D5.5 Protocols and APIs Implementation

Grant Agreement nr	856879
Project acronym	PRESENT
Project start date (duration)	September 1st 2019 (36 months)
Document due:	August 31st 2021
Actual delivery date	August 31st 2021
Leader	Framestore
Reply to	Richard.Ollosson@framestore.com
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	856879
Project acronym	PRESENT
Project full title	**P**hotoreal **RE**altime **S**entient **ENT**ity
Document name	Protocols and APIs Implementation
Security (distribution level)	Public
Contractual date of delivery	31/08/2021
Actual date of delivery	31/08/2021
Deliverable name	D5.5 Protocols and APIs Implementation
Type	Other
Status & version	Submission Version
Number of pages	18
WP / Task responsible	Framestore
Other contributors	all partners
Author(s)	Theo Jones, Richard Ollosson
EC Project Officer	Ms. Adelina Cornelia DINU - Adelina-Cornelia.DINU@ec.europa.eu
Abstract	This deliverable presents the overall structure of the reference implementation as well as the protocols and specifications developers can use to successfully contribute components to the reference implementation project.
Keywords	digital human, real-time, Unreal Engine
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 10/07/2021	Document created by Theo Jones
1.1 15/08/2021	Version shared for internal peer review
1.2 27/08/2021	Final version for submission

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

As part of the overall project architecture, as presented in D2.2 and D2.3, an approach to component integration was agreed that utilises a reference implementation standard in the form of an Unreal Engine project. This reference implementation project provides a reference platform against which i) all input components can be tested and validated and ii) provides a template for the integration of consortium partner components.

This deliverable presents the overall structure of the reference implementation as well as the protocols and specifications developers can use to successfully contribute components to the reference implementation project.

A detailed breakdown of the various components within the reference implementation project are also presented. This breakdown includes details of the placeholder digital human and user representations and how these can be used for interface and testing purposes.

Examples are also provided of the use case functionality that has already been incorporated in the reference implementation in conjunction with the consortium partners.

A help file with fully documented and commented code for all the plugins is included as part of this deliverable and should provide collaborating developers with all the information they need to contribute compatible code in the form of new plugins and dll's.

This document is written to accompany version 3.2.2 of the reference implementation.

2. INTRODUCTION

This deliverable outlines both the design and structure of the reference implementation along with how-to information on adding new components and plugins to the project.

What is the reference implementation?

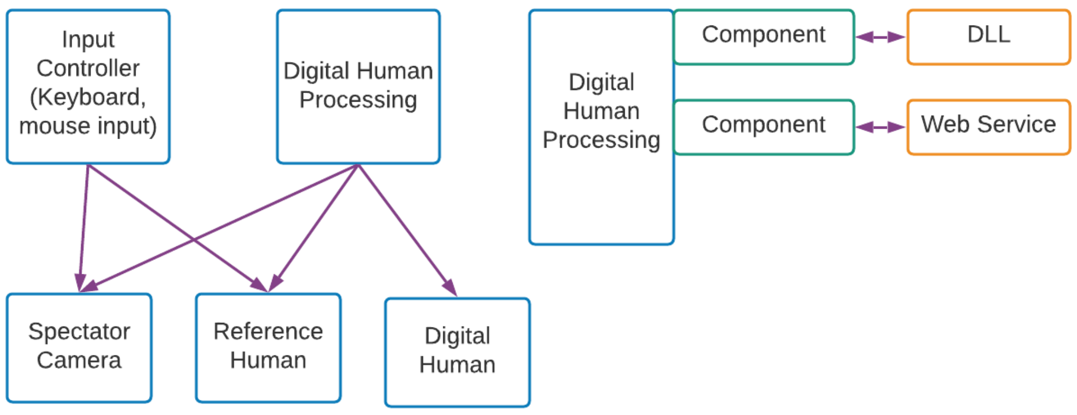
- It is an Unreal project that can be used to share code, data and services between partners.
- The functionality of partners' services are exposed in Unreal as separate components that are implemented in different plugins.
- Partners can access each other's services by simply calling functions in Unreal blueprints.
- Each use case has its own map in Unreal and specific use case functionality can be implemented in both C++ and blueprints.

The documentation below should provide all the information required to successfully contribute new components and interface those components with functionality contributed by other partners. This process has already been successfully undertaken for a number of the use-cases present in the project, such as the Adam mirroring use case and the Sports Broadcast Analysis use case.

3. DOCUMENTATION

3.1 - How is the reference implementation structured?

- There are three main elements in the project: The *DigitalHuman*, the *ReferenceHuman* and the *DigitalHumanProcessing*. The diagram below shows the relationships between these elements and the detail of the *DigitalHumanProcessing*.

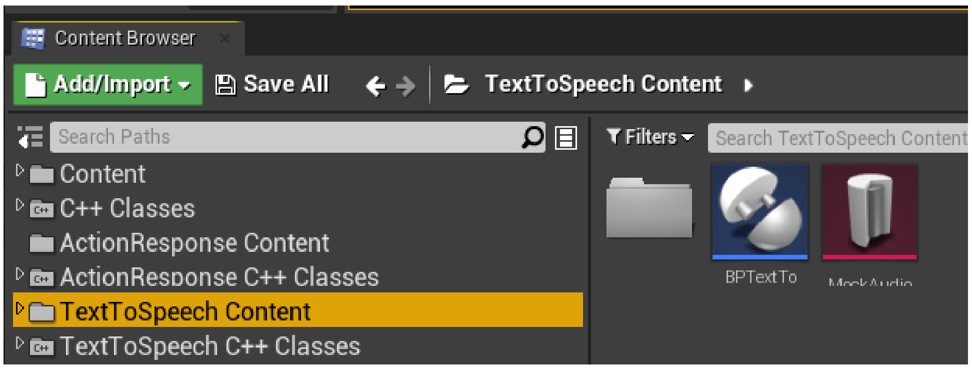


- The *InputController* handles user inputs from keyboard and mouse as a stub implementation. It is aimed to give the user control of the camera(s) and the reference human. This input can get replaced by specific use case input devices, eg VR headset.

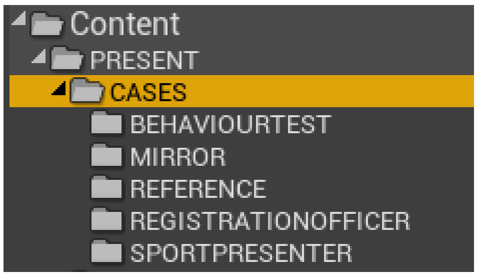
- The *SpectatorCamera* sees the scene from a third person point of view. It can be controlled by keyboard / mouse or blueprints. It is possible to add multiple spectator cameras and switch between them during runtime.
- The *ReferenceHuman* is a representation of the user. It has functions that can be called by the *InputController,* or used by the *DigitalHumanProcessing* in a particular use case.
- The *DigitalHuman* represents the main agent or several agents in a crowd simulation. It receives events and function calls from *DigitalHumanProcessing*.
- The *DigitalHumanProcessing* contains all the components that define services, code and data from the partners. It is the chief manipulator of the *DigitalHuman* instances. The components are defined in separate plugins. The components can internally contain a local C++ implementation, a DLL or a web service client. The DLL or client code is not visible to the *DigitalHumanProcessing*, but components expose functions that call the internal APIs.

3.2 - How is the content organized?

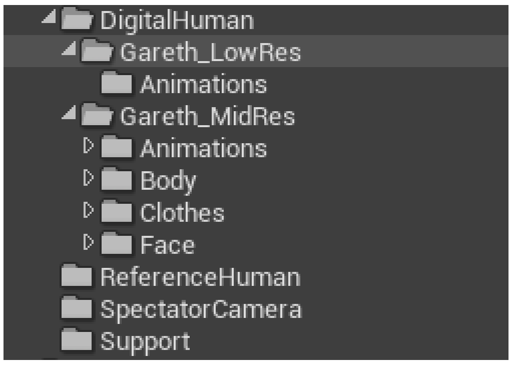
- The partner's actor components are developed as plugins, so the content (i.e. assets) for each component is located in the plugin content folder. (*View Options→Show Plugin Content*)



- There are also case folders that contain assets for each use case. This could be the main map, specific data assets or blueprints that are specific to that use case. This helps facilitate data between partners.



- The current case folders are:
  - REFERENCE: Contains an example test map with all actors and a DigitalHumanProcessing with all components. It is maintained by Framestore.
  - BEHAVIOURTEST: Added as a way to test the communication between UPF's behaviour planner and InfoCert's security module.
  - MIRROR: For the mirroring use case.
  - REGISTRATIONOFFICER: For the registration officer use case.
  - SPORTPRESENTER: For the sport presenter use case.
  - MULTIAGENT: For testing Inria's crowd simulations.
- Each case folder has a map, a DigitalHumanProcessing and a DigitalHuman (Agent) blueprint. The Processing blueprint contains the different components that connect and communicate with the Agent. It has variables that give access to the Agent, the main Spectator Camera and the Reference Human.
- If the use case is using any offline data assets (e.g. animations, audio, sequences) these will also be stored in the case folder.

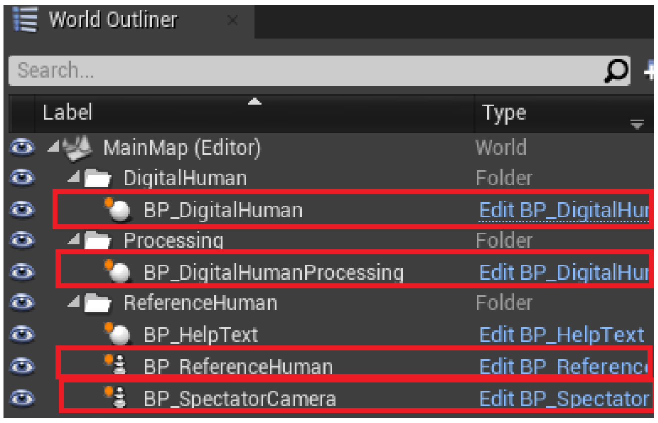


- Since version ***3.2.0*** the project contains multiple agent classes with different resolutions.
  - The **LowRes** agent was the original (placeholder) rig and is kept in case any partner wants to use it for any high-performance tests. It is currently not used by default in any use case map.
  - The **MidRes** agent is produced and maintained by Cubic Motion. It is currently used by default in all use cases. Animations can be triggered by playing level sequences (produced offline).

- The **HighRes** agent will be added in a later version and will be produced and maintained by Framestore. It will share the rig structure with the *MidRes* agent and should be able to use the same *Motion Generation* component for controlling the motions (streaming animations).

3.3 - Where are the elements in the Scene?

- The *World Outliner* shows the elements (actors) in the map.

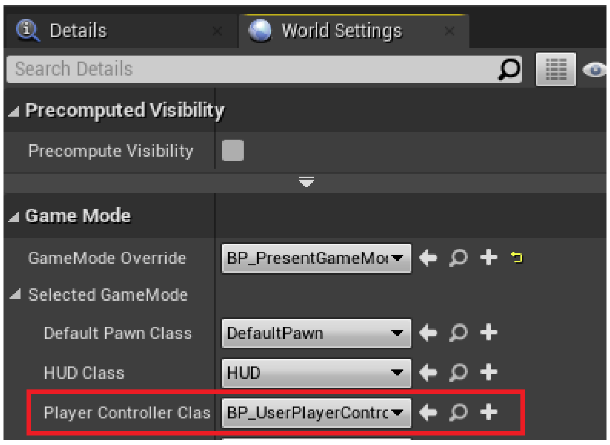


World Outliner

Search...

Label	Type
MainMap (Editor)	World
DigitalHuman	Folder
BP_DigitalHuman	Edit BP_DigitalHu
Processing	Folder
BP_DigitalHumanProcessing	Edit BP_DigitalHu
ReferenceHuman	Folder
BP_HelpText	Edit BP_HelpText
BP_ReferenceHuman	Edit BP_Referenc
BP_SpectatorCamera	Edit BP_Spectator

- The *InputController* is set up in the *Game Mode* in *World Settings*.



Details | World Settings

Search Details

Precomputed Visibility

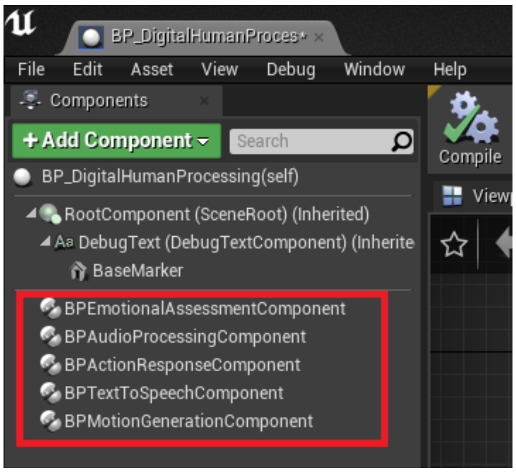
Precompute Visibility	☐

Game Mode

GameMode Override	BP_PresentGameMo
Selected GameMode	
Default Pawn Class	DefaultPawn
HUD Class	HUD
Player Controller Clas	BP_UserPlayerContr

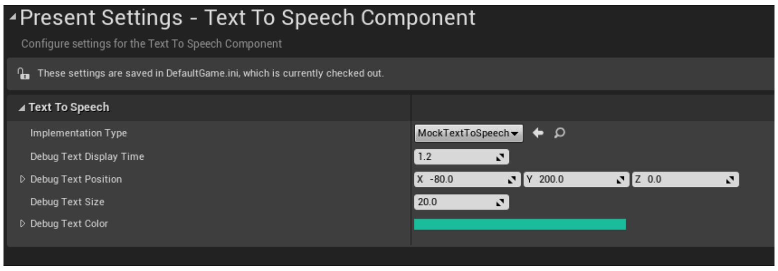
3.4 - How are components included and set up?

- The components are included in the *DigitalHumanProcessing* blueprints.



BP_DigitalHumanProcessing(self)
- RootComponent (SceneRoot) (Inherited)
  - DebugText (DebugTextComponent) (Inherite
    - BaseMarker
- BPEmotionalAssessmentComponent
- BPAudioProcessingComponent
- BPActionResponseComponent
- BPTextToSpeechComponent
- BPMotionGenerationComponent

- The global options/settings for each component are in the project settings.



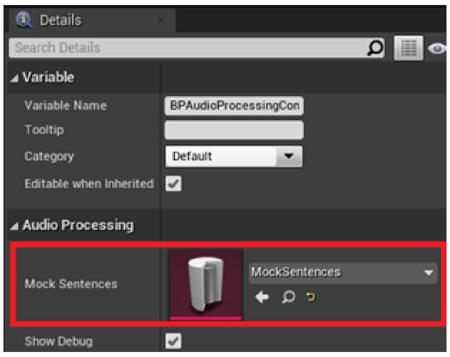
Present Settings - Text To Speech Component

Configure settings for the Text To Speech Component

These settings are saved in DefaultGame.ini, which is currently checked out.

Text To Speech	
Implementation Type	MockTextToSpeech
Debug Text Display Time	1.2
Debug Text Position	X -80.0 Y 200.0 Z 0.0
Debug Text Size	20.0
Debug Text Color	🟩

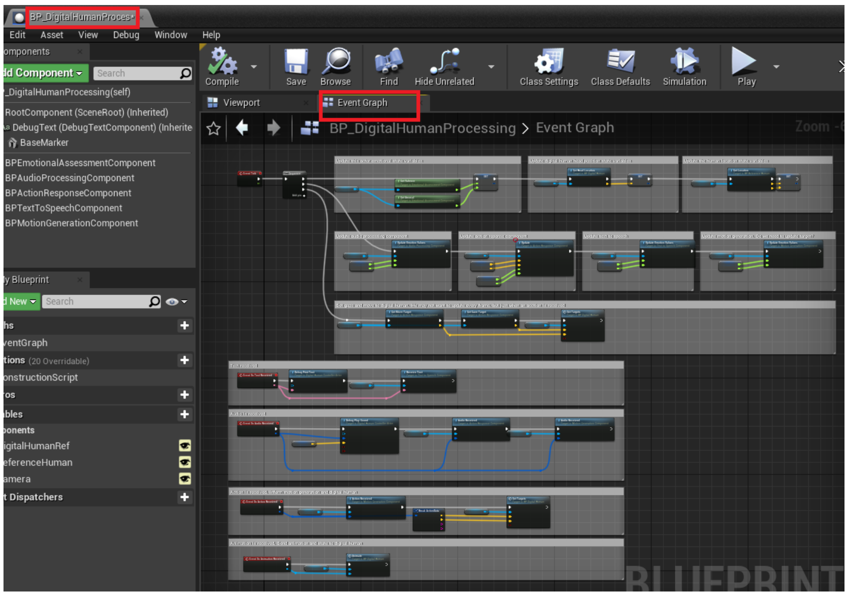
- The properties panel in the *DigitalHumanProcessing* blueprint exposes the component's properties. For example, a property can be the (mock) text sentences used by the audio processing or the number of agents used in a crowd simulation.



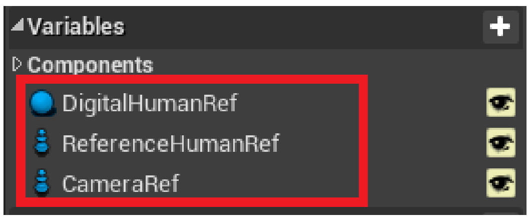
Variable	
Variable Name	BPAudioProcessingCon
Tooltip	
Category	Default
Editable when Inherited	✓
Audio Processing	
Mock Sentences	MockSentences
Show Debug	✓

3.5 - How do components work together?

- The logic that connects components is in a single event graph (in the *DigitalHumanProcessing* blueprints). It is structured this way to be easy to maintain and integrate. The graph should be changed according to the particular use case by using different services of the partners.

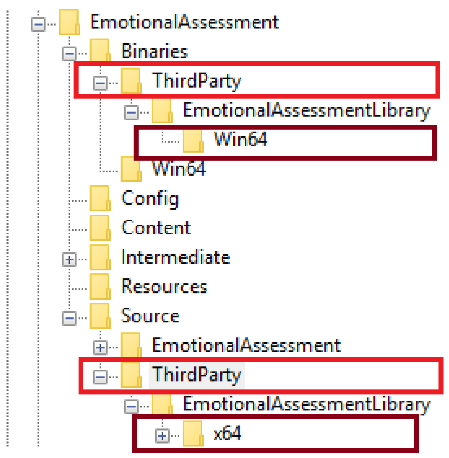

BP_DigitalHumanProces
Edit Asset View Debug Window Help
Compile
Save
Browse
Find
Hide Unrelated
Class Settings
Class Defaults
Simulation
Play
Viewport
Event Graph
BP_DigitalHumanProcessing > Event Graph
Add Component
_DigitalHumanProcessing(self)
RootComponent (SceneRoot) (Inherited)
DebugText (DebugTextComponent) (Inherite
BaseMarker
BPEmotionalAssessmentComponent
BPAudioProcessingComponent
BPActionResponseComponent
BPTextToSpeechComponent
BPMotionGenerationComponent
My Blueprint
EventGraph
ConstructionScript
DigitalHumanRef
ReferenceHuman
Camera
Event Dispatchers
BLUEPRINT

- The graph usually calls the components' functions during start (*BeginPlay*), update (*EventTick*) and when receiving events from the components like *OnUserSentenceReceived*, *OnAgentResponseReceived* etc. These events are fired by the components and they will be created according to partners requirements. Events should generally be created when a message has been received from a DLL or web server.
- Note that events triggered by received messages/responses (web services) or callbacks (DLLs) are to be preferred over periodically checking if a state has changed for performance reasons (with the exception of emotional values from the EmotionalAssessment component as it has a steady stream of updates).
- The graph also has variables that can be used to call functions in other actors (SpectatorCamera, ReferenceHuman and DigitalHuman). The functions in those actors will be added according to partners' requirements and can change depending on use case.



3.6 - How to use a DLL implementation?

- Plugins that implement services by using a DLL have the following structure:

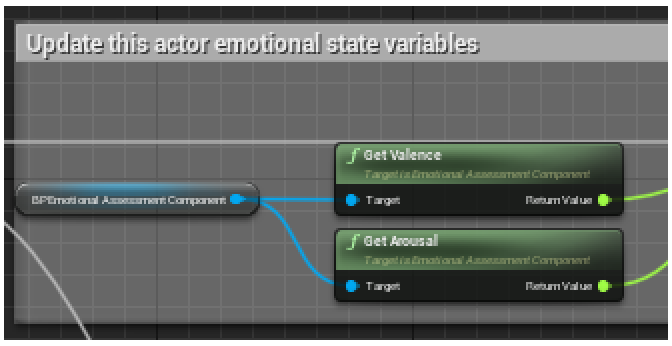


```
EmotionalAssessment
    Binaries
        ThirdParty
            EmotionalAssessmentLibrary
                Win64
        Win64
    Config
    Content
    Intermediate
    Resources
    Source
        EmotionalAssessment
        ThirdParty
            EmotionalAssessmentLibrary
                x64
```

- The ThirdParty directory under *Source* contains a Visual Studio solution that builds the DLL code. The result of the 'Release' build is in the x64 directory. The build should also automatically copy the *.dll* file into the *Binaries/ThirdParty/Win64* directory. This copy is the one loaded by the reference implementation. ***NOTE: The .dll file is only copied when building in 'Release' mode.***
- A DLL implements global functions of the form:

```
LIBRARY_IMPORT void OnModuleLoaded();
// Getter for the valence
LIBRARY_IMPORT float GetValenceValue();
```

- The reference implementation's plugin (e.g. *Source/EmotionalAssessment/Public/ ThirdPartyEmotionalAssessment.h*) wraps the DLL functions and makes them accessible to blueprints.

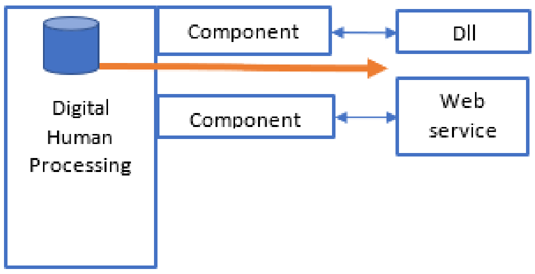

Update this actor emotional state variables
BPEmotional Assessment Component
Get Valence
Target is Emotional Assessment Component
Target
Return Value
Get Arousal
Target is Emotional Assessment Component
Target
Return Value

- ThirdParty service functionalities are implemented in the DLL. The DLL and plugins are built using Visual Studio 2019. Unreal must be closed before building the DLL as the copy won't work otherwise. Plugins can be built with

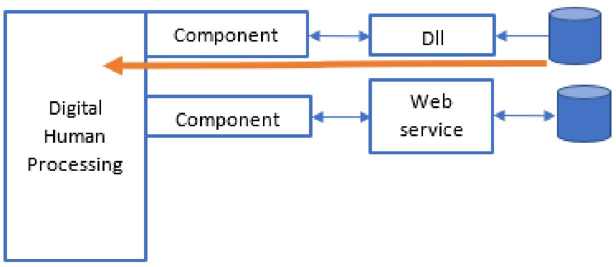
Unreal running but it is required to close and re-launch Unreal after the build, so that the plugin can be reloaded.

3.7 - How can data be used in the reference implementation?

- Data can be added in two ways:
  - As a data asset in Unreal. For example animation and audio files produced offline. This type of data will be added to the reference implementation according to the requirements of the partners. If the data needs to be read by the DLL or service code, then we will provide access functions from Unreal to the third party code (red arrow below).



  - Data can be part of the DLL or web API. If the data is local, then it can be added as part of the reference implementation. To access this type of data in Unreal, the component will be changed to have read access functions according to the requirement of the partner.



4. REFERENCE IMPLEMENTATION ACTORS

4.1 - Digital Human agent

The agent will exist in multiple resolutions maintained by different partners.

- LowRes agent
  - Placeholder agent used in the first versions of the reference implementation.

    - It will be kept in the project for a while if any partner wants to use it for high-performance tests. If no interest is shown after introducing the other agents it will be removed once the MidRes & HighRes agents have been stabilized.
- MidRes agent (since version ***3.2.0***)
    - Produced and maintained by Cubic Motion.
    - Use third party plugins ***Control Rig*** and ***Rig Logic***. These need to be enabled in the Unreal project for the face animations to work. This is done by default in the Reference Implementation project but if any partner wants to copy the assets to another project those plugins need to be enabled before migrating the assets!
    - Body animations, face control rig animations and audio files are synced up in *level sequences* which are produced offline. Sequences are associated with a specific map and are added as data properties to the DigitalHuman blueprint. Sequences are then identified by index when playing them.
- HighRes agent (not released yet)
    - Produced and maintained by Framestore.
    - Will share the rig structure with the MidRes agent.
    - Should be able to use the same Motion Generation component to generate animations and thus it should be possible to control it in the same way as the MidRes agent.

4.2 - Reference Human

The reference human actor is a basic pawn that is meant to only be used as a visual reference of where the user's body and head are positioned. An "actor" in Unreal is any object that can be placed in the scene and a "pawn" is an actor that can be controlled by the user.

4.3 - Digital Human Processing

The processing actor does not have a visual representation in itself. Instead it has a "debug sphere" for each actor component it contains which are visible during play. The spheres indicate when the different components are active and what outputs they produce. The spheres can be disabled in the project settings.

4.4 - Input Controls

By default it is possible to move both the Reference Human and Spectator Camera(s).

Reference Human controls:

- Move the ReferenceHuman horizontally with the arrow keys.
- Note that this should work regardless of which camera is used.
- When using the ReferenceHuman camera it can be panned and tilted by moving the mouse.

Spectator Camera controls:

- Move the spectator camera horizontally with WASD and vertically with QE.
- Pan/Tilt the spectator camera by moving the mouse.
- Note that it is only possible to move the camera that is currently in use. If the ReferenceHuman camera is in use then no input is propagated to spectator cameras.
- Apart from user input it is also possible to use blueprint functions (e.g. "*SetLocation*" and "*SetLookAtTarget*") to set camera properties. The current functionality is mostly a proof of concept. It can be improved or extended if needed for a use case.

Moving the Digital Human:

- *Basic movement:*
  - By default the DigitalHuman should follow the ReferenceHuman with an offset in most maps, including the reference map.
  - In a few case maps, e.g. the mirroring, it is not possible to move the DigitalHuman at all during runtime.
  - When using the multi agent case map the crowd simulation should control the position and orientation of all digital humans.
- *Playing offline animations:*
  - As of v3.2.0 all animations are produced offline and synced with audio in *level sequences*.
  - The sequences are associated with a specific map and are added as data properties in the digital human blueprints.
  - The sequences can be played by pressing the corresponding index number key or by selecting in the drop down widget.
  - When a sequence has finished playing an idling sequence will start automatically.
  - No blending is currently applied.
- *Streaming animations:*
  - In v3.2.1 the first version of animation streaming was added for the mirroring case map (logic also included in the reference case map).

- It uses python pickle files to map the emotional values to the face control rig keys. Therefore ***PythonScripting*** and ***SequencerScripting*** plugins need to be enabled.
- In the Digital Human blueprints there is a "Mirroring" function that solves for the keys and sets the corresponding face controls.
- Streaming animations/control rig keys will be the primary way to move the Digital Human in later versions of the project. However, the implementation will be embedded in the Motion Generation component instead of using python and blueprint functions.

5. PARTNER COMPONENTS OVERVIEW

This section gives a quick overview of the different actor components in the DigitalHumanProcessing. The DigitalHumanProcessing is the connecting link between all the components and is maintained by Framestore. It propagates messages and data between all the plugins, the reference human, the camera(s) and the digital human(s).

5.1 - Emotional Assessment (University of Augsburg)

- *Purpose:* Determine the current emotional state of the user
- *Input:* Audio & Video (SSI_Pipeline - run as local service)
- *Output:* Valence & Arousal (V&A, "Emotional Input") of both user and agent, both as continuous values (floats) and as discrete labels (strings).
  - Later: Expressiveness factor (how intense the emotions are) (float)
- *Connections:* Audio Processing, Action Response, Motion Generation, Text to Speech
- *Integration:* DLL, fetch values each frame through a UDP connection
  - Localhost, ports: 1111 (EmoSim)
  - Other ports in use: 2221, 2222, 8888, 9999

5.2 - Audio Processing (University of Augsburg)

- *Purpose:* Convert user's audio to text
- *Input:* Audio (SSI_Pipeline - run as local service)
- *Output:* User's speech in text, Text Time Code (TC)
- *Connections:* Action Response
- *Integration:* DLL, checks for new sentences periodically via UDP
  - Localhost, ports: 1112 (EventBoard/Speech)

5.3 - Action Response (UPF)

- *Purpose:* Find agent's response in text. Convert all available information to the agent's body & facial behaviour. Register User & Agent with security manager, periodically check security verification.
- *Input:* V&A, User's sentence (text), Text TC, Points of Interest (POIs)
- *Output:* Agent's response (text), Text TC, Targets (MoveTo, LookAt, PointTo), Face expression, Body Gesture, Security requests
- *Connections:* Text-to-Speech, Motion Generation, Security Manager
- *Integration:* Web sockets, trigger events on received messages
  - wss://webglstudio.org/port/9002/ws/

5.4 - Text to Speech (University of Augsburg)

- *Purpose:* Convert agent's text response & emotional state to an audio response.
- *Input:* Agent's response as text, V&A, Text TC
- *Output:* Agent's response as audio (SpeechData), Audio TC, Audio info (e.g. text, phonemes, visemes)
- *Connections:* Motion Generation
- *Integration:* Not implemented
  - Will probably be a DLL and check for new results periodically.

5.5 - Motion Generation (Cubic Motion)

- *Purpose:* Convert agent's behaviour, audio response, emotions and targets to an offline animation to play (with smooth blending transitions) or a continuous animation stream.
- *Input*: Agent's response (audio + text), behaviour data, emotional values & targets (MoveTo, LookAt, PointTo)
- *Output*: Animation stream, event triggers.
- *Connections:* Mid-res rig & High-res rig, BroadcastSportsAnalysis
- *Integration:* To be defined.
  - Probably a direct integration in Unreal with local assets.

5.6 - Security Manager (InfoCert)

- *Purpose:* Initial and continuous security authorisation for both Agent & User.
- *Input:* Initial and continuous verification requests.
- *Output:* Authentication results.
- *Connections:* Action Response
- *Integration:* REST (HTTPS) API

        - Making REST requests to remote server: https://present.etuitus.it/present/api/v1

5.7 - BroadcastSportsAnalysis (Brainstorm)

- *Purpose:* Integration to Brainstorm's InfinitySet software.
- *Input:* Event triggers from InfinitySet & Motion Generation.
- *Output:* Propagated events.
- *Connections:* Motion Generation, InfinitySet
- *Integration:* TCP communication between InfinitySet and the RefImpl
    - Localhost, ports: 5123
    - Web Remote Control (HTTP): 30010

5.8 - MultiAgent (Inria)

- *Purpose:* Integration to Inria's UMANS software.
- *Input:* Crowd simulation data from the UMANS API.
- *Output:* Position, orientation, trajectories and gaze targets for each of the simulated agents.
- *Connections:* Motion Generation, Agents (Digital Humans)
- *Integration:* Locally built DLL.

6. API DOCUMENTATION

All classes and methods in the reference implementation have been documented with Doxygen code block comments. For each released version the comments have been generated to HTML files that are easy to view in any web browser. The files have also been combined into a single Compiled HTML file that is included in the released zip file. Unfortunately the formatting in the compiled HTML file is not as smooth as in the original HTML files.

The compiled file can be searched or navigated through via class hierarchy or file list. However, as the resulting file links to all the classes, structs and namespaces in the project it can be difficult to find what you are looking for if you don't already know the answer. Therefore the most important classes have been added as separate "Modules" that can be found on the main navigation toolbar. These module classes are the actor components which in turn contain all the methods that are exposed in Unreal via the blueprint graph in the DigitalHumanProcessing.

7. CONCLUSIONS

This deliverable clearly demonstrates that key milestones have been achieved for the development of API’s and protocols within the reference implementation project and the project's function in facilitating the interface of multiple partner contributed components.

The project structure outlined in this document has already been used to successfully deliver two M18 POC deliverables, featuring work contributed by multiple partners. Thanks to the flexible structure, utlising native Unreal blueprint functionality to allow for the configuration of multiple component interfaces, the reference implementation project can be used to meet the needs of multiple varying use cases.

Work is ongoing to support additional components as they become available from project partners but based on the experience so far and the specifications presented there is no reason to believe the current structure will not be well suited to accommodate all the anticipated needs.