the media. Additionally, for video content it will also accept the information on hierarchical temporal segmentation contained in the AVDP format. We expect to use MPEG CDVS format for images and a suitable extension for the video content.

In the search mode, the Visual Search Proto Engine will take a spatial region in an image and return ranked matching images or frames present in the media library. The engine will also provide a localisation for the matched regions and a measure reflecting confidence/quality of the match.

Some analysis and annotation functionalities will require media decoding functionalities that may be provided within the engine or, through a proxy component, directly by the external MediaFramework engine.

The following activity diagram describes a possible workflow to perform a Visual Search tasks that includes an initial media analysis (descriptor extraction) and the actual search.

Table 12

<table>
<thead>
<tr>
<th>Internal Component/Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Analysis for VS</td>
<td>• Media file (JPEG)</td>
<td>MPEG CDVS</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td>Video Analysis for VS</td>
<td>• Media file (MPEG-4)</td>
<td>MPEG V-CDVS</td>
</tr>
<tr>
<td></td>
<td>• Media temporal structure (ADVP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td>Image search in Image Library</td>
<td>• Media file (JPEG) and/or image ROI</td>
<td>List of ranked matching images, spatial</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td>localisation and confidence (XML)</td>
</tr>
<tr>
<td></td>
<td>• Media library descriptors</td>
<td></td>
</tr>
<tr>
<td>Image search in Video Library</td>
<td>• Media file (JPEG) and/or image ROI</td>
<td>List of ranked matching frames, spatial</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td>localisations and confidence (XML)</td>
</tr>
<tr>
<td></td>
<td>• Media library descriptors</td>
<td></td>
</tr>
<tr>
<td>Local descriptor (feature point) extraction (for Media Analysis Engine)</td>
<td>• Media file (MPEG-4)</td>
<td>Standard MPEG-7 AVDP/XML description</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td>Local descriptor (feature point) extraction for 3D reconstruction</td>
<td>• Media file (MPEG-4)(image sequence) + optional ROI</td>
<td>Feature points (XML)</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.3 Interactions with other Engines

The **Visual Search Proto Engine** will primarily interact with the Authoring tools, but it will also interface to **Media Analysis Proto Engine** and **3D Reconstruction Proto Engine**.

The Authoring tool will request **Visual Search Proto Engine** to analyse new content (images, videos) or to perform visual search based on specific query (image, ROI).

The **Media Analysis Proto Engine** will request **Visual Search Proto Engine** to compute local descriptors as these are needed for media classification. The **Visual Search Proto Engine** will use the results of Media Analysis in visual search.

The **3D reconstruction Proto Engine** will interface to **Visual Search Proto Engine** in order to locate media suitable for reconstruction tasks and to extract and match local features for such reconstruction.

Additional APIs may be defined between the **Visual Search Proto Engine** and other engines, for direct access to the results of the **Visual Search Proto Engine**, as needed. Media parsing and decoding functionalities may reuse functionalities provided by the MediaFramework engine.

### 7.5.3 3D reconstruction Proto Engine

#### 7.5.3.1 Description

The purpose of the **3D Reconstruction Proto Engine** is to create a 3D model from input images and/or videos. The resulting 3D model is of a hybrid structure. The nature of this structure is a matter of ongoing research during the lifetime of this project. For now it is expected that this structure contains simultaneously data of a point cloud representation, a mesh representation and simplified model based representation (such as a cube etc...). The resulting hybrid 3D model will be provided as output to the rendering tools. Further on, the **3D Reconstruction Proto Engine** will tell the user where the input images were taken from with respect to a 3D coordinate system, i.e. a 3D world representation.

#### 7.5.3.2 Interfaces and API

The **3D Reconstruction Proto Engine** requires one interface with the functionalities listed below. Note, that the details of the interface may change dependent on the outcome of the WP6 research. In general, the interface will provide following input/output:

- **input**: multi-media data: single images, video sequence images, audio attached to video
- **output**: hybrid 3D model

A more detailed description can be found in table 12:

<table>
<thead>
<tr>
<th>Internal Component/Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
</table>
| Local descriptor matching for 3D reconstruction | • Media file (MPEG-4)(image sequence) + optional ROI  
• Configuration parameters | Feature points and matching information (XML) |
<table>
<thead>
<tr>
<th>Internal Component</th>
<th>Content</th>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
</table>
| 3D-Reconstruction  |         | `create3DModel`  
|                    |         | • input: ImageSequence  
|                    |         | • output: Hybrid3DModel, CameraCalibration Vector  
|                    |         | `refine3DModel`  
|                    |         | • input: ImageSequence, Hybrid3DModel, CameraCalibration Vector  
|                    |         | • output: Hybrid3DModel, CameraCalibration Vector  
| ImageSequence      | • Raw image data or IDs to image data or file names of image data  
|                    | • sequence IDs image/sequence metadata (EXIF information etc...)  
| Hybrid3DModel      | • Raw 3D point cloud data or file names where those are stored  
|                    | • Raw 3D splatting data or file names where those are stored  
|                    | • Raw simplified 3D model data or file names where those are stored  
|                    | • Raw 3D mesh cloud data or file names where those are stored  
|                    | • sequence/image IDs relevant to map information to `ImageSequence` class  
|                    |         | structure required for wrapping data and parameter for the API / interface  
|                    |         | structure required for wrapping data and parameter for the API / interface  
|                    |         | API / interface to call the 3D Reconstruction Engine functionalities  

<table>
<thead>
<tr>
<th>Internal Component</th>
<th>Content</th>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Calibration</td>
<td>• Camera calibration information</td>
<td></td>
<td>structure required for wrapping data and parameter for the API / interface</td>
</tr>
<tr>
<td></td>
<td>• sequence/image IDs relevant to map information to ImageSequence class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Calibration Vector</td>
<td>• Vector of CameraCalibration data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FeaturePoints</td>
<td>• Raw 2D feature points</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• sequence/image IDs relevant to map information to ImageSequence class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShapeInformation</td>
<td>• Raw shape information</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• sequence/image IDs relevant to map information to ImageSequence class</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.5.3.3 Interactions with other Engines

The following interactions with other engines may be required. Note, that the details of the interface may change depending on the outcome of the WP6 research.

- **Media Analysis Proto Engine**
  - provide audiovisual quality metadata.

- **Visual Search Proto Engine (CDVS)**
  - provide feature points for each image.
    - input: ImageSequence (see description above).
    - output: FeaturePoints (see description above).
  - provide matching feature descriptors between image pairs.
    - input: ImageSequence (see description above).
    - output: FeaturePoints (see description above).
  - shape information, region of interest etc.
    - input: ImageSequence (see description above).
    - output: ShapeInformation (see description above).

- **3D Compression Engine**
  - provide efficient [de]compression of the reconstructed 3D models.

7.5.4 Fingerprint Extractor Engine

7.5.4.1 Description

The Fingerprint Extractor TE is integrated both in the Professional Authoring Peer and the End User Peer: it is the module in charge of extracting an audio fingerprint (AFP) from chunks of few seconds of
uncompressed audio. The robustness of AFP to noise and time misalignments guarantees reliably when synchronising BRIDGET applications with the main screen, thus identifying trigger points for activation/deactivation of bridgets.

In the Professional Authoring Peer, the Fingerprint Extractor TE is accessed by the application logic. In the End Used Peer, the Synchroniser TE utilises the Fingerprint Extractor TE to continuously compute AFPs from the input received by the built-in microphone.

The Fingerprint Extractor Engine aggregates several internal components (Figure 21):

- The Transform Module computes the short-time Fourier transform (STFT) of the audio signal, in order to enable the analysis of the input data in a transformed domain.
- The Analyzer Module applies a peak-picking strategy to the data represented in the transformed domain, in order to identify locally predominant points in the magnitude spectrogram, representing them as a sparse “constellation map”.
- The Binarizer Module creates a compact binary representation of the constellation map, namely the final AFP.

![Figure 21 – Fingerprint Extractor Engine Component Diagram](image)

7.5.4.2 Interfaces and API

The envisaged interfaces and API between the Fingerprint Extractor TE and the other parts of the BRIDGET system are extremely simple. In particular, Fingerprint Extractor TE exposes one functionality (extractAFP) enabling processing of uncompressed audio and computation of related AFP:

```
Byte *AudioAFP extractAFP (Byte *inputWav)
```

Where *inputWav indicates a buffer containing a chunk of uncompressed data (e.g. PCM format), and *AudioAFP is a binary string storing the computed AFP.

The following activity diagram describes the workflow of sequential operations computed during the execution of the extractAFP method:
• STFT computation (Transform Module);
• Detection of peaks in the transformed domain (Analyzer module);
• Computation of a constellation map (Analyzer Module);
• Generation of binary AFP (Binarizer module).

The following table shows the expected inputs and outputs for each internal component of the engine:

<table>
<thead>
<tr>
<th>Internal Component</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform Module</td>
<td>Uncompressed audio</td>
<td>STFT representation</td>
</tr>
<tr>
<td>Analyzer Module</td>
<td>STFT representation</td>
<td>Indexed representation of the constellation map</td>
</tr>
<tr>
<td>Binarizer Module</td>
<td>Indexed representation of the constellation map</td>
<td>Binary AFP</td>
</tr>
</tbody>
</table>

7.5.4.3 Interactions with other Engines

On the Professional Authoring Peer, the Fingerprint Extractor TE is interfaced to the application logic, that is accessing the exposed extractAPF API; on the End User Peer, the same API is accessed by the Synchroniser TE.
7.5.5 BRIDGET Description Engine

7.5.5.1 Description

The Bridget Description Engine shall ensure uniform access to bridgets and the associated metadata. It shall expose via a RESTful API access to all the components of a bridget and allow querying, updating, saving and removing bridgets.

7.5.5.2 Interfaces and API

<table>
<thead>
<tr>
<th>Type of access</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
</table>
| Query          | Bridget identifier  
JSON string containing the bridget property in the format  
• {("bridget_id": bridget identifier)} | Bridget property value or Error message |
| Update         | JSON string containing the bridget property in the format  
• {("bridget_id": bridget identifier),  
(bridget property identifier: bridget property value)} | Bridget property value or Error message |
| Save           | JSON string containing bridget properties in the format:  
• {{bridget property identifier: bridget property value}, ... } | Bridget identifier or Error message |
| Remove         | JSON string containing the bridget property in the format  
• {("bridget_id": bridget identifier)} | Bridget Identifier or Error message |

7.5.5.3 Interactions with other Engines

The BRIDGET Description Engine shall be accessible by any other engines via the REST API, without having a specific interface for any of the engines.

7.5.6 3D Compression Engine

7.5.6.1 Description

The 3D Compression Engine shall ensure the efficiency in the transmission of 3D models by achieving both their efficient compression, at the server's end of the BRIDGET system, and their decompression before rendering, at the client's end of the system. When used to compress a 3D model, it shall take as input the (hybrid) 3D model created by the 3D Reconstruction Engine, and produce as output an efficiently compressed version of it. When used to decompress and render a 3D model, it shall take as input both the compressed version of the 3D model and the viewpoint selected by the user, and yield as output a 3D graphics data structure that will be used by a modified (WP7) version of the GPAC renderer.
7.5.6.2 Interfaces and API

When used to compress a 3D model:

- input: 3D model.
- output: compressed 3D model.

When used to de compress a 3D model:

- input: compressed 3D model, viewpoint.
- output: 3D graphics.

7.5.6.3 Interactions with other Engines

- 3D Reconstruction Proto Engine
  - provide reconstructed 3D models.

7.5.7 Synchroniser Engine

7.5.7.1 Description

The Synchroniser TE is a component of the End User Peer: in particular, it represents the entity in charge of notifying the application logic about the exact timing for activating/deactivating bridgets.

This task is accomplished relying on audio synchronisation, in particular on audio fingerprint. For this reason, the Synchroniser TE is interfaced to the Fingerprint Extractor TE, acting as a wrapper between such module and the application logic.

The Synchroniser TE accesses the SynchronisationDataDB (Data Layer), in order to compare the AFP computed from the audio captured by the built-in microphone with the AFPS associated to the synchronisation times of bridgets.

The architecture of the Synchroniser TE can be represented as follows:

![Figure 23 – Synchroniser Engine Component Diagram](image)

The synchronisation is regulated by control logic that is continuously activating the AFP Extractor, the Audio Recorder and the AFP Comparator Modules, until the input audio fingerprint matches one of the AFPS stored on the SynchronisationDataDB. The comparison of AFPS is performed by the AFPComparator. The AFPExtractor represents a wrapper in charge of accessing the Fingerprint Extractor TE for computing real-time AFP of uncompressed input data. The audio recorder module receives as an input from the application logic a continuous stream of uncompressed audio data, and bufferises it into chunks of the length required for the AFP computation.
7.5.7.2 Interfaces and API

The envisaged interfaces and API between the Synchroniser TE and the other parts of the BRIDGET is relatively simple: in particular, the Synchroniser TE exposes one method to the application logic:

\[ \text{bridgetId findTrigger (Byte *inputAudio)} \]

Where *inputAudio* indicates a buffer containing a chunk of uncompressed PCM data capturing environment sound through the mobile device microphone, and bridgetId refers to the identifier of a bridget associated to the AFP stored on the SynchronisationDataDB.

The sequence of operations performed by the Synchroniser TE when findTrigger procedure is executed is shown in the next figure:

![Figure 24 – Activity Diagram for the FindTrigger procedure](image)

The audio captured by the built-in microphone is transferred by the application logic to the Synchroniser TE: the Audio Recorder module stores it into a buffer, up to a minimum duration necessary to guarantee reliable matching of the computed AFPs.

Regularly (e.g. every 500 ms) the buffered audio chunks are transferred to the AFPExtractor module, that accesses the extractAFP API of the Fingerprint Extractor TE engine in order to compute a fingerprint.

The AFP is compared against those stored on the SynchronisationDataDB, in order to identify possible matches.

In case a stored AFP matches the computed one, the procedure is stopped, and the id of the related bridget is returned to the application logic; otherwise, new input data are gathered from the application logic. The procedure encompasses also a termination control (in case no input data is received), thus allowing the application logic to notify some unexpected termination of the synchronisation process.

The following table shows the expected inputs and outputs for each internal component of the engine:
Table 16

<table>
<thead>
<tr>
<th>Internal Component</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio Recorder Module</td>
<td>• Uncompressed audio stream</td>
<td>PCM buffer</td>
</tr>
<tr>
<td>AFP extractor Module</td>
<td>• PCM buffer</td>
<td>Binary AFP</td>
</tr>
<tr>
<td>AFP comparator Module</td>
<td>• Binary AFP</td>
<td>Id of the identified bridge</td>
</tr>
</tbody>
</table>

7.5.7.3 Interactions with other Engines

In the End User Peer, the Synchroniser TE is interfaced with two components, namely the application logic and the Fingerprint Extractor TE.

7.5.8 MediaFramework Engine

7.5.8.1 Description

The MediaFramework Engine is a high level MPEG-M Engine, grouping together several media specific engines such as Video, Image, Audio and File Format Engine. It also implements common functionalities (independent on the media type) such as resource loading and saving. This engine is defined in ISO/IEC MPEG-M Part 2 and detailed interfaces and APIs are provided in an HTML format specification with a normative value.

7.5.8.2 Interfaces and API

The MediaFramework Engine holds two main interfaces:

- an interface for accessing the Content (called AccessMedia)
- an interface for creating the Content (called CreationMedia)

A typical implementation of the AccessMedia interface of the MediaFramework Engine first loads a resource, demultiplexes it, checks the type of the elementary streams within the resource and calls the associated elementary stream access engines.

A typical implementation of the CreationMedia interface of the MediaFramework Engine calls the associated elementary stream creation engines, initialises them with encoding parameters and finally saves the multiplexed resource.

7.5.8.3 Interactions with other Engines

The MediaFramework engine can provide uncompressed video or audio frames through the application logic layer to the Media Analysis Proto Engine and to the Visual Search Proto Engine.
8 BRIDGET Infrastructure

8.1 Servers and Applications

The BRIDGET platform consists of two major components:

- Processing Platform
- Software Platform

The BRIDGET Processing Platform contains the processing units (physical or virtual), operating systems, network and infrastructure. It is designed to allow flexibility, interoperability, scalability and ease of deployment. In this context, we propose a cloud structure based on virtual clusters.

All the computation resources are provisioned in a virtual cluster, not in a physical cluster. The virtual cluster is a manageable entity based on the need to dedicate a portion of the physical infrastructure, with a dedicated set of machines, as part of the same virtual cluster. The concept of virtual cluster (and virtual cluster node) bring the operational efficiency of the system in that all the administrative tasks required to manage the lifecycle and the elasticity of a virtual cluster instance can be reliably and repetitively automated by the system. In general, there will be a one-to-one mapping between the virtual resources spectrum and the physical resources spectrum. For example, CPU affinity policies will apply when assigning a physical CPU core to a VM's vCPU.

In general, virtual clusters are homogenous (all the nodes are instantiated using the same OS image, have the same number of vCPUs, the same amount of RAM and the same software packages deployed). The platform should allow the deployment of a hybrid type of cluster where the nodes differ in the number of vCPU, amount of RAM and software packages deployment. This can be viewed as a PaaS (Platform as a Service) approach but the particularity of the processing calls introduces the necessity of a custom service that will store and schedule the processes according to the cluster infrastructure.

A configuration management system is used throughout the system, to enforce the consistency of the software that is installed in any virtual cluster instance. This is done by way of running a special program where all successive versions are handled in a central source code control repository. Thus, the provisioning of a fully geared virtual cluster instance could be replayed as many times as necessary though a simple API call.

A virtual cluster is comprised of a complete stack of pre-installed and pre-configured software packages running on one or multiple cluster node instances which altogether constitute an operational virtual cluster on which end-users can launch performance intensive tasks. A virtual cluster may come in different flavours or implementations that are specified in the virtual cluster manifest.

In summary a virtual cluster can be thought of as a pre-built setup that a user could customise in terms of number and flavours of compute nodes, I/O nodes and visualisation nodes to get the required processing and storage capacity.

Figure 25 - Platform Layer based structure
As shown in Figure 25, there is a well-defined layer based architecture to support the instantiation of Virtual Clusters. The layer based structure ensures flexibility by providing mechanism to add or remove physical machines, manage the entire system, handle OS images, handle templates (virtual cluster definitions), storage volume configurations, and configure each node.

Each node is instantiated based on an OS image and does not contain any custom pre-installed packages nor software. At each cluster restart, the node is instantiated based on the same image and the custom software packages are deployed at each boot – everything that is deployed after the instantiation is volatile. This technique ensures the flexibility of using the same base images to deploy a type of hybrid cluster, where one of the nodes is in charge with the exposure of the storage volumes to the nodes.

![Figure 26 - Hybrid Virtual Cluster Schematic](image)

As shown in Figure 26, the hybrid virtual cluster consists of different types of nodes. Each virtual cluster is instantiated based on a template which contains definitions such as the node types (each node type contains the number of vCPUs and amount of RAM), storage and gateway layers configuration.

The network must be partitioned at the virtual cluster level in tight virtual subnets to enforce:

1. traffic isolation across the virtual clusters instances
2. a quality of service (QoS) setup to reserve bandwidth for the various categories of traffic
3. ensure that the services can be accessed from outside the virtual cluster's internal network (through a gateway)

The platform has to ensure that all the nodes can communicate with each other directly (through sockets, pipes etc.) and by opening files generated/saved by other nodes (by having access to the common storage space). As shown in Figure 27, the nodes are connected to the same virtual LAN and the Gateway node has 2 interfaces, one with a virtual LAN IP and the other can be configured with a public IP address.
A common storage space is exposed to all the nodes as a non-volatile volume. The stored data is persistent and is not vulnerable to cluster reconfigurations and start/stop procedures.

To ensure proper functionality of the entire platform, minimum and optimal characteristics are defined. These parameters are strongly dependent on the applications requirements such as: processing power, amount of memory, storage space, data access time.

The following table presents the characteristics for each processing machine:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimal</td>
</tr>
<tr>
<td>CPU cores</td>
<td>2 x Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz</td>
</tr>
<tr>
<td>CPU flags</td>
<td>fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat clflush mmx fxsr sse sse2 syscall nx lm rep_good unfair_spinlock pni pclmulqdq ssse3 cx16 sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes hypervisor lahf_lm</td>
</tr>
<tr>
<td>RAM</td>
<td>4 GB</td>
</tr>
<tr>
<td>Network</td>
<td>1 x GBit</td>
</tr>
<tr>
<td>Persistent Storage space (shared)</td>
<td>1 TB</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Amount</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>Minimal</td>
</tr>
<tr>
<td>Persistent Storage space read performance</td>
<td>30.00 MB/sec</td>
</tr>
<tr>
<td>Persistent Storage space write performance</td>
<td>20.00 MB/sec</td>
</tr>
<tr>
<td>Scratch Storage Space (volatile)</td>
<td>20 GB</td>
</tr>
</tbody>
</table>

Currently, the operating system for the components of BRIDGET processing platform is CentOS 6.5 X86_64.

The BRIDGET Software Platform can be defined as a Software as a Service (SaaS) and contains the software packages with the corresponding deployment and configuration rules.

All the software packages and their dependencies should be packaged in a standard format such as RPM or TAR-BALL. The processing platform allows collecting and installing software from different versioning systems (such as SVN, GIT or by calling a plain HTTP request). The deployment rules should be written as a part of the Node's deployment rules (usually using Chef Recipes). Any additional configurations or operations can be specified using scripting languages (server can be supported).

There are strict rules regarding the software deployment: location, dependencies location and execution rules. Several software packages that use the same dependencies but with different versions, can be deployed on the same node without any interference. The execution rules should specify the dependencies loading location and priority.

A proposed structure is presented below:

![Figure 28 - Software package deployment structure](image-url)
In general, a node is not directly accessible from outside the virtual cluster (Figure 29). The only node that has a public IP address is the gateway node. A load balancing mechanism can be implemented to redirect the request to the least loaded node that is able to handle the client request (contains the corresponding software).

![Figure 29 - Virtual Cluster - Nodes network visibility](image)

### 8.2 User Devices

Within Bridget, we will mainly focus on user devices that run Android OS. The constraints for these devices include: GPU acceleration support, a minimum of 1 GB amount of RAM memory, network connection and a minimum OS version of Android 4.0.0.

The user device chosen as reference platform for the development and the project demonstrations is the Huawei MediaPad M1, that has the following characteristics:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Android OS, v4.2.2 (Jelly Bean)</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
<tr>
<td>CPU</td>
<td>Hisilicon Kirin 910 - Quad-core 1.6 GHz</td>
</tr>
<tr>
<td>Screen size</td>
<td>8 inches</td>
</tr>
<tr>
<td>Screen resolution</td>
<td>800 x 1280 pixels</td>
</tr>
<tr>
<td>GPU</td>
<td>Mali-450MP4</td>
</tr>
</tbody>
</table>
8.3 Network

The BRIDGET network infrastructure is designed to facilitate and support the communications and interactions among the main five distributed elements defined in the BRIDGET architecture, as depicted in Figure 31.

*The Bridget and Content Server* (Bridget Repository) stores the Bridge-enriched media contents generated by the Professional Authoring Peer. Logically, it can be defined a *Serving Node* and it can be accessed via Internet by the (Hybrid) Virtual Cluster and by each Bridget End User Peer.
The stored contents are retrieved through their URL location (basically, a bridget identifies a resource stored in the repository using its URL path).

**The Programme Repository** (Digital Asset Management system) and the **Broadcast Planning** provide access to the original programmes to be Bridget-enriched and to their schedule information. Logically, they can be defined as *Serving Nodes* and they can be accessed via the Internet only by the (Hybrid) Virtual Cluster.

**The (Hybrid) Virtual Cluster** is the core element of BRIDGET Network architecture. Logically, it can be defined a *Border Node* and it can be accessed via Internet only by the Professional User. As shown in Figure 29, it consists of different types of nodes, each of them implementing different functionalities in the Professional Authoring Peer. The nodes are connected to the same virtual LAN via Gigabit Ethernet and the Gateway node has 2 interfaces, one with a virtual LAN IP and the other can be configured with a public IP address. It's important to remark the great importance of this public IP address interface, because the Virtual Cluster needs to communicate and manage a large amount of data downloaded from the Programme Repository, and to access the Bridget Repository to store the bridgets once they have been generated by the authoring process.

Thus, because of the large amount of media content data that the Virtual Cluster needs to access and manage, it's recommended to use:

- High capacity downstream connection between the Programme Repository and the Virtual Cluster, e.g. Hybrid Optical Fiber Solution (Fiber To The Cabinet Technology, up to 30 Mbit/s downstream via VDSL2), Pure Optical Fiber Solution (Fiber To The Home Technology, up to 100Mbit/s downstream), ADSL2+ (up to 20 Mbit/s downstream).

- High capacity upstream connection between the Bridget Repository and the Virtual Cluster, e.g. Hybrid Optical Fiber Solution (Fiber To The Cabinet Technology, up to 3 Mbit/s upstream via VDSL2), Pure Optical Fiber Solution (Fiber To The Home Technology, up to 10Mbit/s upstream).

**The Professional User Authoring Application** is a web application accessing and managing all the Virtual Cluster functionalities via the Internet. Logically, it can be defined as an End Node and it can be used only by the Professional User in the production environment.

**The End User Application** it's a BRIDGET compliant application running on portable second screen devices. Logically, it can be defined an End Node and it provides the End User the opportunity to access and enjoy bridget-enriched media contents stored in the Bridget Repository via Internet.

The second screen device should connect to the home network via WLAN interface (IEEE 802.11 b/g/n up to 450 Mbit/s downstream), and access the Bridget Repository via internet using high capacity downstream connection, e.g. Hybrid Optical Fiber Solution (Fiber To The Cabinet Technology, up to 30 Mbit/s downstream via VDSL2), Pure Optical Fiber Solution (Fiber To The Home Technology, up to 100Mbit/s downstream), ADSL2+ (up to 20 Mbit/s downstream).
9 Conclusions

This document describes the initial system architecture and infrastructure of BRIDGET, providing an overview of its distributed infrastructure modules, engines, and interfaces.

The documentation of the modules and engines is preliminary and reflects the state of actual development, even though some of them have already a defined API.

As planned, there will be a major update of this document, namely D3.3 BRIDGET System Architecture and Interfaces – Version B; it will focus on the BRIDGET environment, completing the engines description with APIs, and describing in details the interaction of interconnected parts of the system, according to the progress in the project development.

The BRIDGET project will contribute to ISO/IEC MPEG-M standard, proposing several more new MXM Technology Engines providing functionalities such as extraction and compression of image descriptors for visual search (CDVS) and defining the related APIs as suitable extensions to MPEG-M Part 2.
10 References

    -- Part 1: Architecture
    -- Part 2: MPEG extensible middleware (MXM) API
    -- Part 3: Conformance and reference software
    -- Part 4: Elementary services
    -- Part 5: Service aggregation
    Universidad Politécnica de Madrid, University of Surrey and Visual Atoms, Proposal for a
    definition of CDVS engine APIs, Submission to MPEG 108, M33380, March 2014
    N14253
    11: Scene description and application engine
    processing — The Virtual Reality Modeling Language — Part 1: Functional specification
    and UTF-8 encoding
[15] ISO/IEC DIS 16262 Information technology - ECMAScript: A general purpose, cross-
    platform programming language
    Profile (AVDP)", Vienna, Austria, August 2013 (available at
    http://mpeg.chiariglione.org/sites/default/files/files/standards/docs/w13869%20_AVDP
    _WhitePaper_final.docx)
[17] EBU Core Metadata Set, EBU Tech 3293,
    Interoperable Applications”, 2013.