BRIDGET System Architecture and Interfaces
Version B
Period Covered 2015/12/01–2016/01/31

<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>31 January 2016</td>
</tr>
<tr>
<td>Actual date of delivery</td>
<td>29 January 2016</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>
### Deliverable Identification Sheet

<table>
<thead>
<tr>
<th><strong>Project ref. no.</strong></th>
<th>FP7-ICT-610691</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project acronym</strong></td>
<td>BRIDGET</td>
</tr>
<tr>
<td><strong>Project full title</strong></td>
<td>BRIDging the Gap for Enhanced broadcastT</td>
</tr>
<tr>
<td><strong>Document name</strong></td>
<td>BRIDGET System Architecture and Interfaces – Version A</td>
</tr>
<tr>
<td><strong>Security (distribution level)</strong></td>
<td>CO/PU</td>
</tr>
<tr>
<td><strong>Contractual date of delivery</strong></td>
<td>31/01/2016</td>
</tr>
<tr>
<td><strong>Actual date of delivery</strong></td>
<td>29/01/2016</td>
</tr>
<tr>
<td><strong>Document number</strong></td>
<td>D3.3</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Final</td>
</tr>
<tr>
<td><strong>Status &amp; version</strong></td>
<td>Final</td>
</tr>
<tr>
<td><strong>Number of pages</strong></td>
<td>40</td>
</tr>
<tr>
<td><strong>WP / Task responsible</strong></td>
<td>WP3</td>
</tr>
<tr>
<td><strong>Other contributors</strong></td>
<td>Gianmatteo Perrone, Diego Gibellino, Leonardo Chiariglione, Alberto Messina, Milos Markovic, Ingo Feldmann, Adrian Gabrielli, Christian Tulvan, Veronica Scurtu, Marius Preda, Stavros Paschalakis, Miroslaw Bober</td>
</tr>
<tr>
<td><strong>Project Officer</strong></td>
<td>Alberto Rabbachin</td>
</tr>
<tr>
<td><strong>Abstract</strong></td>
<td>This document describes the final version of the BRIDGET system architecture as well as the final infrastructure of the system according to the list of functional requirements organised and presented in D2.5 Version B [2] and to the development occurred until month 27 of the project.</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>Bridget, Components, Architecture, Interface, API, Media Linking Application Format, Infrastructure, Auditing</td>
</tr>
<tr>
<td><strong>Sent to peer reviewer</strong></td>
<td>29 January 2016</td>
</tr>
<tr>
<td><strong>Peer review completed</strong></td>
<td>29 January 2016</td>
</tr>
<tr>
<td><strong>Circulated to partners</strong></td>
<td>29 January 2016</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>0.1</td>
<td>2016-01-02</td>
</tr>
<tr>
<td>0.2</td>
<td>2016-01-15</td>
</tr>
<tr>
<td>0.3</td>
<td>2016-01-26</td>
</tr>
<tr>
<td>0.4</td>
<td>2016-01-27</td>
</tr>
<tr>
<td>0.5</td>
<td>2016-01-29</td>
</tr>
<tr>
<td>1.0</td>
<td>2016-01-29</td>
</tr>
</tbody>
</table>
Contents
1 Executive Summary .................................................................................................................. 5
2 Introduction ................................................................................................................................. 6
3 Glossary ..................................................................................................................................... 6
4 High Level Architecture ............................................................................................................ 7
5 Detailed Logical Architecture .................................................................................................. 10
  5.1 Professional Authoring Peer Logical Architecture ................................................................. 11
  5.2 End User Peer Logical Architecture ....................................................................................... 12
  5.3 End User Authoring Peer Logical Architecture ...................................................................... 13
  5.4 Components description ........................................................................................................ 14
    5.4.1 Media Analysis Component ............................................................................................... 14
    5.4.2 Visual Search Component ................................................................................................ 17
    5.4.3 3D reconstruction Component ......................................................................................... 19
    5.4.4 Fingerprint Extractor Component .................................................................................... 21
    5.4.5 3D Compression Component ........................................................................................... 23
    5.4.6 Synchroniser Component ................................................................................................. 23
6 Detailed Physical Architecture ................................................................................................. 26
  6.1 BRIDGET back-end – Pyramid ............................................................................................ 26
  6.2 BRIDGET end user devices and applications ...................................................................... 30
7 Media Linking Application Format ......................................................................................... 31
  7.1 MLAF Specification Structure .............................................................................................. 31
  7.2 Media Linking Presentation ................................................................................................... 33
    7.2.1 Layout Ref ......................................................................................................................... 34
    7.2.1.1 Syntax .......................................................................................................................... 34
    7.2.1.2 Semantics ...................................................................................................................... 34
    7.2.2 Main Layout Ref ................................................................................................................ 34
    7.2.2.1 Syntax .......................................................................................................................... 34
    7.2.2.2 Semantics ...................................................................................................................... 35
    7.2.3 Bridget Layout Scene ....................................................................................................... 35
    7.2.3.1 Syntax .......................................................................................................................... 35
    7.2.3.2 Semantics ...................................................................................................................... 35
    7.2.4 Bridget Layout Descriptor ............................................................................................... 35
    7.2.4.1 Syntax .......................................................................................................................... 35
    7.2.4.2 Semantics ...................................................................................................................... 35
    7.2.5 Bridget Layout Statement ................................................................................................. 35
    7.2.5.1 Syntax .......................................................................................................................... 35
    7.2.5.2 Semantics ...................................................................................................................... 36
    7.2.6 Main Layout Descriptor ................................................................................................... 36
    7.2.6.1 Syntax .......................................................................................................................... 36
    7.2.6.2 Semantics ...................................................................................................................... 36
    7.2.7 Main Layout Statement ................................................................................................... 37
    7.2.7.1 Syntax .......................................................................................................................... 37
    7.2.7.2 Semantics ...................................................................................................................... 37
8 Conclusions ............................................................................................................................... 37
9 Annex A –BRIDGET auditing schema .................................................................................... 38
  9.1 The BRIDGET auditing schema structure .......................................................................... 38
10 References ............................................................................................................................... 40
1 Executive Summary

This document is a revision of Deliverable D3.1 "BRIDGET System Architecture and Interfaces - Version A" [1]. It substantially updates the system architectural design defined in the early months of the project by taking into account the evolution of the developments occurred until month 27 of the project.

The BRIDGET Engines were defined in [1] as standard components in the reference MPEG-M model, but in order to achieve a more generic and reusable approach have been developed as Generic Technology Component, still interacting with each other in a BRIDGET Middleware, implementing BRIDGET services and applications. This document is mainly focused on the development of the logical and physical architecture updates of these components APIs and additional interfaces to fulfill new BRIDGET requirements and use scenarios that will be implemented in the 3rd project year, provided by D2.5 “BRIDGET Requirements and Functionalities - Version B”[2].

The new End User Authoring Peer Logical Architecture will be described in detail, along with the updates of the Professional Authoring Peer and of the End User Peer relating to the implementation of the two new scenarios selected in [2].
Last, this document will focus on the result of the consortium effort in creating a dedicated format for the internal BRIDGET data model, evolving from the early chosen and adopted MPEG ARAF in [1] to the Media Linking Application Format (MLAF) [3], which has been developed and used as the standard format for bridgets.

2 Introduction

This document describes the final version of the BRIDGET system architecture as well as the final infrastructure of the system according to the list of functional requirements organised and presented in D2.5 Version B [2] and to the development occurred until month 27 of the project.

The document is organised as follows: Chapter 3 provides a list of terms used in the rest of the document, while Chapter 4 describes the BRIDGET high-level architecture, providing an overview of the architectural model implemented. Chapter 5 presents the BRIDGET logical detailed architecture, focusing on the updates of the Professional Authoring Peer Logical Architecture and of the End User Peer Logical Architecture. It also provides a description of the new End User Authoring Peer Logical Architecture, which is directly derived from the D2.5 Version B [2] new functional requirements for the third project year, and a detailed description of each component interacting in the logical architecture domain.

Chapter 6 defines the detailed physical architecture, focusing on BRIDGET back-end and describing client-side devices and application developed.

Chapter 7 introduces the Media Linking Application format (in short MLAF). This format will be published as ISO/IEC International Standard as Part 18 of the ISO/IEC 23000 (MPEG Application Formats – MPEG A, ISO/IEC 23000-18 [3]). It has been defined based on the activities carried out in BRIDGET, and generalized in order to fit many different scenarios that require media content linking, navigation and presentation capabilities.

Finally, chapter 8 draws the conclusions describing the plans for the last year of project, while Chapter 9 provides a list of references mentioned in the document.

Appendix A defines a general auditing schema developed to facilitate the collection of measurements from clients during the fruition of BRIDGET services.

Compared to the previous version, Version B includes the following additional new elements, according to the scenarios and requirements identified and selected for the second part of the Project:

- End User Authoring Peer, a simplified Peer based on a subset of the Professional Authoring Tool functionalities. Allows end users to automatically analyse and tag multimedia content in order to link broadcaster content to other available multimedia content enabling a “Social” dimension to the BRIDGET platform
- Media Linking Application Format including the description of the format structure and the presentation information. In particular, the presentation structure allows possible implementation based on different presentation layers such as BIFS, HTML5, etc.
- Auditing schema, defining a generic format to collect and represent measures from BRIDGET clients

In addition, the following sections have been updated to support the new functionalities and to reflect the actual level of implementation for the BRIDGET platform:

- High Level Architecture
- Detailed Logical Architecture
- The following component descriptions and APIs:
  - 3D Reconstruction Component
  - 3D Compression Component
- Detailed Physical Architecture

3 Glossary

Table 1 - Terms and definitions
### Table: BRIDGET Component Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Compression Component</td>
<td>A Technology Component 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 Component</td>
<td>A Technology Component 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>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>Component</td>
<td>An organised set of technologies communicating with other Components and Applications via API</td>
</tr>
<tr>
<td>Fingerprint Extractor Component</td>
<td>A Technology Component 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>Media Analysis Component</td>
<td>A Technology Component that provides 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.</td>
</tr>
<tr>
<td>Middleware</td>
<td>A collection of components that support the execution of Applications via High Level API</td>
</tr>
<tr>
<td>Synchroniser Component</td>
<td>A Technology Component that monitors an audio stream to detect synchronisation points</td>
</tr>
<tr>
<td>Visual Search Component</td>
<td>A Technology Component that analyses images and videos in the content libraries, extracts compact descriptors (compliant to Compact Descriptors of Visual Search – CDVS standard), and stores in a database a combination of query image, reference image and score</td>
</tr>
</tbody>
</table>

### 4 High Level Architecture

This section describes the BRIDGET architecture from a high-level view. Figure 1 presents the basic model enabled by BRIDGET. 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 technology 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 postproduction environment) or an end user.

Figure 1 - BRIDGET Architectural 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 2. 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 3, while Figure 4 depicts the structure of the BRIDGET Middleware.
The BRIDGET Professional Authoring Peer and the BRIDGET End User Peer include the application logic as well as the BRIDGET middleware components required in order to fulfil all the scenarios selected in [4] and [5].

It is important to notice that during the third project year the adoption of the MLAF management layer will provide End User Peer capabilities to modify Bridget descriptions and link multimedia broadcaster content to other available content, enabling End User to share them on Social Networks through the application logic developed on the End User App. Thus, the End User Peer will implement itself a Simplified Authoring Peer, with limited access to a subset of the Professional database and with access to only a limited set of functionalities easy to execute on mobile terminals (for further details, see D8.6 [6]).

5 Detailed Logical 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.

The application layer interacts with a middleware layer, providing BRIDGET-specific media functionalities exposed as Technology Component through a BRIDGET High Level API. The application logic coordinates and orchestrates the different Component.

The third layer, the Data layer, logically represents the components acting as Data sources or destinations (including media content and metadata) for the system.

In the following sections, we provide the logical architecture diagrams for the Professional Authoring Peer, the End User 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 in charge of the related development. It should be noted that a few components are white coloured, meaning that the component were not entirely developed within the project but reused or adapted. 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.

5.1 Professional Authoring Peer Logical Architecture

The Professional Authoring Peer (Version B) 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 bridget to a defined point (or portion) in the timeline of the programme, thanks to the Fingerprint Extractor Component;
- Consolidation and rendering of editorial decisions concerning bridgets;
- Media content analysis on a set of selected media items, via the Media Analysis Component;
- Content search service to select content to be linked, via the Visual Search Component;
- 3D reconstruction on a set of selected 2D media items, via the 3D Reconstruction Component.

Figure 5 describes the component, main interfaces and the interactions between components for the Professional Authoring Peer.

Table 2 provides the list of Components used by the Professional Authoring Peer developed for Version B. For each component, 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 2 - Professional Authoring Peer Components

<table>
<thead>
<tr>
<th>Component</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>5.4.1</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>5.4.2</td>
<td>WP5</td>
</tr>
<tr>
<td>3D Reconstruction</td>
<td>creates a 3D model from input images and/or videos</td>
<td>5.4.3</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>5.4.4</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>5.4.5</td>
<td>WP6</td>
</tr>
</tbody>
</table>

5.2 End User Peer Logical Architecture

The End User Peer (Version B) is a system that 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 Component;
- Media aligned and global bridgets bookmarking for future use;

Figure 6 describes the component, main interfaces and the interactions between components for the End User Peer (Version B).
Table 2 provides the list of Components used by the End User Peer developed for Version B. For each Component 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 3- End User Peer Components**

<table>
<thead>
<tr>
<th>Component</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>5.4.5</td>
<td>WP6</td>
</tr>
<tr>
<td>Synchroniser</td>
<td>Monitors an audio stream to detect synchronisation points</td>
<td>5.4.6</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>5.4.4</td>
<td>WP7</td>
</tr>
</tbody>
</table>

### 5.3 End User Authoring Peer Logical Architecture

The End User Authoring Peer (Version B) is a system which provides End Users a simplified subset of the AT functionalities, allowing them to modify Bridget descriptions and tag multimedia content: thus, broadcaster content can be linked to other available multimedia content and, potentially, shared on social networks. This limited set of Authoring functionalities is enabled by the adoption of the Media Linking Application Format, further described in section 7 of the present document, and provides End User the following high-level functionalities:

- Answering a questionnaires and optionally sending the answers back to the source;
- Editing the bridget replacing destination content, adding new destination content at an already defined positions etc;
- Distributing the original or edited bridget to a group of people on social networks.
The End User Authoring Peer has the same component, main interfaces and interactions between components of the End User Peer described above.

5.4 Components description
The following sub-sections describe the different components identified for Version B.

5.4.1 Media Analysis Component

5.4.1.1 Description
The purpose of the Media Analysis Component 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 Component and 3D Reconstruction Component, by pre-filtering their input and/or post-processing their results.

5.4.1.2 Interfaces and API
The envisaged interfaces and API between the Media Analysis Component and other parts of the BRIDGET system are simple. At the most basic level, the various modules of the component will take a media file as input and produce standardised output files. MPEG-7 AVDP [7] is XML-based and is specifically designed to carry the type of results that the Media Analysis Component 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 component or, through a proxy component, directly by the extended GPAC Media Framework.

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:
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 module of the component:

### Table 4

<table>
<thead>
<tr>
<th>Internal module</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 Component</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>
5.4.1.3 Face Clustering and Media Segmentation

Face Clustering and Media Segmentation components of WP4 are integrated through a simple mechanism based on a shared folder on the network. Whenever a new file is imported through the BRIDGET Authoring Tool, a copy of the file is put in a special directory. This directory is polled by the Face Clustering and Media Segmentation component (FCMS) for new files. Once detected, each new file gets transferred into a repository near the platform on which the FCMS component is deployed. The resulting data are delivered in the same folder in AVDP format.

Segmentation information is rendered in AVDP by using a TemporalDecomposition element of the main AudioVisual element.

```xml
<mpeg7:AudioVisual id="cs_AVID_1">
  <mpeg7:StructuralUnit href="http://www.ebu.ch/metadata/cs/mpeg/avdp/StructuralUnitCS#1"/>
  <mpeg7:MediaTime>
    <mpeg7:MediaRelIncrTimePoint>0</mpeg7:MediaRelIncrTimePoint>
  </mpeg7:MediaTime>
  <mpeg7:TemporalDecomposition criteria="http://www.ebu.ch/metadata/cs/mpeg/avdp/DecompositionCS#50" id="computational_scene_detection">
    <mpeg7:Header xsi:type="mpeg7:DescriptionMetadataType">
      <mpeg7:Comment>
        <mpeg7:FreeTextAnnotation>START/TALK/PERFORMANCE/INSERT/SEPARATOR/END segmentation based on audio and video analysis</mpeg7:FreeTextAnnotation>
      </mpeg7:Comment>
      <mpeg7:Instrument>
        <mpeg7:Tool>
          <mpeg7:Name>eu.bridget.rai.wp4.programmestructuring.hmm.ComputationalSceneDetector</mpeg7:Name>
        </mpeg7:Tool>
      </mpeg7:Instrument>
    </mpeg7:Header>
  </mpeg7:TemporalDecomposition>
  <mpeg7:AudioVisualSegment id="cs_AV_0">
    <mpeg7:Header xsi:type="mpeg7:DescriptionMetadataType">
      <mpeg7:Comment>
        <mpeg7:FreeTextAnnotation>start</mpeg7:FreeTextAnnotation>
      </mpeg7:Comment>
    </mpeg7:Header>
    <mpeg7:MediaTime>
      <mpeg7:MediaRelIncrTimePoint>0</mpeg7:MediaRelIncrTimePoint>
      <mpeg7:MediaIncrDuration>18</mpeg7:MediaIncrDuration>
    </mpeg7:MediaTime>
  </mpeg7:AudioVisualSegment>
  <mpeg7:AudioVisualSegment id="cs_AV_1">
    <mpeg7:Header xsi:type="mpeg7:DescriptionMetadataType">
      <mpeg7:Comment>
        <mpeg7:FreeTextAnnotation>separator</mpeg7:FreeTextAnnotation>
      </mpeg7:Comment>
    </mpeg7:Header>
    <mpeg7:MediaTime>
      <mpeg7:MediaRelIncrTimePoint>18</mpeg7:MediaRelIncrTimePoint>
      <mpeg7:MediaIncrDuration>71</mpeg7:MediaIncrDuration>
    </mpeg7:MediaTime>
  </mpeg7:AudioVisualSegment>
</mpeg7:AudioVisual>
```

Figure 9 - Top-level decomposition for segmentation data.

```xml
<mpeg7:AudioVisualSegment id="cs_AV_0">
  <mpeg7:Header xsi:type="mpeg7:DescriptionMetadataType">
    <mpeg7:Comment>
      <mpeg7:FreeTextAnnotation>start</mpeg7:FreeTextAnnotation>
    </mpeg7:Comment>
  </mpeg7:Header>
  <mpeg7:MediaTime>
    <mpeg7:MediaRelIncrTimePoint>18</mpeg7:MediaRelIncrTimePoint>
    <mpeg7:MediaIncrDuration>71</mpeg7:MediaIncrDuration>
  </mpeg7:MediaTime>
</mpeg7:AudioVisualSegment>
```

Figure 10 - An example of segmentation. Two segments: one start and one separator.
Face clustering information is rendered in AVDP by using a **HierarchicalSummary**. **SummarySegmentGroups** inside the summary represent face clusters. For each cluster a representative video clip (a face) is also included as reference to the originating video.

**Figure 11** - Top-level summarization for face clustering data.

**Figure 12** - An example of Summary Segment Group containing two face segments.

### 5.4.1.4 Interactions with other Components

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

Media parsing and decoding functionalities may reuse functionalities provided directly by the extended **GPAC Media Framework**.

Additional APIs may be defined between the **Media Analysis Component** and other components, for direct access to the results of the analysis, as needed.

### 5.4.2 Visual Search Component

#### 5.4.2.1 Description

**Visual Search Component** 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 component 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 Component** will inform the aggregation process. In the search mode, the component 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 Component will be used for scene classification tasks by the Media Analysis Component.

5.4.2.2 Interfaces and API

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

In the analysis mode, the component 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 Component will take a spatial region in an image and return ranked matching images or frames present in the media library. The component 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 component or, through a proxy component, directly by the extended GPAC Media Framework. The following table shows the expected inputs and outputs for each internal module of the component:

<table>
<thead>
<tr>
<th>Internal module</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 localisation and confidence (XML)</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></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 localisations and confidence (XML)</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Media library descriptors</td>
<td></td>
</tr>
<tr>
<td>Local descriptor (feature point) extraction (for Media Analysis Component)</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>
<tr>
<td>Local descriptor matching for 3D reconstruction</td>
<td>• Media file (MPEG-4)(image sequence) + optional ROI</td>
<td>Feature points and matching information (XML)</td>
</tr>
<tr>
<td></td>
<td>• Configuration parameters</td>
<td></td>
</tr>
</tbody>
</table>
5.4.2.3 Interactions with other Components

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

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

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

The 3D reconstruction Component will interface to Visual Search Component 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 Component and other components, for direct access to the results of the Visual Search Component, as needed.

Media parsing and decoding functionalities may reuse functionalities provided directly by the extended GPAC Media Framework.

5.4.3 3D reconstruction Component

5.4.3.1 Description

The purpose of the 3D Reconstruction Component is to create a 3D model from input images and/or videos. The resulting 3D model is either of a point-cloud-based structure or of a mesh-based structure. Further on, a hybrid version as combination of point-cloud-based and mesh-based structure is possible. As this is the most general representation form it will be used in the following description.

For the general 3D media analysis, the generated hybrid 3D model is provided as output to the rendering tools. Further on, the 3D Reconstruction Component will tell the user where the input images were taken from with respect to a 3D coordinate system, i.e. a 3D world representation.

5.4.3.2 Interfaces and API

The 3D Reconstruction Component requires one interface with the functionalities listed below. 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 6:

<table>
<thead>
<tr>
<th>Internal module</th>
<th>Content</th>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DReconstruction</td>
<td></td>
<td>create3DModel</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• input: ImageSequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• output: Hybrid3DModel, CameraCalibration Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>refine3DModel</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• input: ImageSequence, Hybrid3DModel, CameraCalibration Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• output: Hybrid3DModel, CameraCalibration Vector</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: API / interface to call the 3D Reconstruction Component functionalities
<table>
<thead>
<tr>
<th>Internal module</th>
<th>Content</th>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageSequence</td>
<td>• Raw image data or IDs to image data or file names of image data</td>
<td></td>
<td>structure required for wrapping data and parameter for the API / interface</td>
</tr>
<tr>
<td></td>
<td>• sequence IDs image/sequence metadata (EXIF information etc...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hybrid3dModel</td>
<td>• Raw 3D point cloud data or file names where those are stored</td>
<td></td>
<td>structure required for wrapping data and parameter for the API / interface</td>
</tr>
<tr>
<td></td>
<td>• Raw 3D splatting data or file names where those are stored</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Raw simplified 3D model data or file names where those are stored</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Raw 3D mesh cloud data or file names where those are stored</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>CameraCalibration</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>CameraCalibration 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>
</tbody>
</table>
### 5.4.3.3 Interactions with other Components

The following interactions with other components are required.

- **Media Analysis Component**
  - provide audiovisual quality metadata.

- **Visual Search Component (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 Component**
  - provide efficient [de]compression of the reconstructed 3D models.

### 5.4.4 Fingerprint Extractor Component

#### 5.4.4.1 Description

The **Fingerprint Extractor Component** 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 Component** is accessed by the application logic. In the End Used Peer, the **Synchroniser Component** utilises the **Fingerprint Extractor Component** to continuously compute AFPS from the input received by the built-in microphone.

The Fingerprint Extractor Component aggregates several internal functions (Figure 13):

- 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.

---

<table>
<thead>
<tr>
<th>Internal module</th>
<th>Content</th>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
</table>
| ShapeInformation | • Raw shape information  
• sequence/image IDs relevant to map information to ImageSequence class | | |
5.4.4.2 Interfaces and API

The envisaged interfaces and API between the Fingerprint Extractor Component and the other parts of the BRIDGET system are extremely simple. In particular, Fingerprint Extractor Component 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 module of the component:

<table>
<thead>
<tr>
<th>Internal module</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>

5.4.4.3 Interactions with other Components

On the Professional Authoring Peer, the Fingerprint Extractor Component 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 Component.

5.4.5 3D Compression Component

5.4.5.1 Description

The 3D Compression Component 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 Component, 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.

5.4.5.2 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.

5.4.5.3 Interactions with other Components

- 3D Reconstruction Component
  - provide reconstructed 3D models.

5.4.6 Synchroniser Component

5.4.6.1 Description

The Synchroniser Component 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 Component is interfaced to the Fingerprint Extractor Component, acting as a wrapper between such module and the application logic.
The **Synchroniser Component** 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 bridges.

The architecture of the **Synchroniser Component** can be represented as follows:

![Figure 15 - Synchroniser 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 Component** 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.

### 5.4.6.2 Interfaces and API

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

```c
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 Component** when **findTrigger** procedure is executed is shown in the next figure:
The audio captured by the built-in microphone is transferred by the application logic to the Synchroniser Component: 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 Component 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 module of the component:

<table>
<thead>
<tr>
<th>Internal module</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 bridget</td>
</tr>
</tbody>
</table>

5.4.6.3 Interactions with other Components

In the End User Peer, the Synchroniser Component is interfaced with two components, namely the application logic and the Fingerprint Extractor Component.
6 Detailed Physical Architecture

The physical architecture 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 are deployed on every node, the scheduler should chose 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).

6.1 BRIDGET back-end – Pyramid

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 proposed 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.
As shown in Figure 17, 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.

As shown in Figure 18, 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 16, 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 9

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Minimal</th>
<th>Amount</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cores</td>
<td>2 x Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz</td>
<td>4 x Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz</td>
<td></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 sse3 cx16 sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes hypervisor lahf_lm</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 sse3 cx16 sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes hypervisor lahf_lm</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>4 GB</td>
<td>8 GB</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>1 x GBit</td>
<td>1 x GBit</td>
<td></td>
</tr>
<tr>
<td>Persistent Storage space (shared)</td>
<td>1 TB</td>
<td>4 TB</td>
<td></td>
</tr>
<tr>
<td>Characteristic</td>
<td>Amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimal</td>
<td>Optimal</td>
<td></td>
</tr>
<tr>
<td>Persistent Storage space read</td>
<td>30.00 MB/sec</td>
<td>100.00 MB/sec</td>
<td></td>
</tr>
<tr>
<td>performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistent Storage space write</td>
<td>20.00 MB/sec</td>
<td>60.00 MB/sec</td>
<td></td>
</tr>
<tr>
<td>performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scratch Storage Space (volatile)</td>
<td>20 GB</td>
<td>80 GB</td>
<td></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:

![Software package deployment structure](image)

In general, a node is not directly accessible from outside the virtual cluster (Figure 21). 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 21 - Virtual Cluster - Nodes network visibility**

### 6.2 BRIDGET end user devices and applications

Within BRIDGET, we mainly focused 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 10**

<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>

**Figure 22 - A picture of Huawei Media Pad M1 tablet**
The end user’s BRIDGET Application (or hereafter simply referred to as the BRIDGET Application) is an Android application designed to make end users enjoy the BRIDGET experience. The architecture of this application is shown in Figure 23.

The application is composed of four main components:

- The Launcher, i.e. a dashboard for configuring the BRIDGET experience and subscribing to the enriched programmes
- The Synchroniser Component, able to identify the programme being broadcasted on the main screen and the temporal instant on the media timeline of the programme
- The Bridget Player, i.e. a module able to show the bridget information and the associated destination content, according to the decisions taken by the author. It is also able to interact with the backend visual search service provided by the broadcaster
- The Simplified Authoring Tool, i.e. a simplified version of the BRIDGET Professional Authoring Tool, aimed to support basic authoring functionalities at the user side.

7 Media Linking Application Format

7.1 MLAF Specification Structure

Media Linking Application Format (in short MLAF) is a standard of the MPAG family in the suite ISO/IEC 23000 (MPEG Application Formats – MPEG A). Once standard, MLAF will be Part 18 of MPEG-A. MLAF is structured as a restriction of MPEG-21 DIDL [10] including some key descriptors from MPEG-7 and other parts of MPEG-21. MLAF is basically the standard format for bridgets. This section briefly outlines the structure of such specification. For full reference and details please refer to[8].

The main concept of MLAF consists in modelling bridgets-related information as a hierarchy of DIDL elements. The topmost (root) Container is the wrapper of the whole linking information between a certain Digital Item, the source item, and its bridgets. Each of the former and the latter is modelled as a specific restriction of the generic DIDL Item (Figure 24).
Figure 24 - Top Level MLAF structure.

The Bridgets Item contains the list of bridges associated to the Source Item and information about the intended presentation Layout, expressed in a restricted dialect of MPEG-4 xmta[9] (Figure 25).

Figure 25 - MLAF Bridgets structure.

Each Bridget is considered as a DIDL Item as well and contains a reference to the spatiotemporal portion of the Source Item where it is active (i.e., it defines the segments of the timeline and the picture region of the Source item associated to the Bridget information which shall be shown on the rendering device). This information is wrapped in MPEG-21 Annotation structure and is implemented making use of MPEG-7 Moving Region and AudioVisual Segment structures[11]. The Bridget structure may also contain information about the recommended usage of the Bridget information, implemented by making
reference to MPEG-21 part 22 (User Description) Recommendation Information [13]. Naturally, the Bridget item contains the reference to one or more destination items, i.e. the Digital Item used as enrichment of the main Source Item in the specified spatiotemporal portion. Since a bridget is a result of an editorial work, each Bridget can be described making use of an EBU Core Descriptor and represented by a multimedia element. See also Figure 26 for a complete rendition of the Bridget Structure.

\[\text{Figure 26 - Bridget structure in MLAF.}\]

Finally, the Source Item contains descriptive information in form of MPEG-7 or EBU Core descriptors [12]. Each of these descriptors are modelled as restrictions of the MPEG-21 DIDL Descriptor (Figure 27).

\[\text{Figure 27 - Source Item Structure.}\]

### 7.2 Media Linking Presentation

The Bridget Presentation is capable of representing 2D and 3D scenes that may contain multimedia content such as 2D and/or 3D graphic elements, images, videos and sounds. The multimedia content can be part of the scene (stored locally) or it can be referenced (through hyper-links) from external repositories where compliant media files are available. Moreover, a scene can be static or animated and can connect to sensory information that can be either local or remote. The Bridget Presentation supports external interaction allowing the user to communicate with the scene elements and therefore affecting the scene behaviour. The actions that can happen in a scene can be programatically triggered by using
time sensors, routes and scripting while other actions can be triggered by the user interaction through touch sensors.

7.2.1 Layout Ref

Specifies a bridget layout which can be either the bridged media layout itself (scene graph) or a URI to a file where the layout is described.

7.2.1.1 Syntax

```
<xs:complexType name="LayoutRefType">
  <xs:annotation>
    <xs:documentation>This complex type defines the layout presentation of a Bridget destination</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="BridgetLayoutUrl">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:anyURI">
            <xs:attribute name="mimeType" type="xs:string"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="BridgetLayoutScene" type="BridgetLayoutSceneType"/>
  </xs:sequence>
</xs:complexType>
```

7.2.1.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BridgetLayoutUrl</td>
<td>An element of type xs:anyURI pointing to a file where a bridget presentation is described.</td>
</tr>
<tr>
<td>@mimeType</td>
<td>The MIME type of the file where the bridget presentation is described (e.g., text/html)</td>
</tr>
<tr>
<td>BridgetLayoutScene</td>
<td>An element of type BridgetLayoutSceneType where the description of a bridget presentation can be defined locally.</td>
</tr>
</tbody>
</table>

7.2.2 Main Layout Ref

Specifies an element that allows a bridget player to link a bridget layout representation to the main scene graph of a bridged programme, as a subset of it.

7.2.2.1 Syntax

```
<xs:complexType name="MainLayoutRefType">
  <xs:annotation>
    <xs:documentation>This complex type links the bridget layout representation to the main programme layout, where the bridget presentation takes place</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="MainLayoutUrl" type="mlaf:Inline"/>
    <xs:element name="MainLayoutScene" type="mlaf:TransformType"/>
  </xs:sequence>
</xs:complexType>
```
7.2.2.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainLayoutUrl</td>
<td>An element of type mlaf:Inline that can inline a bridget layout representation to the main scene graph of the bridgeted programme.</td>
</tr>
<tr>
<td>MainLayoutScene</td>
<td>An element of type mlaf:TransformType that allows to connect a bridget layout representation to the main scene graph of the bridgeted programme.</td>
</tr>
</tbody>
</table>

7.2.3 Bridget Layout Scene

Describes how a bridget layout should be represented in order for a bridget player to be able to interpret the information and present the bridget. The graphical elements used to describe a bridget layout are defined in the attached XML schema and they are a subset of the elements defined in ISO/IEC 14496-11.

7.2.3.1 Syntax

```xml
<xs:complexType name="BridgetLayoutSceneType">
  <xs:annotation>
    <xs:documentation>This complex type defines the XMT layouting information of a Bridget</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element ref="mlaf:Scene"/>
  </xs:sequence>
</xs:complexType>
```

7.2.3.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlaf:Scene</td>
<td>An element (a structure of elements) derived from the type xmta:Scene defined in ISO/IEC 14496-11 by restricting the set of elements.</td>
</tr>
</tbody>
</table>

7.2.4 Bridget Layout Descriptor

A restriction of MPEG-21 DIDL Descriptor representing a bridget Layout information.

7.2.4.1 Syntax

```xml
<xs:complexType name="BridgetLayoutDescriptorType">
  <xs:complexContent>
    <xs:restriction base="BridgetDescriptorType">
      <xs:sequence>
        <xs:element ref="didmodel:Condition" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="BridgetLayoutStatement"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

7.2.4.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>didmodel:Condition</td>
<td>As defined in MPEG-21 Part 2</td>
</tr>
<tr>
<td>BridgetLayoutStatement</td>
<td>A bridget Layout Statement</td>
</tr>
</tbody>
</table>

7.2.5 Bridget Layout Statement

A restriction of MPEG-21 DIDL Statement carrying bridget Layout descriptors.
7.2.5.1 Syntax

```xml
<xs:complexType name="BridgetLayoutStatementType">
  <xs:complexContent>
    <xs:restriction base="didl:StatementType">
      <xs:sequence>
        <xs:element name="BridgetLayout" type="LayoutRefType"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="BridgetLayoutStatement" type="BridgetLayoutStatementType" substitutionGroup="didl:Statement"/>
```

7.2.5.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BridgetLayout</td>
<td>An element of type LayoutRefType</td>
</tr>
</tbody>
</table>

7.2.6 Main Layout Descriptor

A restriction of MPEG-21 DIDL Descriptor representing a Main bridget Layout information.

7.2.6.1 Syntax

```xml
<xs:complexType name="MainLayoutDescriptorType">
  <xs:complexContent>
    <xs:restriction base="didl:DescriptorType">
      <xs:sequence>
        <xs:element ref="didmodel:Condition" minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice>
          <xs:element ref="MainLayoutStatement"/>
        </xs:choice>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="MainLayoutDescriptor" type="MainLayoutDescriptorType" substitutionGroup="didl:Descriptor"/>
```

7.2.6.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>didmodel:Condition</td>
<td>As defined in MPEG-21 Part 2</td>
</tr>
<tr>
<td>MainLayoutStatement</td>
<td>A Main Layout Statement</td>
</tr>
</tbody>
</table>
7.2.7 Main Layout Statement

A restriction of MPEG-21 DIDL Statement carrying Main bridge Layout descriptors.

7.2.7.1 Syntax

```xml
<xs:complexType name="MainLayoutStatementType">
  <xs:complexContent>
    <xs:restriction base="didl:StatementType">
      <xs:sequence>
        <xs:element name="MainLayout">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="LayoutRefType">
                <xs:choice>
                  <xs:element name="MainLayoutUri" type="mlaf:InlineType"/>
                  <xs:element name="MainLayoutScene" type="mlaf:TransformType"/>
                </xs:choice>
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

7.2.7.2 Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainLayout</td>
<td>An element containing URI or inline definition of Main Layout</td>
</tr>
<tr>
<td>MainLayoutUri</td>
<td>An URI pointing to a Main Layout</td>
</tr>
<tr>
<td>MainLayoutScene</td>
<td>Inline description of Main Layout</td>
</tr>
</tbody>
</table>

8 Conclusions

This document is a major update to the D3.1 BRIDGET System Architecture and Interfaces – Version A, and describes the final system architecture and infrastructure of BRIDGET, providing an overview of its distributed infrastructure components, internal modules and interfaces.

It describes the evolution of the architecture during this first two year of project, completing the Components description with the related APIs and describing in details the interaction of interconnected parts of the system, according to the project development.

Moreover, it shows the evolution of the BRIDGET environment in order to satisfy all the scenarios selected in [4] and [5], introducing the new End User Authoring Peer and focusing on the importance of the adoption of the Media Linking Application Format.

Finally, a BRIDGET Auditing schema is proposed, defining a generic format to collect and represent measures from BRIDGET clients.
9 Annex A – BRIDGET auditing schema

In this section we propose a possible auditing schema that may be refined during the third year of project by the technical WPs, in order to trace and collect information about End User’s behaviour and usage of the End User Application.

9.1 The BRIDGET auditing schema structure

The BRIDGET auditing schema has been designed in order to provide detailed information to BRIDGET back-end system about End User’s behaviour. The collected information can be eventually stored in a BRIDGET trace collector on the Bridget Content Storage server and used to generate analytics that could be monetized, drive the improvement of the application itself, or provide support for the application debugging.

The BRIDGET tracing will be generated on the End User Application in a XML language, and sent via Get/Post calls on HTTP protocol to the BRIDGET trace collector.

Figure 28 describe the structure of the XML trace schema detailed below.

```
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="applicationTracing">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="general" />
        <xs:element ref="measure" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="general">
    <xs:complexType>
      <xs:attribute name="mcaddress" type="xs:string" use="optional" />  
      <xs:attribute name="hardwareModel" type="xs:string" use="optional" />  
      <xs:attribute name="hardwareName" type="xs:string" use="optional" />  
      <xs:attribute name="softwareVersion" type="xs:string" use="optional" />  
      <xs:attribute name="application" type="xs:string" use="optional" />  
      <xs:attribute name="hardwareType" type="xs:string" use="optional" />  
      <xs:attribute name="IP" type="xs:string" use="optional" />  
    </xs:complexType>
  </xs:element>
  <xs:element name="measure">
    <xs:complexType>
      <!-- Define the structure of measure -->
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 28 - BRIDGET tracing XML schema.
The root element applicationTracing is made up of one general element and one or more measure element. The general element provides specific client application parameters, and its attributes are extensible if needed without any modification to the XML schema. The measure elements provide detailed information through their specific attributes name (what we are measuring), time (how long does the action go on) and a sequence of param. Every applicationTracing provides at least one measure with name=StartTracing and one measure con name=StopTracing.

The schema proposed is very flexible and it should easily adapt to the present and future Bridget tracing purposes.

We briefly provide a sample of a trace that could be generated by the End User's application through the selection of one bridget related to the Gulp Girl scenario defined in [4].

```xml
<applicationTracing>
  <general softwareVersion="2.01" mcaddress="00:00:00:00:00:00" hardwareModel="P8" hardwareName="Huawei" application="Bridget" hardwareType="Smartphone IP=" 80.20.149.215 " />
  <measure name="StartTracing " time="22/01/2016-12:19:34.355 " />
  <measure name="SpecificTracingEvent " time="22/01/2016-12:23:14.277 ">
    <param name="contentProvider " value="RAI " />
    <param name="section" value="GULP GIRL " />
    <param name="action" value="bridgetSelected" />
    <param name="assetID" value="bridgetInternalId_50245879" />
    <param name="contentTitle " value="hilary_swank_1 " />
  </measure>
  <measure name="StopTracing " time="22/01/2016-12:24:01.090 " />
</applicationTracing>
```
10 References

[12] EBU Tech 3293, EBU Core Metadata Set ver. 1.5