



D2.3 REVISED REPORT ON MODULAR ARCHITECTURE, PROTOCOLS AND APIs



Grant Agreement nr	856879
Project acronym	PRESENT
Project start date (duration)	September 1st 2019 (36 months)
Document due:	31/08/2020
Actual delivery date	31/08/2020
Leader	Framestore
Reply to	richard.olloson@framestore.com
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	856879
Project acronym	PRESENT
Project full title	Photoreal RE altime S entient ENT ity
Document name	Revised Report On Modular Architecture, Protocols And APIs
Security (distribution level)	Public
Date of delivery	31/08/2020
Deliverable name	D2.3 Revised Report On Modular Architecture, Protocols And APIs
Type	Report
Status & version	Submission Version
Number of pages	64
WP / Task responsible	Framestore
Other contributors	all partners
Author(s)	Theo Jones
EC Project Officer	Ms. Adelina Cornelia DINU - Adelina-Cornelia.DINU@ec.europa.eu
Abstract	This deliverable represents the initial architecture that will govern the overall design and deliverables associated with the PRESENT project. The goal is to provide a high level overview of the various components required to deliver the overall functionality.
Keywords	Software, systems architecture, use case reflections, risk analysis
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Revision History

Revision	Date	Description
v1.0	24/2/2020	First draft specification for distribution.
v1.1	27/2/2020	Revised and improved based on partner feedback.
v1.2	1/3/2020	Added Use Cases
v1.3	3/3/2020	Q2 Meeting Draft
v2.0	16/3/2020	Report Release
v2.1	18/3/2020	Final version: <i>Due to the COVID19 situation, it has been necessary to carry out the document review and adjustment process without the participation of the deliverable leader. Thanks to the very well structured information and organization of the document, and being it almost ready to deliver, this fact has not really affected the final result. For this final version, alternative contact is jmontesa@brainstorm3d.com</i>
v3.0	14/08/2020	Revised report based on current progress and partner feedback - for UPF review
v4.0	28/08/2020	Final submission version

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY	7
2. INTRODUCTION	8
2.1. Background	8
2.2. Objectives and Goals	8
2.3. Third Party Tools	8
3. METHODOLOGY	10
3.1. Software Loops and Continuity	10
3.2. Software Components and Applications	13
3.3. Input and Output Processing	14
3.4. Exploitation Environment	14
3.5. Reference Implementation	15
3.6. Agent Creation	16
3.6.1. Character Capture	16
3.6.2. Character Build	17
3.6.3. Performance Capture	18
4. DATA FLOW	20
5. TECHNICAL RISK ANALYSIS	21
5.1. RISK 1: Knowledge Base	21
5.1.1. Issue	21
5.1.2. Solution	21
5.1.3. Caveats	21
5.2. RISK 2: Multiple Agent Creation	21
5.2.1. Issue	21
5.2.2. Solution	22
5.2.3. Caveats	22
5.3. RISK 3: Audio Generation	22
5.3.1. Issue	22
5.3.2. Solution	22
5.3.3. Caveats	23
5.4. RISK 4: Unreal Engine Platform Constraint	23
5.4.1. Issue	23
5.4.2. Solution	23
5.5. RISK 5: Background Visualisation	23
5.5.1. Issue	23
5.5.2. Solution	23
6. HIGH LEVEL SYSTEMS ARCHITECTURE	24
6.1. Speech Input	25

6.2. Emotional Input	26
6.3. Body Input	27
6.4. Audio Trigger Input	28
6.5. Blueprint Input	29
6.6. Camera Input	30
6.7. Dialog	31
6.8. Security	32
6.9. Haptics Output	34
6.10. Motion Generation	35
6.11. Lightweight Asset	37
6.12. High Quality Asset	38
6.13. Agent State Output	39
6.14. Animation Data Format Specification	41
6.15. Core Application Logic	42
6.16. Components not covered in this report	44
6.16.1. Super Render Computer (WP6T2)	44
7. HIGH LEVEL AGENT CREATION ARCHITECTURE	45
8. CHANGE MANAGEMENT	46
8.1. Identification and Proposal	46
8.2. Approval	46
8.3. Re-Release	46
8.4. Component Update	46
9. REFERENCE IMPLEMENTATION	47
9.1 Components	47
9.2 Mock Variables	50
9. 3 Third Party Signatures	51
Emotional Assessment:	52
Audio Processing:	52
Text To Speech:	52
Action Response:	52
Motion Generation:	53
10. TESTS, VERSIONING AND DISTRIBUTION	53
10.1. Functional tests	53
10.2. Unit tests	53
10.3. Versioning	53
10.4. Component Distribution	54
11. USE CASE REFLECTIONS	55
11.1. Adam 0.1	55
11.2. Complex Social Situation	56
11.3. Greek Chorus	57

11.4. Sports Broadcast	58
11.5. Virtual Clerk	59
11.6. Virtual Production	60
12. CONCLUSIONS	61
APPENDIX A - PROJECT TERMINOLOGY	62

1. EXECUTIVE SUMMARY

This document represents an update to the initial architecture document delivered as D2.2. This update reflects the results of ongoing meetings and discussions between the partners that have refined the overall architecture and connections between certain components.

The project architecture reported here has been guided by

- i. Weekly meetings between Framestore, Cubic Motion, Augsburg, Inria and UPF.
- ii. Discussions in the quarterly project meetings attended virtually by all partners.
- iii. Meetings between all individual partners and Framestore (FS) as the partner responsible for the drafting of this document.

As defined at the grant proposal stage, a modular architecture has been pursued with 'Inputs' and 'Outputs' clearly defined and separated. Further, the design has been split into Components and Applications, establishing the separation between the functional components and the integration projects bringing these components together to create the use cases.

Complementary to this deliverable, work has been undertaken by Framestore to create a reference implementation in Unreal Engine, defining and prototyping the interfaces outlined in this document. This has been shared amongst the partners and provides a reference platform against which i) input components can be tested and validated and ii) provides a template for the integration projects. Although a number of key components are represented in the current, released version of the reference implementation there are further components to be added, in particular those relating to the integration of Brainstorm and Infocert's components.

As part of ongoing discussions, gaps have been identified in the overall system. These gaps are identified in this report and discussion is given to the current strategies and plans to mitigate / manage these. As virtual human interaction is a very active field of research, it is expected that over the 3 year cycle of this project, there will be many advances in the field. It is therefore expected that some of the gaps identified in the overall architecture may well be filled by open source third parties. The modular architecture presented here plays to these strengths as components can be interchanged either globally within the system or for specific use cases.

2. INTRODUCTION

2.1. Background

This deliverable is a fourth quarter deliverable against WP2 and updates the overall architecture of the project as well as adding further detail where planning and work has progressed.

This report details current progress on the reference implementation as well as updates to key aspects of the interconnection between components where discussions amongst the partners has led to more specific detail.

While each of the partners has a high degree of individual domain knowledge it should be emphasized that the challenge of this project does not lay in any one specific knowledge domain but rather in the integration and execution of the whole.

2.2. Objectives and Goals

- The overarching goal of PRESENT is to provide a Human Computer exchange framework capable of accepting and interpreting human natural language input and responding in an emotionally appropriate manner.
- Human speech and expression will be processed for both semantic and emotional content and these will inform and guide the PRESENT agent response.
- The PRESENT architecture will be constructed in a modular fashion such that individual feature enhancement and/or component substitution can occur with minimal to no disruption to the rest of the architecture
- To provide a clearly defined set of high level interfaces for exchanging data with downstream components.
- To provide a process for technical collaboration.
- To define a common technical platform on which the project is built.
- To establish a clear set of principles for how the system should be designed.

2.3. Third Party Tools

In order to understand the overall goals of the project, it is essential to understand the underlying framework and the design decisions taken toward achieving these goals at the very same time the technologies that underpin the framework are evolving at a very fast pace.

A key point to make is to look at where this project sits relative to other related efforts, most notably those of the current crop of digital assistants: Google Assistant, Amazon Alexa and Apple Siri. All of these technologies are designed in order to be embedded in as wide a variety of deployment models as possible - both in software and in hardware. Similarly, the financial incentives for capturing this portion of the digital assistant market are extremely lucrative and represent not only commercial possibility but are also the leading edge of AI, arguably they are the first steps toward Artificial General Intelligence.

As called out in the proposal, the PRESENT project is well placed to utilise these developments and tools and their use is integrated into the architecture outlined in this report.

3. METHODOLOGY

3.1. Software Loops and Continuity

The concept of a loop is a way to define a call and response operational state. An example from everyday life of a loop is a neighbour greeting you in the morning and your reply in kind. There is identification, engagement, acknowledgement and response.

The PRESENT project will deliver an environment wherein a human IRL (“In Real Life”) user engages the PRESENT agent (a CG creation) in a meaningful and conversational manner. In order to facilitate this dialogue, there are a number of “loops” that function within the larger architectural framework to accept input, process meaning and deliver a response

The challenge of defining an architecture such as this is the need for a dynamic response system where the base components respond in a continuous and consistent manner, independent of changing input parameters or component substitution. The overall goal of the project is natural human computer interaction that accepts and returns both audio and visual information. This includes all the human foibles such as awkward silences, poorly formed dialogue and incomplete sentences.

Continued next page...

- **Continuous Idle Loop** – While the application is waiting for user input, the CG agent must present itself in a manner that is natural and does not suggest to the user that it is anything but present in the room
- **Continuous Input** – This part of the pipeline represents the capture of user activity (audio as well as visual) and the continuous parsing and processing of that information. Processed user input information, such as simple values describing the user's emotional state, are continuously fed into the agent rendering and animation systems, allowing them to instantly respond to for example a change in mood in the user.

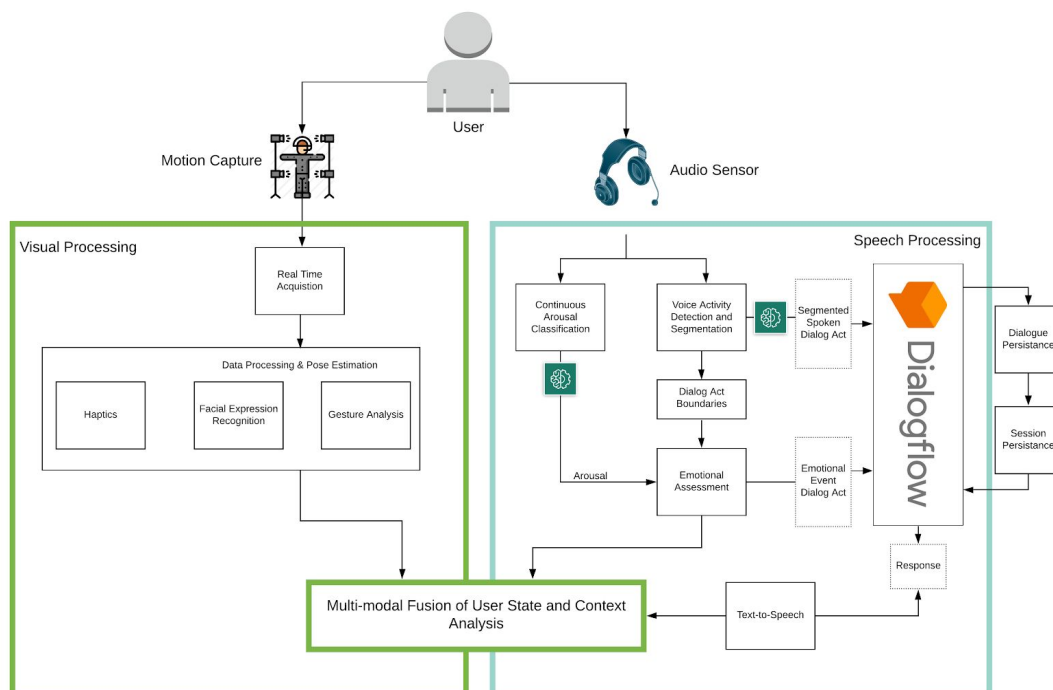


Figure 1: Input Loop

- **Continuous Authorisation** – The system is continuously ensuring that the current user is allowed to access the system.

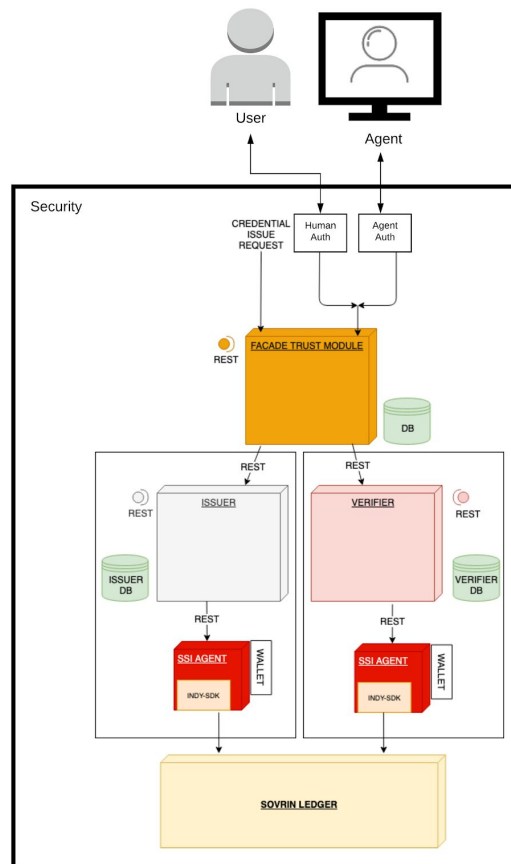


Figure 2: Authorisation Loop

- **Continuous Response** – This is a time sensitive path where interpreted emotional and semantic state, and evaluated dialogue and/or previous session persistence is converted into a response that can be packaged to:
 - Drive the visual CG rig (ie; lips are in sync, facial emotional cues are consistent and body language considered)
 - Deliver a response consistent with the user dialogue in a low-latency manner
 - Synthesize the verbal audio response

3.2. Software Components and Applications

The PRESENT architecture is divided into two fundamentally different concepts: **Software Components** and **Applications**, as illustrated in the figure below:

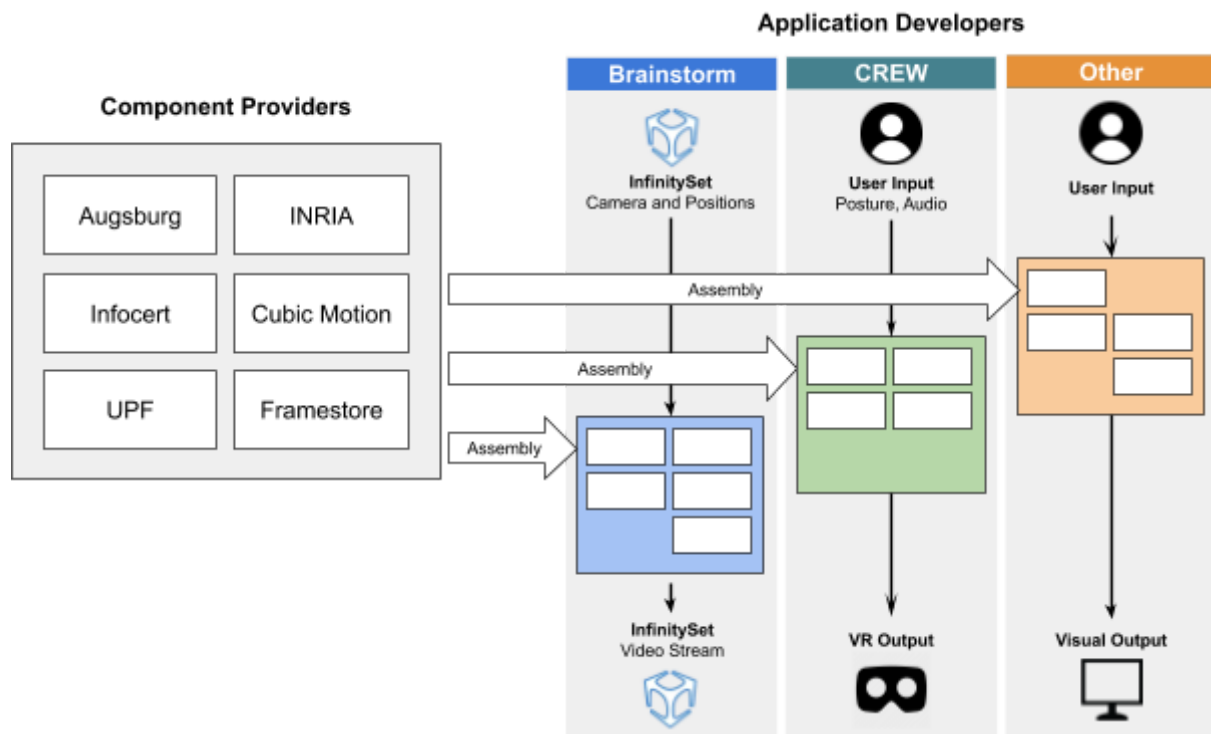


Figure 3: Separation of Components and Applications

- **Components** are distinct, orthogonal and independent units of functionality.
- **Applications** are projects which bind the components together to form a fully functioning executable.
- In the PRESENT consortium, most partners are **component providers**. Infocert, University of Augsburg, INRIA, Framestore and Cubic Motion are building software solutions that can be used in the context of PRESENT.
- On the PRESENT consortium, CREW and Brainstorm are **application developers** delivering executables to clients by integrating Unreal engine components into a fully functioning product.
- Each **component** has a well defined interface contract as well as a set of service level objectives (SLOs) to define its performance characteristics.
- The interface for each component is **versioned** with an integer number that keeps incrementing whenever a change to the interface is agreed upon.
- A special **Reference Implementation** Application is provided alongside the architecture to provide a fully working example, showing all the Components assembled into a fully working state. This is outlined in detail below.

With this architecture, components are highly independent, meaning that they can be easily utilised outside this particular context. For example, a visual input component that provides

the Unreal Engine environment with tracking data from a user may be utilised in other separate projects. With each component implementing a well defined interface, it is also easy to test and run a test application with a mix of components of different maturity.

3.3. Input and Output Processing

Game engines are part of a subclass of software called real time simulation software. Flight simulators are also based on the same philosophy (Ex: [CAE](#), the largest flight simulator company in the world, follows a similar architectural philosophy with their simulation software).

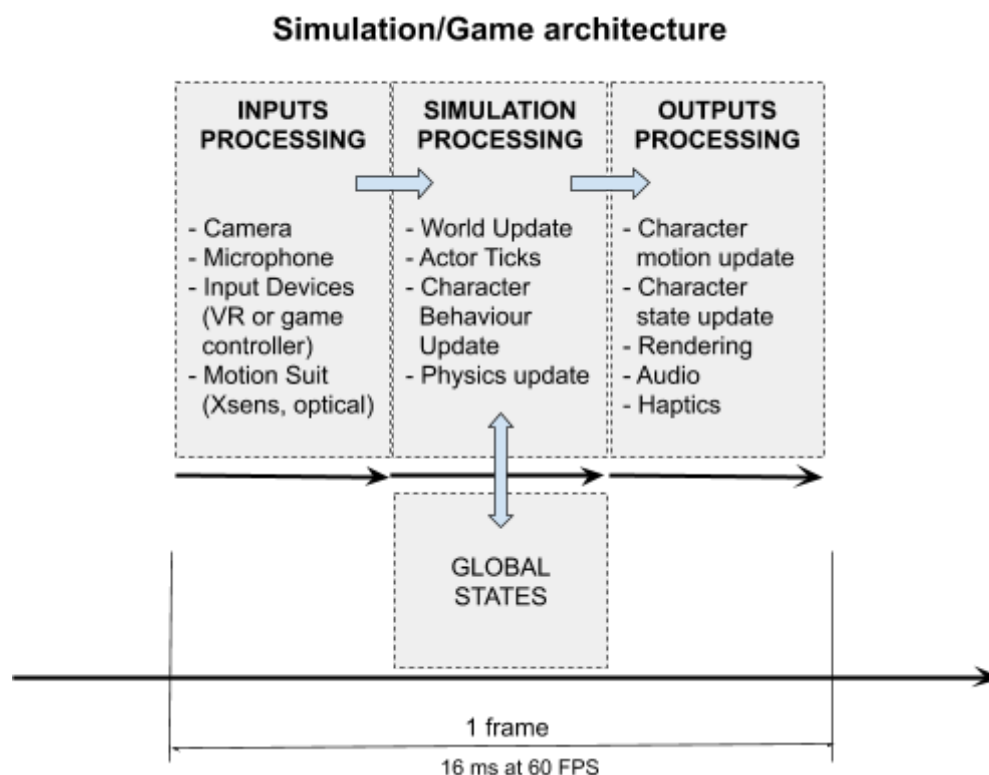


Figure 4: Simulation / Game Architectural Concepts

One of the core software architecture philosophies behind real time simulation software is the clear boundaries between components doing input processing and components doing output processing.

Note: While we could tolerate having monolithic software components able to handle both inputs and outputs, from an architecture standpoint and from a documentation standpoint, they will be represented by more than one interface.

3.4. Exploitation Environment

Technical collaboration on the PRESENT project requires a platform on which partners can safely and efficiently exchange technology and digital assets as well as share and collaborate on software components in an effective and low risk fashion. Furthermore, a well

defined realtime platform is a requirement for designing the high performance API interfaces required for the loops as well as the real time data exchange required by PRESENT.

This document therefore defines the principal operational environment for PRESENT as being **Unreal Engine**. Unreal Engine is the standard development platform in both the Broadcast and Visual Effects markets and is used by millions of developers and hundreds of millions of users. All interfaces and components outlined in this document are thus defined within Unreal Engine environment and are based on Unreal Engine SDK.

🚩 **Note:** While Unreal Engine is the central point where all components are being joined together into an application, the software architecture imposes no restrictions on how each component chooses to implement its core business logic. For example, an implementation may consist of a minimal unreal engine plugin that connects via socket IPC to a separate process which contains the business logic.

🚩 **Note:** As this is a three year project, we need to acknowledge that there may be advances in Unreal Engine which we want to exploit. The version of Unreal Engine being used as the reference implementation will therefore be reviewed at month 15 and month 24 and an assessment made as to whether to switch reference versions. The decision on whether to change will be taken in consultation with all partners and agreed in the Architectural Review Board. If a new version of Unreal Engine is decided upon, a new reference implementation will be created and circulated as appropriate in order that all parties have a common application environment to test within

3.5. Reference Implementation

When creating a set of independent software components, with the plan of later on assembling these into an application, it is crucial to connect them into a working environment as early as possible. Therefore, a reference implementation has been created in Unreal Engine v4.23, consisting of simple stub interfaces and component interfaces.

- For **application developers**, the reference implementation acts as a sample and a starting point for their work.
- For **component providers**, the reference implementation makes it easy to test that the integration of a component operates as expected in a full system.
- For all partners, it means work can happen in a safe and decoupled fashion.

Initial releases of this reference implementation have been made available to the partners. The latest release version is v2.1.1 and this contains representations of the following components:

- Emotional Assessment
- Audio Processing
- Text to Speech

- Action Response
- Motion Generation

The components associated with integrating Brainstorm's InifinitySet software as well as InfoCert's security API's are not yet present. Support for these components will be added in future releases.

3.6. Agent Creation

3.6.1. Character Capture

Creating the highest quality agent requires the highest quality photographic reference data of a source actor. The benchmark capture system for the highest quality facial geometry is The 'Medusa Facial Capture System' developed by Disney Research. Further, the highest quality facial texture data is provided by the 'Dorothy' capture system developed by Clear Angle. Pursuant to gaining the best source data for the project, per the amendment 'Proposal for reallocation of Ikinema work' Framestore have i) cast a source actor and ii) performed full capture sessions with both Medusa and Dorothy iii) received, QC'd and ingested all captured material. The capture systems above have the following key benefits over other solutions.

Compared to other systems the Medusa system captures not only the endstate of the fully triggered shape but generates a temporally consistent capture of an expression from the neutral face to the final end result. Other systems are not able to give us more than just a singular snapshot of the individual expression. In order to achieve high quality believable deformations it is however vital to know about the movement of a given shape and therefore muscle in all its states besides just the extremes. Humans when talking fire a lot of facial muscles simultaneously but rarely is a muscle triggered to more than 50% or more when talking or subtly emoting. The Medusa capture system is currently the only way to capture these complex low range motions that we otherwise have to invent ourselves or just rely on the defomer's linear interpolation which will likely yield unsatisfying results. Also as a positive side effect the resulting data saves time for the modelling artist creating the facial setup and ensures detail is maintained in a realistic fashion between the varying states of a given shape. It achieves this by providing a constant topology on the facial meshes for each face shape captured and therefore increasing the quality of the final output we are able to achieve.

The Dorothy capture system is a purpose built rig, specifically for facial capture, which combines high resolution cameras with a 360 degree light rig designed specifically to produce the flat texture data that is required for CG asset creation. This system also provides excellent reference for the lookdev artist beyond the texture acquisition step. It achieves this by firing many lights sequentially in rapid succession in a controlled lighting environment which allows artists to successfully adjust the shader models mimicking the complex behaviours of the human skin in various lighting conditions in a realistic way.

Framestore put out two casting calls to secure the services of a source actor and reviewed over 80 audition tapes. 7 actors were called in to audition in front of Framestore and Cubic Motion representatives. This process resulted in 'Gareth Leighton' being cast as the source actor and presented to the consortium partners in the Q3 project meeting. Gareth's casting was based on 3 parameters i) acting ability, ii) ability to accurately pull FACs shapes (specific facial expressions) to drive the facial rig being generated, iii) neutral facial anatomy and features. Gareth's performance was judged over these three parameters respectively as constituting the best combination of performance to i) provide the best possible acting source data for Cubic Motion's work packages to derive and learn performance data, ii) form the basis of a pose based facial rig and iii) provide as neutral a template for potential future transfer to other agents with different facial anatomy.

Due to the extremely labour intensive process of creating a computer generated agent that advances the state-of-the-art in real-time rendered humans, only a single hi-res agent is within the scope of the project. Therefore only one actor was cast as the source for this build. Evidently one actor cannot encompass the full range of human gender/race/age/etc. In order to address this concern a rig control template will be provided and documented that will allow any future agent adhering to this convention to be plugged into the system. Although outside of the scope of this project, it is hoped that this will allow future agents of different genders, races and ages to be easily incorporated into the system.

In addition to the two capture sessions outlined above, Cubic Motion will carry out a series of motion capture / head mounted camera sessions with Gareth to capture high volumes of body / facial footage of him interacting with users in the use cases defined. Cubic Motion will tag and analyse Gareth's performance and his reactions to different users in different emotional states to build a library of performance from which to build algorithms to derive wholly new performances depending on the emotional state and body language of the person interacting with the agent.

The ability of Cubic Motion to carry out these motion capture sessions has been severely impacted by the COVID-19 pandemic and the resulting restrictions on travel and multiple person gatherings. In order to maintain progress Cubic Motion are utilising generic motion capture for the development phase of their component. This situation will be monitored and the Gareth specific sessions scheduled as soon as restrictions allow.

3.6.2. Character Build

There are different applications for the agent within the project scope which require different agent visual and computational complexity. For some use cases / partners / research streams, the goal is to run the avatar at realtime in a lower specification computational resource environment (e.g. web based), whereas in other cases the goal is ultimate visual fidelity.

We have therefore split the agent development into two streams, the 'lightweight' agent and the 'high fidelity' agent. Taking the Medusa and Texture scans of the actor from who's image the agent is to be derived, 2 versions of the agent will be created. Framestore are building

the 'high fidelity' agent using the highest quality character rigging and shading models and pushing what is possible with the highest end hardware. Using the same source data (photogrammetry scans of Gareth Leighton), Cubic Motion are building the 'lightweight' agent using their established CG character build process.

Given the 'lightweight' approach uses established build processes, it is expected that this will reduce the time to having a first pass agent available in the system. This allows Cubic Motion to do their initial animation training on the 'lightweight' agent and rig whilst the 'high fidelity' agent and rig are being built. By working closely on the rig controls, Framestore and Cubic Motion have aligned the rig control design such that retargeting the animation to the higher resolution rig at a later date is as simple as possible.

By aligning on a common rig interface and control layout, the two agent rigs have been designed to be compatible. This means a common interface between the rigs and a common animation solve in the main loop with the same animation able to drive either rig. This approach is intended to be complementary to the overall goals of the project, specifically a single animation driving both rigs and delivering a varying level of visual fidelity depending on use case.

In order to focus on getting the highest visual fidelity agent possible, Framestore will create a single agent of the highest visual fidelity (the 'high fidelity' agent). By having the 'lightweight' agent build approach, we potentially unlock a workflow for the creation of other agents at lower effort / cost. This approach is also intended to unlock the issue of potentially creating multiple agents that can better reflect diversity of gender, race and age in the future. By designing and supplying a rig interface specification that clearly lays out the structure required for system compatibility other partners will in future be able to create agent rigs that can be driven by the system using the same user interface or via a retargeting module.

One of the challenges of introducing other agents into the system will be the visual result of driving animation 'learned' from our performer's animation (Gareth Leighton) to drive the animation of a second person. A person's anatomy and expression are intrinsically linked. In casting Gareth, we have chosen an actor with a neutral facial anatomy in an effort to mitigate these influences.

3.6.3. Performance Capture

In order to model the behaviour of the actor we require large volumes of performance data. This consists of the face and body movement, as well as the actor's voice. This needs to be done concurrently, since there are correlations between these aspects of behaviour which should be reflected in the performance of the virtual agent. These capture sessions have been delayed due to the COVID-19 travel restrictions. Therefore work is currently progressing using generic motion capture that will then be swapped out for capture done with Gareth Leighton when restrictions allow.

For facial performance capture, the gold standard is multi-view 4D reconstruction of geometry using a system such as Medusa. This is impractical for our purposes due to cost

and the inability to capture face and body at the same time. We propose to use a head-mounted camera system to obtain the best quality we can within these constraints. We will use two synchronised 2k computer vision cameras operating at 60fps mounted on a helmet to gather video data of the actor's performance. The position of these cameras will be fixed relative to each other, allowing stereo calibration and reconstruction of the actor's facial geometry. This will not be of the same quality as the Medusa data but will be state of the art for head-mounted capture. We will undertake experiments to refine the hardware to improve capture and reconstruction quality.

Body movements will be captured using industry standard optical motion capture, solving to a body skeleton matching the actor's dimensions. Since the body movements of the actor are fairly limited, this will be sufficient for our purposes. Hand and finger movements are more difficult to capture and important for the purposes of this project due to the role of hand gestures in the communication of the agent. There are two options for finger capture: optical motion capture and gloves with inertial sensors. We will evaluate these options and choose the method delivering highest fidelity.

Audio will be captured in a sound-proofed studio environment. This will be synchronised with the face and body data using industry standard timecode.

The range and type of data we choose to capture is an open-ended question. Some standard behaviour will be required to cover basic communication, idling and transitions, but we will also need specific performance data related to the use cases in the project. We will need to capture all of this data over a range of different emotional and behavioural variations to provide enough data to build a generative model of performance. We propose to stagger the motion capture sessions over a number of months to allow for experimental progress and to account for the development of multiple use cases.

This approach is intended to provide a means of capturing a large, high quality data set capable of delivering photo-real animation and modelling the range of behaviour needed for a believable virtual agent.

4. DATA FLOW

Figure 5 outlines the data flow through the system (as presented by Framestore at the Q1 summit) and highlights partner responsibilities within the project. Further, it calls out the gaps in the partner work packages and indicates the proposed solutions to these gaps where applicable. This breakdown has formed the starting point for the iteration process which has led to the high level architecture specification outlined in this document.

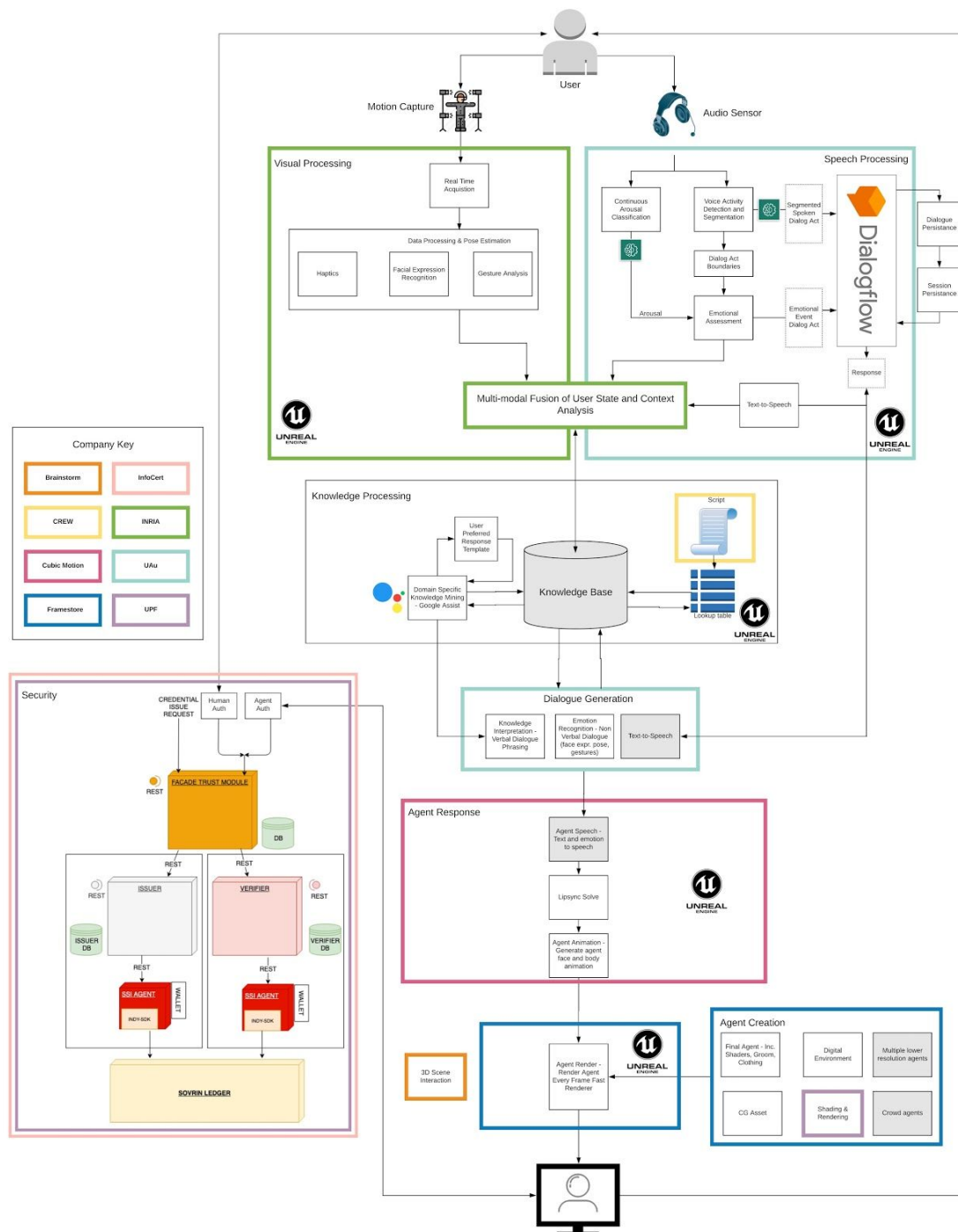


Figure 5: Data Flow Overview (v1.1)

5. TECHNICAL RISK ANALYSIS

Over the course of the architectural design process, risks have been identified in the system which fall between the work packages.

5.1. RISK 1: Knowledge Base

5.1.1. Issue

The level of specialised knowledge required to address issues surfacing around the Knowledge Base strike to the core of the challenges facing AI and more appropriate to this project, AGI (Artificial General Intelligence). Directly addressing these challenges are out of scope for the partnership.

5.1.2. Solution

It is anticipated that developments in the larger community around digital assistants will lead to progress and enable an increasing level of proficiency around the integration of "off the shelf" solutions. By keeping the architecture and API's modular it is anticipated that current offerings (DialogFlow) will naturally lead to more advanced integrations (Google Assistant)

The knowledge base will therefore be implemented in the core application logic, specific to each use case. Some applications make use of a simple lookup table, others may use Google Assist / Amazon Alexa.

UPF have started research that may provide a common framework for underpinning the application logic in some or all of the use cases.

5.1.3. Caveats

As a result of developments in this area, it is possible that the dialogue component may well evolve over the course of the project.

5.2. RISK 2: Multiple Agent Creation

5.2.1. Issue

The creation of high resolution, photoreal CG characters is a highly time and labour intensive process. The capture costs alone are a significant proportion of Framestore's project costs.

In order to push the visual limits of real time character development, Framestore's skills and focus need to be focused on creating a single character / agent of the highest quality (derived from the scan of our source actor). However, there are use cases in PRESENT that require multiple agents, not necessarily at the highest visual quality.

Further, Framestore's rigging technology is both proprietary and requires very highly skilled individuals to use.

Likewise, per the 'Performance Capture' section of this report (METHODOLOGY), the animation training dataset for the PRESENT project will be trained from the performance of our main visual agent (the acting of our source actor). Again, training is a highly time and labour intensive process and Cubic Motion will be focusing on a single solve to a single performer.

As the rig is the interface between Cubic Motion and Framestore, this presents challenges. i) The animation training in Cubic Motion's work packages needs to be targeted to a single rig interface. ii) other partners need to be able to create simpler agents with simple rigging tools that have a common interface.

5.2.2. Solution

Cubic Motion are a specialist provider of rigs for the games industry and have licensable plugins for the creation of rigs suitable for high end game characters. Cubic Motion and Framestore are aligning rig controls to a common interface such that the animation system can be retargeted between the Framestore rig and the Cubic Motion rig. This will allow alternative agents to be created by partners using the commercial Cubic Motion plugin and be driven from the same animation system. Cubic Motion will provide a license for the plugin to each partner for the duration of the project.

5.2.3. Caveats

Regardless of the rigging technology used, human performance and anatomy are fundamentally directly linked. Therefore mapping animation performance derived from one human to another human (CG character) anatomy will yield results that will break this link. In casting our source actor, we have taken care to cast an individual with 'generic' features with the aim that the dataset is as transferable as possible. However, it will always be the case that the animation system will work best on the actor from which the performance is derived.

5.3. RISK 3: Audio Generation

5.3.1. Issue

The generation of agent speech poses a number of unique challenges to the consortium. While significant effort has been made to capture and process human input, the resulting generation of the agent speech requires a skill set outside that contained in the group. There are design decisions: go with a synthesized voice and determine how to match phonemes to appropriate emotional content and delivery, use ML techniques on a captured audio dataset matching the human used for the base scan and use this to drive the facial rig.

5.3.2. Solution

There are two possible solutions to this issue:

- The first solution is to use Google Dialogue flow to generate the speech (audio). This solution is presented in the architecture presented here.
- The second solution, at least for the high resolution agent, is to procedurally process audio captured during the Performance Capture of the source actor. This is currently presented as a stretch goal, but may yield results in future.

5.3.3. Caveats

The use of Google Dialogue Flow will result in a computer generated voice. This will reduce the sense of 'Presence'.

5.4. RISK 4: Unreal Engine Platform Constraint

5.4.1. Issue

With the architecture outlined in this document, Unreal Engine is singled out as the single application environment where PRESENT use cases can be exploited. The modular nature of the architecture allows components to be used in isolation or together to fulfil each partner and use case needs. Deciding on Unreal as the integration platform is a key architectural decision necessary in order to mitigate risk and allow partners to effectively exchange code as well as digital assets.

5.4.2. Solution

Partners are aligned on Unreal as the application environment. The definition of the interfaces specified in this report, coupled to the reference implementation, allows partners to develop components in whatever development environment best fits the functionality of that component.

5.5. RISK 5: Background Visualisation

5.5.1. Issue

The project specification outlines a component that handles background visualisation. (WP3T20. Because this is handled in month 13-24 of PRESENT, this specification is not yet defined.

5.5.2. Solution

This component is to be discussed in the next quarterly meeting in order to map out a solution satisfactory to all partners.

6. HIGH LEVEL SYSTEMS ARCHITECTURE

The diagram below outlines the different software components in the system. It also outlines a high level design for the application logic - the part of the system that each application implementation would need to construct in order to bind all the components together into a fully working system.

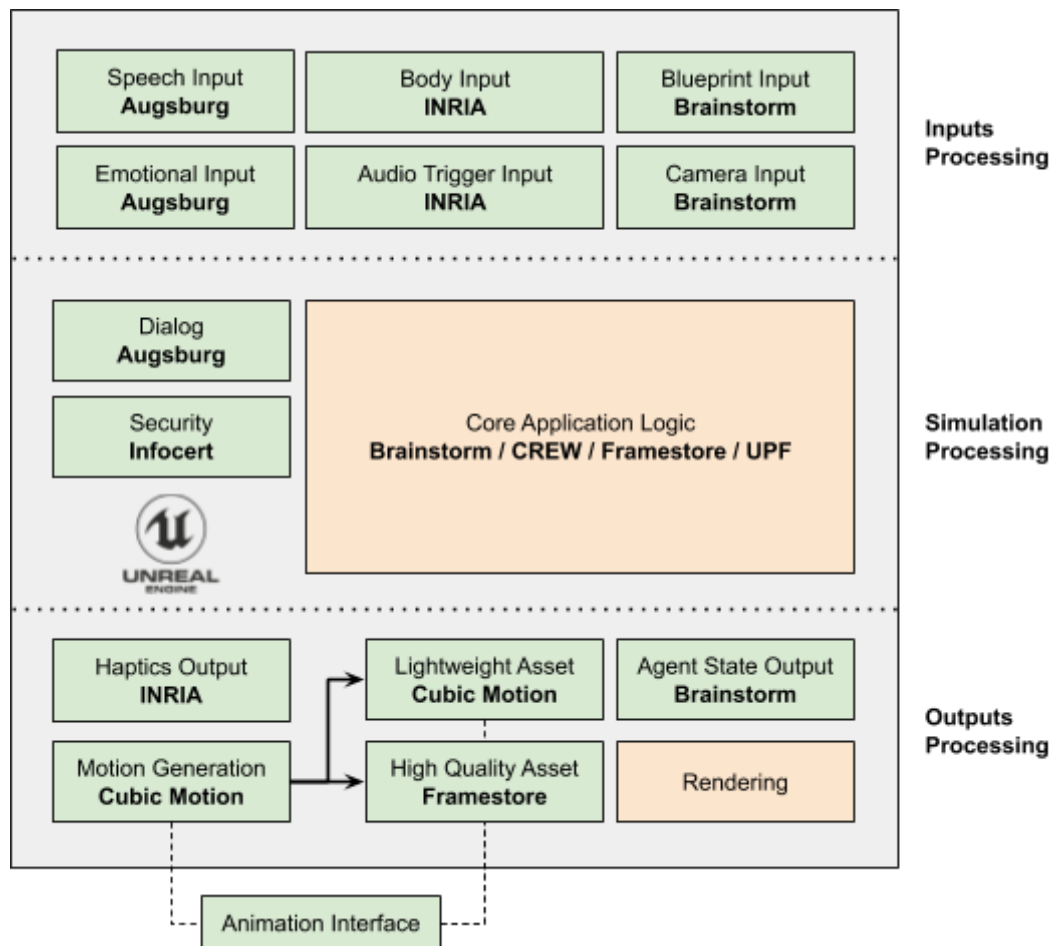


Figure 6: High Level Architecture

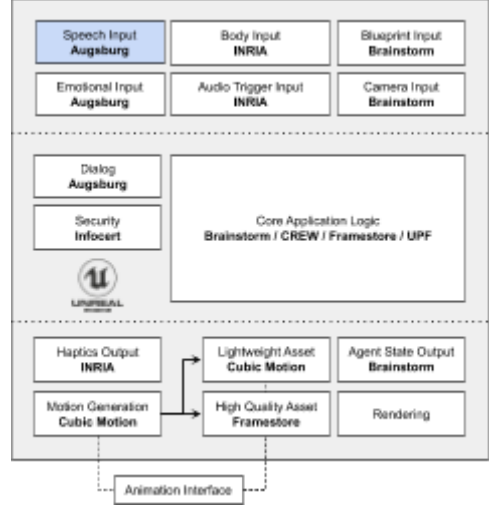
- Each component is implemented as part of an unreal plugin which can be accessed either via C++ or via a blueprint.
- Each of the components outlined above has a well defined interface that specifies its data exchange.



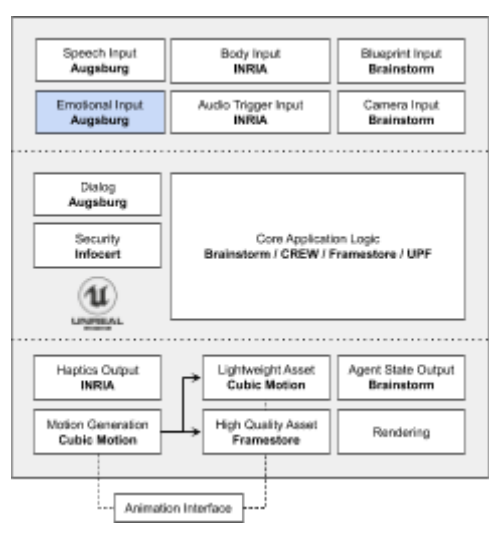
Note: The breakdown in Figure 6 outlines the component *interfaces*. While each interface needs to be part of an unreal plugin, there is nothing stopping a plugin from implementing multiple interfaces if this makes the implementation easier.

In the sections below follow principal interface details for all components defined above.

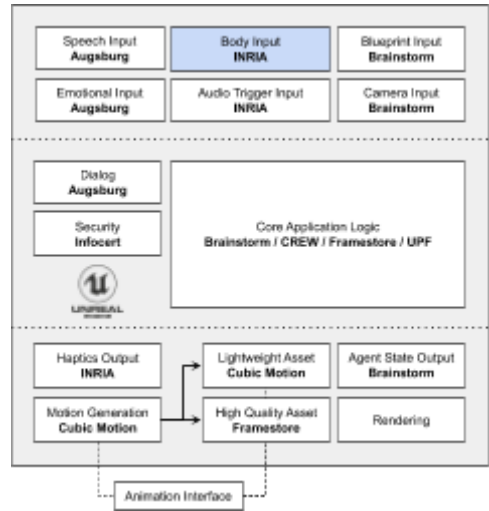
6.1. Speech Input

Description	Processes user speech input and returns this as text.	 <p>The diagram illustrates the system architecture. It is divided into several layers. The top layer contains input modules: Speech Input (Augsburg), Body Input (INRIA), Blueprint Input (Brainstorm), Emotional Input (Augsburg), Audio Trigger Input (INRIA), and Camera Input (Brainstorm). Below these is a middle layer with Dialog (Augsburg), Security (Infocert), and a central Core Application Logic block (Brainstorm / CREW / Framestore / UPF). The bottom layer includes Haptics Output (INRIA), Motion Generation (Cubic Motion), Lightweight Asset (Cubic Motion), High Quality Asset (Framestore), Agent State Output (Brainstorm), and Rendering. An Animation Interface is shown at the very bottom, connected to the Motion Generation and High Quality Asset blocks.</p>	
Revision	2		
Owner	University of Augsburg		
Implementation Details	This will utilize Google Dialog Flow for its implementation. It will process an audio stream from a microphone, determine when a sentence or phrase ends, and at that point connect to the Dialog Flow web service to request a translation.		
Performance/ SLA	The system will wait until the end of a sentence (or for a maximum of 10 seconds) and will then respond within 2 seconds (including going through Google Dialog Flow). For short sentences, a roundtrip will take ~2.5s. Worst case scenario is that an error is returned after 10+2=12 seconds.		
Notes			
Utilised by use cases	Greek Chorus, Adam, CSS (CREW), Virtual Clerk (UPF), Sports Broadcast (Brainstorm), Virtual Production (FS)		
Configuration	Hardware (microphone) configuration parameters.		
Interface Inputs	The ability to <ul style="list-style-type: none">- Register an event that fires when a sentence begins.- Register an event that fires when a sentence ends.- Register an event that fires when processing is complete.		
Interface Returns	<ul style="list-style-type: none">- Spoken dialog, as a string.- Audio data in raw form.		

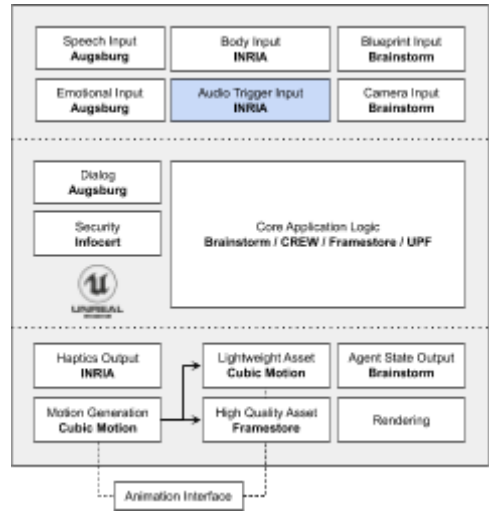
6.2. Emotional Input

Description	Processes user input and returns an approximation of the user's emotional state.	 <p>The diagram illustrates the system architecture. It is divided into several functional blocks. At the top, there are input modules: 'Speech Input Augsburg', 'Body Input INRIA', 'Blueprint Input Brainstorm', 'Emotional Input Augsburg', 'Audio Trigger Input INRIA', and 'Camera Input Brainstorm'. Below these are 'Dialog Augsburg' and 'Security Infocent'. The central part is a large box labeled 'Core Application Logic Brainstorm / CREW / Framestore / UPF'. Below the core logic, there are output and processing modules: 'Haptics Output INRIA', 'Motion Generation Cubic Motion', 'Lightweight Asset Cubic Motion', 'High Quality Asset Framestore', 'Agent State Output Brainstorm', and 'Rendering'. At the bottom, an 'Animation Interface' connects the motion generation and asset processing blocks.</p>
Revision	1	
Owner	University of Augsburg	
Implementation Details	The system consists of an unreal implementation and an external executable that communicates using an IPC mechanism (sockets). The external executable processes audio and video at a constant rate of 30FPS and can be queried about its last known state at any given time.	
Performance/ SLA	<ul style="list-style-type: none">- Return a response within 10ms- Ensure that response data is never older than 150ms (~5 frames @ 30FPS)	
Notes		
Utilised by use cases	CSS, Adam (CREW), Sports Broadcast (Brainstorm), Virtual Clerk (UPF)	
Configuration	Hardware (microphone and camera) configuration parameters.	
Interface Inputs	<ul style="list-style-type: none">- A method to query if a user has been detected.- A method to query emotional state.	
Interface Returns	<ul style="list-style-type: none">- Returns a floating point tuple (arousal, valence) describing a coordinate in an emotional state space for the currently detected user.	

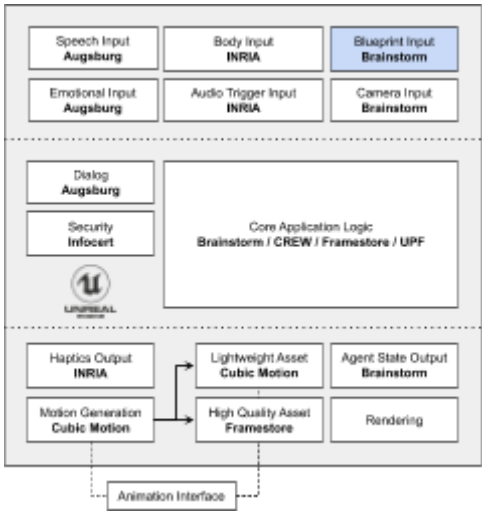
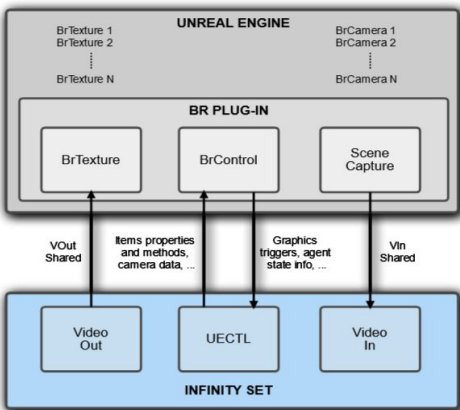
6.3. Body Input

Description	Processes user input in order to output a model for the user body or hand motion.	 <p>The diagram illustrates the system architecture. At the top, there are three input modules: 'Speech Input Augsburg', 'Body Input INRIA' (highlighted in blue), and 'Blueprint Input Brainstorm'. Below these are 'Emotional Input Augsburg', 'Audio Trigger Input INRIA', and 'Camera Input Brainstorm'. A dashed line separates these from the core logic. The core logic section includes 'Dialog Augsburg', 'Security Infocent', and a large box for 'Core Application Logic: Brainstorm / CREW / Framestore / UPF'. Below the core logic is the 'Animation Interface' section, which contains 'Haptics Output INRIA', 'Motion Generation Cubic Motion', 'Lightweight Asset Cubic Motion', 'High Quality Asset Framestore', 'Agent State Output Brainstorm', and 'Rendering'. Arrows indicate the flow of data from inputs through the core logic to the various outputs.</p>
Revision	1	
Owner	INRIA	
Implementation Details	Processes inputs from a Xsens motion suit at interactive rate. Outputs data to describe the user's body language.	
Performance/ SLA	Requires a fast response (50ms average response time, 1.5 frame buffering at 30fps).	
Notes	An almost direct connection to the xsens motion suite. There is an opportunity to potentially align the interface of this component so that it could be implemented in full by a third party plugin.	
Utilised by use cases	Greek Chorus, CSS, Adam (CREW)	
Configuration	Hardware configuration parameters. <ul style="list-style-type: none">- Ability to need hand tracking/full body tracking	
Interface Inputs	The ability to query for the current body motion state of the user: <ul style="list-style-type: none">- The global coordinates of limbs- Centre of mass- Position of the eyes / head- Position of the feet- Velocities and Acceleration	
Interface Returns	Data structure of a rigged hierarchy of matrices denoting the orientation and velocity of the limbs	

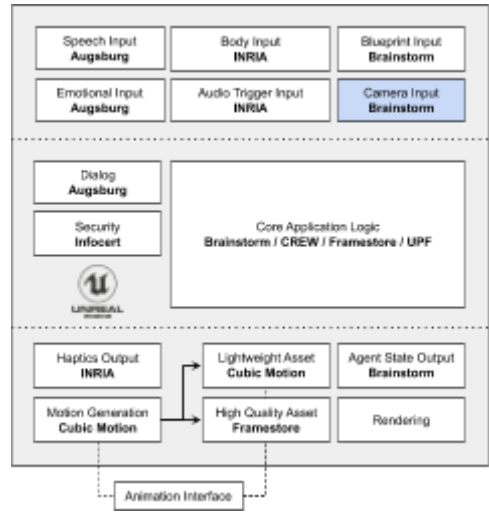
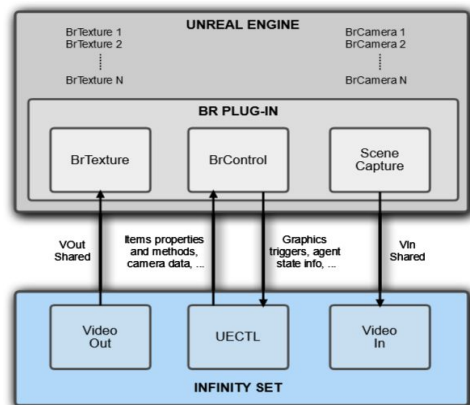
6.4. Audio Trigger Input

Description	Processes audio and sends an event once it reaches a given level.	 <p>The diagram illustrates the system architecture. At the top, there are three input modules: 'Speech Input Augsburg', 'Body Input INRIA', and 'Blueprint Input Brainstorm'. Below these are 'Emotional Input Augsburg', 'Audio Trigger Input INRIA' (highlighted in blue), and 'Camera Input Brainstorm'. These inputs feed into a central 'Core Application Logic' block, which also receives 'Dialog Augsburg' and 'Security Infocent' inputs. The core logic is associated with the 'UPF' logo. Below the core logic, there are three output modules: 'Haptics Output INRIA', 'Lightweight Asset Cubic Motion', and 'Agent State Output Brainstorm'. The 'Haptics Output' and 'Lightweight Asset' modules feed into 'Motion Generation Cubic Motion', which then feeds into 'High Quality Asset Framestore'. Finally, the 'High Quality Asset Framestore' feeds into 'Rendering'. An 'Animation Interface' block is shown at the bottom, connected to the 'Motion Generation' and 'High Quality Asset' modules.</p>
Revision	1	
Owner	INRIA	
Implementation Details	This component monitors external audio hardware and sends an event once the audio signal reaches above a given threshold.	
Performance/ SLA	Event emission within 20ms after audio detected.	
Notes	Used to for example trigger logic after a user claps their hands.	
Utilised by use cases	n/a	
Configuration	Hardware (microphone) configuration parameters. Threshold level to trigger at.	
Interface Inputs	The ability to register an event that fires when the threshold is reached.	
Interface Returns	Event contains details of the detected audio amplitude (level) detected.	

6.5. Blueprint Input

Description	Processes external data and turns that into a stream of commands that control blueprint state.	
Revision	2	
Owner	Brainstorm	
Implementation Details	This component will enable communication with existing Brainstorm's tools, such as InfinitySet. It will allow those tools to command some aspects of Unreal Engine.	
Performance/SLA	Once the request has arrived in the engine, processing should be instant (<5ms).	
Notes	<p>This component does not expose an interface to the application layer:</p> <ul style="list-style-type: none">- Ability for a client to connect.- Ability for a client to introspect a subset of application blueprints at runtime.- Ability for client to read, write blueprint values	
Utilised by use cases	Sports Broadcast (Brainstorm)	
Configuration	<ul style="list-style-type: none">- Connection parameters- Which blueprints and parameters are available for introspection.	
Interface Inputs	Ability to start and stop the service	
Interface Returns	A series of asynchronous events coming from clients. Events notifying the application about connection and disconnection.	

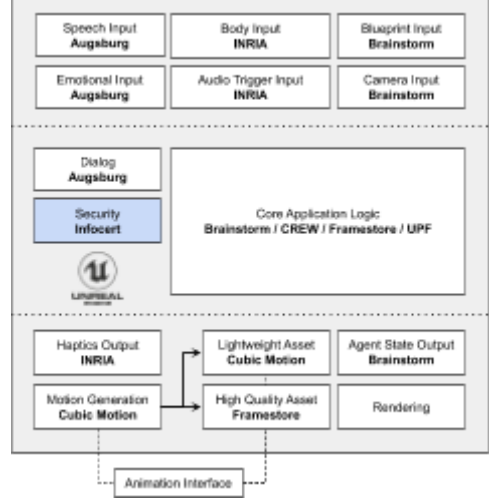
6.6. Camera Input

Description	Provides video streams from InfinitySet and/or their camera tracking data when available.	 <p>The diagram illustrates the system architecture. At the top, there are three input modules: 'Speech Input Augsburg', 'Body Input INRIA', and 'Blueprint Input Brainstorm'. Below these are 'Emotional Input Augsburg' and 'Audio Trigger Input INRIA'. The 'Camera Input Brainstorm' module is highlighted in blue. These inputs feed into a central 'Core Application Logic' block, which also receives 'Dialog Augsburg' and 'Security Infocent' inputs. The core logic is associated with the 'Unimark' logo. Below the core logic, there are three output modules: 'Haptics Output INRIA', 'Lightweight Asset Cubic Motion', and 'Agent State Output Brainstorm'. These feed into 'Motion Generation Cubic Motion', 'High Quality Asset Framestore', and 'Rendering' respectively. An 'Animation Interface' block is at the bottom, connected to the motion generation and framestore modules.</p>
Revision	2	
Owner	Brainstorm	
Implementation Details	A low latency solution (to be defined by implementation) that allows the InfinitySet render and its tracking information to be shared and used to drive Unreal camera. It is also possible to use this module to share just video or just tracking information.	
Performance/ SLA	Once the request has arrived in the engine, processing should be instant (<5ms).	
Notes	 <p>The diagram shows the 'UNREAL ENGINE' at the top, containing 'BrTexture 1', 'BrTexture 2', ..., 'BrTexture N' and 'BrCamera 1', 'BrCamera 2', ..., 'BrCamera N'. Below the engine is the 'BR PLUG-IN' layer, which includes 'BrTexture', 'BrControl', and 'Scene Capture'. The 'INFINITY SET' layer at the bottom contains 'Video Out', 'UECTL', and 'Video In'. Arrows indicate data flow: 'vout Shared' from Video Out to BrTexture; 'Items properties and methods, camera data, ...' from BrTexture to UECTL; 'Graphics triggers, agent state info, ...' from BrControl to UECTL; and 'vin Shared' from Video In to Scene Capture.</p>	
Utilised by use cases	Sports Broadcast (Brainstorm)	
Configuration	Not required	
Interface Inputs	n/a	
Interface Returns	Events notifying the application about connection and disconnection.	

6.7. Dialog

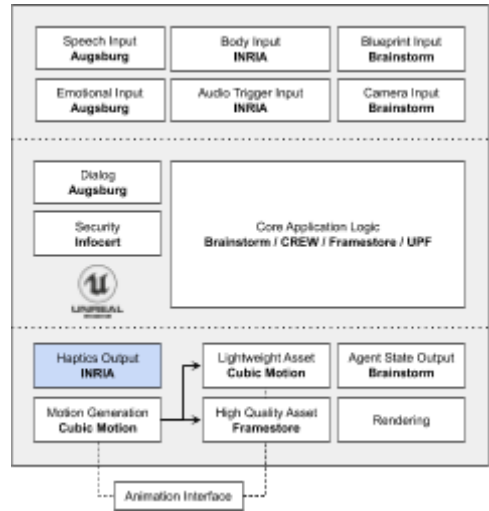
Description	The 'conversation engine'. Receives a text based query and passes back a text and audio response.	<pre>graph TD subgraph Inputs SI[Speech Input Augsburg] BI[Body Input INRIA] BSI[Blueprint Input Brainstorm] EI[Emotional Input Augsburg] ATI[Audio Trigger Input INRIA] CI[Camera Input Brainstorm] end subgraph Processing D[Dialog Augsburg] S[Security Infocent] U[UNREAL ENGINE] CAL[Core Application Logic: Brainstorm / CREW / Framestore / UPF] end subgraph Outputs HO[Haptics Output INRIA] MG[Motion Generation Cubic Motion] LA[Lightweight Asset Cubic Motion] HQA[High Quality Asset Framestore] ASO[Agent State Output Brainstorm] R[Rendering] end SI --> CAL BI --> CAL BSI --> CAL EI --> CAL ATI --> CAL CI --> CAL CAL --> HO CAL --> MG CAL --> LA CAL --> HQA CAL --> ASO MG --> HQA HQA --> R ASO --> R HO -.-> AI[Animation Interface] MG -.-> AI LA -.-> AI HQA -.-> AI ASO -.-> AI R -.-> AI</pre>
Revision	2	
Owner	University of Augsburg	
Implementation Details	Based on google Dialog Flow. Running as a containerized service outside of Unreal Engine. Sending http requests over the Internet to google. The Unreal Engine plugin is a thin wrapper around a REST or websockets interface.	
Performance/ SLA	Maximum response time 2 seconds.	
Notes	In an application setting, the current SLO of this component may be too slow to yield a realistic response. This may need to be superseded by an implementation which responds asynchronously and with low latency. Also Dialog Flow does not support embedded phoneme, viseme meta-data so another option may be needed such as Amazon Polly	
Utilised by use cases	Adam (CREW), Virtual Clerk (UPF), Sports Broadcast (Brainstorm)	
Configuration	Service connection parameters	
Interface Inputs	An input string in the form of a sentence (e.g. "What will the weather be like tomorrow in Atlantic City?")	
Interface Returns	<ul style="list-style-type: none">- A string containing a response (e.g. "Tomorrow in Atlantic City, it will be mostly sunny with showers in the afternoon")- Generated audio response- Embedded phonemes and viseme data likely needed	

6.8. Security

Description	A two-way interface for managing a secure session for the interaction between a user and an agent given the facial appearance of the user and the unique credentials of the agent.	 <p>The diagram illustrates the system architecture. It is divided into three main horizontal sections. The top section contains input modules: 'Speech Input' (Augaburg), 'Body Input' (INRIA), 'Blueprint Input' (Brainstorm), 'Emotional Input' (Augaburg), 'Audio Trigger Input' (INRIA), and 'Camera Input' (Brainstorm). The middle section contains 'Dialog' (Augaburg), 'Security Infocert' (with an Unreal Engine logo), and 'Core Application Logic' (Brainstorm / CREW / Framestore / UPF). The bottom section contains output and processing modules: 'Haptics Output' (INRIA), 'Motion Generation' (Cubic Motion), 'Lightweight Asset' (Cubic Motion), 'High Quality Asset' (Framestore), 'Agent State Output' (Brainstorm), and 'Rendering'. An 'Animation Interface' is shown at the very bottom, connected to the motion and asset modules. Arrows indicate the flow of data and control between these components.</p>
Revision	2	
Owner	Infocert	
Implementation Details	The Unreal Engine plugin will be a thin wrapper of a websockets or REST API, connecting with a containerized service which handles the authentication. Please note that this API is both input and output; requests are initiated by the agent as well as by the service.	
Performance/ SLA	Image upload should be less than 1 sec.	
Notes	<p>The high level interaction with with the authentication service would appear to the user as follows:</p> <ul style="list-style-type: none">- The user starts a session with the application, for example by approaching the setup or pressing a start button.- A QR code is presented to the user with instructions on how to scan it on a mobile device.- On the mobile device, the user consents for the application to access personal data.- Once processed on the mobile device, the application will validate the user's face. This process will be transparent to the user.- A secure session is now either initiated or denied. <p>In order for this to be integrated, the <i>application implementation</i> typically implements the following two user interactions:</p> <ul style="list-style-type: none">- UX for presenting the user with a QR code- UX and input logic for capturing an image of the user's face <p>During application execution, an application implementation is responsible to continuously authenticate in order validate the session.</p> <p>At any point, the application may re-request a session exchange to</p>	

	<p>happen.</p> <p>A separate function is required in order to check if the face is in motion so that the service cannot be tricked by a photograph of the person</p>
Utilised by use cases	n/a
Configuration	Service connection parameters
Interface Methods	<p>Authentication Invite Request from application to security plugin.</p> <ul style="list-style-type: none"> - For the very first authentication, will return an image of a QR code. - For subsequent authentications, session id is immediately returned. <p>Face scan request event - triggered by the security plugin when an image of the user's face is needed.</p> <p>Face scan upload - sends image data (of the user's face) to the security plugin. Upon success, a session id is returned.</p> <p>Is face moving/alive - function to check face is in motion to guard against static photos being used to trick the system</p>

6.9. Haptics Output

Description	Generates haptic feedback to an external device.	 <p>The diagram illustrates the system architecture. At the top, there are three input modules: 'Speech Input Augsburg', 'Body Input INRIA', and 'Blueprint Input Brainstorm'. Below these are 'Emotional Input Augsburg', 'Audio Trigger Input INRIA', and 'Camera Input Brainstorm'. A central box contains 'Dialog Augsburg', 'Security Infocent', and 'Core Application Logic: Brainstorm / CREW / Framestore / UPF'. Below this is the 'Ulm University' logo. The bottom section shows 'Haptics Output INRIA' and 'Motion Generation Cubic Motion' on the left, and 'Lightweight Asset Cubic Motion', 'Agent State Output Brainstorm', 'High Quality Asset Framestore', and 'Rendering' on the right. An 'Animation Interface' box is at the bottom, connected to the 'Motion Generation' and 'Lightweight Asset' components.</p>
Revision	1	
Owner	INRIA	
Implementation Details	The haptics system consists of custom external hardware built by INRIA.	
Performance/ SLA	- Be able to detect a haptic response within 200ms	
Notes	Multiple units may be connected.	
Utilised by use cases	n/a	
Configuration	Hardware configuration parameters.	
Interface Inputs	<ul style="list-style-type: none">- A method to trigger a haptic impulse- A method to turn on haptics output- A method to turn off haptics output	
Interface Returns	n/a	

6.10. Motion Generation

Description	Converts high level commands into character motion for both body and face for the agent.	<pre>graph TD subgraph Augsburg [Augsburg] SI[Speech Input] EI[Emotional Input] DI[Dialog] Sec[Security Infocert] end subgraph INRIA [INRIA] BI[Body Input] ATI[Audio Trigger Input] HO[Haptics Output] end subgraph Brainstorm [Brainstorm] BIP[Blueprint Input] CIP[Camera Input] ASO[Agent State Output] end subgraph Core [Core Application Logic] CAG[Core Application Logic: Brainstorm / CREW / Framestore / UPF] end subgraph Output [Output] LAM[Lightweight Asset: Cubic Motion] HQAF[High Quality Asset: Framestore] R[Rendering] end SI --> CAG EI --> CAG DI --> CAG Sec --> CAG BI --> CAG ATI --> CAG BIP --> CAG CIP --> CAG ASO --> CAG CAG --> LAM CAG --> HQAF LAM --> R HQAF --> R HO --> LAM HO --> HQAF LAM -.-> AI[Animation Interface] HQAF -.-> AI AI -.-> R</pre>
Revision	2	
Owner	Cubic Motion	
Implementation Details	The component will run as a separate executable, outside of Unreal Engine. The data will be sent from the Unreal Engine plugin Implementation via some form of IPC and will be returned using Livelihood..	
Performance/SLA	<ul style="list-style-type: none">- High level behavior decisions to be processed in < 200ms.- For mid level behavior decisions, such as muscle reaction time, state transition to new action state should happen within 33ms (1 frame at 30FPS).	
Notes	Document with more detailed specifications for this component has been composed by Cubic Motion and shared with partners	
Utilised by use cases	All Use Cases	
Configuration	- Animation configuration parameters, including any fixed animation responses that an application wishes the system to respond with.	
Interface Inputs	<ul style="list-style-type: none">- Audio waveform of the synthesised speech response- Normalised text transcription – converted to speech tokens (eg.”\$200” - two hundred dollars)- Phonemes tags – token pronunciations- Visemes tags – visual configuration corresponding to the phonemes- Paralinguistics tags – aspects of spoken communication that do not involve words- Emotional state – to be provided based on the emotion tuple (arousal, valence)- Gestural cues – inferred by the user’s data denoting the orientation and velocity of the limbs- Synchronisation information – timestamp and/or frame numbers	

	<p>- World coordinates – relative to the camera</p> <p>Format options:</p> <ul style="list-style-type: none"> - EMMA (Extensible Multimodal Annotation Language) - W3C recommendation - EmotionML - W3C recommendation - SSML (Speech Synthesis Markup Language) - W3C recommendation - BML (Behaviour Markup Language) - draft
Interface Returns	<p>- Unreal livelink with data conforming with the animation interface defined later on in this document.</p>

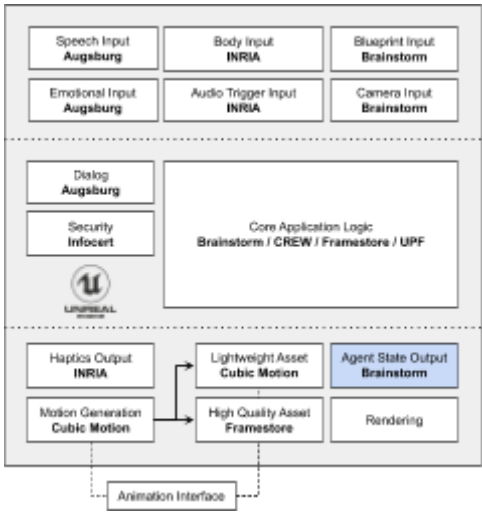
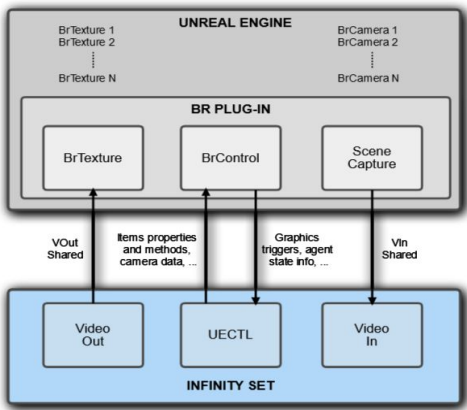
6.11. Lightweight Asset

Description	A lightweight digital human unreal asset.	<pre> graph TD subgraph Inputs SI[Speech Input Augsborg] BI[Body Input INRIA] BP[Blueprint Input Brainstorm] EI[Emotional Input Augsborg] ATI[Audio Trigger Input INRIA] CI[Camera Input Brainstorm] end subgraph Core D[Dialog Augsborg] S[Security Infocent] U[UNREAL logo] CAL[Core Application Logic Brainstorm / CREW / Framestore / UPF] end subgraph Outputs HO[Haptics Output INRIA] MG[Motion Generation Cubic Motion] LA[Lightweight Asset Cubic Motion] HQA[High Quality Asset Framestore] ASO[Agent State Output Brainstorm] R[Rendering] end AI[Animation Interface] SI --> CAL BI --> CAL BP --> CAL EI --> CAL ATI --> CAL CI --> CAL CAL --> HO CAL --> MG CAL --> LA CAL --> HQA CAL --> ASO CAL --> R MG --> LA LA --> HQA LA --> AI AI -.-> MG </pre>
Revision	1	
Owner	Cubic Motion	
Implementation Details	The asset will be assembled through an asset blueprint in UE4 and will be exposed as a skeletal mesh. A LiveLink interface will be provided with this asset blueprint.	
Performance/ SLA	90 FPS (targeting VR render). It should be possible to instance multiple characters within a single application.	
Notes	The animation data is sent on a form defined by the animation data format specification.	
Utilised by use cases	Greek Chorus, CSS, Adam (CREW), Virtual Clerk (UPF),	
Configuration	n/a	
Interface Inputs	Receives livelink data from the motion generation plugin. Data format conforming with the animation interface defined later on in this document.	
Interface Returns	n/a	

6.12. High Quality Asset

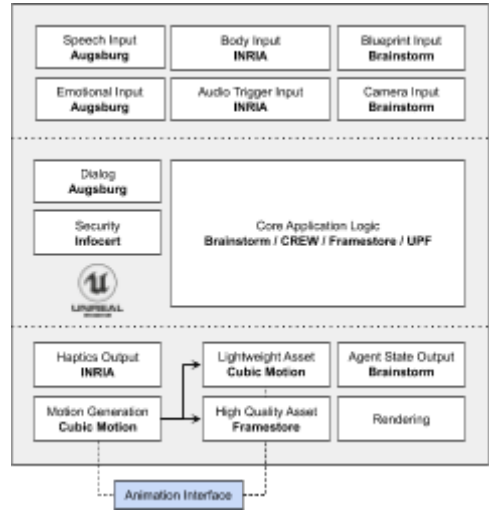
Description	A photoreal digital human unreal asset.	<pre>graph TD subgraph Inputs SI[Speech Input Augsburg] BI[Body Input INRIA] BPI[Blueprint Input Brainstorm] EI[Emotional Input Augsburg] ATI[Audio Trigger Input INRIA] CI[Camera Input Brainstorm] end subgraph Processing D[Dialog Augsburg] S[Security Infocent] CAL[Core Application Logic Brainstorm / CREW / Framestore / UPF] end subgraph Outputs HO[Haptics Output INRIA] LA[Lightweight Asset Cubic Motion] ASO[Agent State Output Brainstorm] MG[Motion Generation Cubic Motion] HQAF[High Quality Asset Framestore] R[Rendering] end SI --> CAL BI --> CAL BPI --> CAL EI --> CAL ATI --> CAL CI --> CAL CAL --> HO CAL --> LA CAL --> ASO MG --> HQAF HQAF --> R MG -.-> AI[Animation Interface] HQAF -.-> AI AI -.-> MG AI -.-> HQAF</pre>
Revision	1	
Owner	Framestore	
Implementation Details	The agent will be assembled through an asset blueprint in UE4 and will be exposed as a skeletal mesh. A LiveLink interface will be provided with this asset blueprint.	
Performance/ SLA	30 FPS (targeting screen render).	
Notes	The animation data is sent on a form defined by the animation data format specification.	
Utilised by use cases	Virtual Production (Framestore), Sports Broadcast (Brainstorm)	
Configuration	n/a	
Interface Inputs	Receives livelink data from the motion generation plugin. Data format conforming with the animation interface defined later on in this document.	
Interface Returns	n/a	

6.13. Agent State Output

Description	Output high level information about the character state, and events to trigger actions, to an external application.	
Revision	1	
Owner	Brainstorm	
Implementation Details	An open, well documented data exchange using the InfinitySet existing protocol. Used to connect to a system such as Brainstorm InfinitySet. System is threaded and non-blocking.	
Performance/SLA	The performance of this module will be largely dependent on the connection from to the receiving client, so a SLO is not practical to define.	
Notes	The Unreal Engine plugin implements a server that one or more clients can connect to in order to receive information asynchronously. No authentication is required in order to connect.	
Utilised by use cases	Sports Broadcast (Brainstorm)	
Configuration	Connection configuration.	
Interface Inputs	<ul style="list-style-type: none">- Ability to query which clients are connected- Ability to send high level emotional state metadata to connected clients. This will be in the form of a well documented json structure with parameters for mood, animation state, position etc.	

	- Ability to send trigger data to control the InfinitySet scene graphics.
Interface Returns	n/a

6.14. Animation Data Format Specification

Description	A data format specification to ensure that the animation data streamed from the motion generation plugin can be consumed correctly by the render plugin.	 <p>The diagram illustrates the system architecture. At the top, there are three input modules: 'Speech Input Augsburg', 'Body Input INRIA', and 'Blueprint Input Brainstorm'. Below these are 'Emotional Input Augsburg', 'Audio Trigger Input INRIA', and 'Camera Input Brainstorm'. A central box labeled 'Core Application Logic: Brainstorm / CREW / Framestore / UPF' is connected to a 'Dialog Augsburg' and 'Security Infocent' module. Below this, there is a 'Motion Generation Cubic Motion' module which feeds into a 'Lightweight Asset Cubic Motion' and a 'High Quality Asset Framestore'. The 'Lightweight Asset Cubic Motion' also feeds into the 'High Quality Asset Framestore'. The 'High Quality Asset Framestore' feeds into 'Rendering'. An 'Animation Interface' module is shown at the bottom, connected to the 'Motion Generation Cubic Motion' and the 'High Quality Asset Framestore'. On the left side of the diagram, there are 'Haptics Output INRIA' and 'Agent State Output Brainstorm' modules.</p>
Revision	1	
Owner	Cubic Motion / Framestore	
Implementation Details	<p>Body Rig:</p> <p>Framestore and Cubic Motion will align on a common skeletal mesh for the body rig - Framestore will initially provide a skeleton to Cubic Motion on which to derive the CM version of the body rig.</p> <p>Face Rig:</p> <p>Framestore and Cubic Motion will align on common blend shapes and GUI controls between their versions of the rig. Framestore will provide Cubic Motion a diagram mapping each control in the UI for retargeting between the 2 versions of the facial rig.</p>	

6.15. Core Application Logic

Description	Main implementation that connects all components into a functional system.	<pre>graph TD subgraph Inputs SI[Speech Input Augsburg] BI[Body Input INRIA] BP[Blueprint Input Brainstorm] EI[Emotional Input Augsburg] ATI[Audio Trigger Input INRIA] CI[Camera Input Brainstorm] end subgraph Core CAL[Core Application Logic: Brainstorm / CREW / Framestore / UPF] end subgraph Outputs D[Dialog Augsburg] S[Security Infocent] HO[Haptics Output INRIA] MG[Motion Generation Cubic Motion] LA[Lightweight Asset Cubic Motion] HQA[High Quality Asset Framestore] ASO[Agent State Output Brainstorm] R[Rendering] end SI --> CAL BI --> CAL BP --> CAL EI --> CAL ATI --> CAL CI --> CAL CAL --> D CAL --> S CAL --> HO CAL --> MG CAL --> LA CAL --> HQA CAL --> ASO CAL --> R MG --> LA MG --> HQA LA --> HQA HQA --> R AI[Animation Interface] -.-> MG AI -.-> HQA</pre>
Revision	n/a	
Owner	One implementation per use case.	
Implementation Details	The core application logic takes all input, ensures that the user is still authenticated, requests a response from the dialog plugin and based on that creates output state for the agent which is sent down to the motion generation plugin for processing. The application also implements rendering and output from Unreal engine, e.g. the logic and configuration for how the pixels should leave the system (VR, Screen, ipad etc).	
Performance/SLA	n/a	
Notes	Please note that this is not a component or unreal plugin – instead it defines the work that needs to be done by the integration partners in order to integrate the software components into a fully functioning system.	



Note: Any functionality or logic not outlined as components in this document is assumed to be handled as part of the application logic.

The breakdown in Figure 7 outlines how an application could be designed given the interfaces presented by the various components to form a fully functioning system as defined by the main PRESENT specification. The core application logic is broken down into four distinct parts:

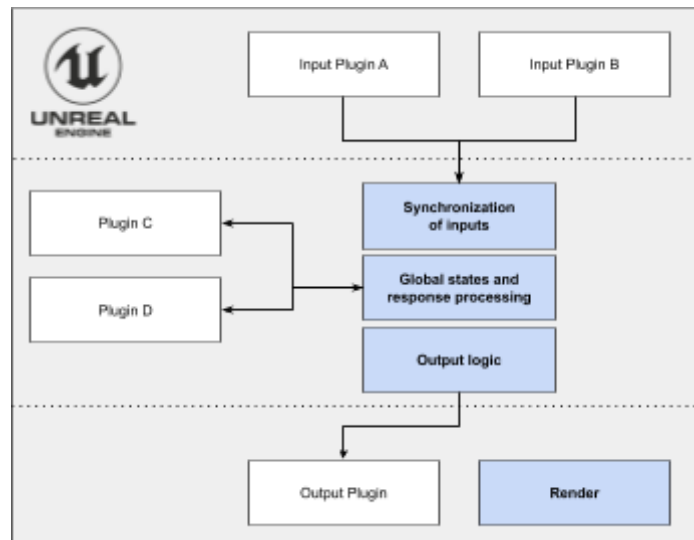


Figure 7: Core Application Logic

Initialisation

An application wishing to implement the security module would need to start by establishing an authenticated session between the user and the system. This currently involves an UX process where a QR code is read in and passed down to the security plugin.

Synchronisation of inputs

At this stage, the goal is to create coherence at the input level. Inputs generated by the speech plugin may have a significant lag (up to 2 seconds after a sentence has completed). These need to be synced with the inputs from the visual input plugin, which may have as little as one frame, or 33ms, of lag. This is done in a smooth fashion, such that the system response (as seen through the agent) is coherent over time.

Once synchronized, speech data as well as the position and emotional state of the user should be available and can be passed onto the next stage.

Global states and response processing

At this stage, the goal is to react to the response collected through the inputs. The logic for generating a response will depend on the input:

- If the user is not talking, determine an 'idle state' for the agent given the mood of the user and what previous state the agent was in.
- If speech is incoming via the inputs, forward it on to the dialog plugin for processing and switch the agent to a state where it is getting ready to talk
- If a speech response is coming back from the dialogue component, prepare for the processing of this.
- Authentication of the current user happens continuously.

A state will be maintained in order to track over time the emotional state of the agent and based on this determine the appropriate animations to request. This is where the application rules are implemented and how the specific system responses will be coded.

Output logic

This part is relatively simple. The goal of the output logic is to send high level commands to the motion generation plugin through a buffered output classes interface. It will also receive information from the motion generation plugin when an animation response has completed, allowing the output logic to feed this back into the state processor.

Rendering

Rendering is separate from the simulation logic and encompasses the code and configuration needed in order for the pixels to leave Unreal engine and be drawn on a display device or similar. Unreal engine supports a wide range of devices, such as screens, VR headsets and tablets, and in the case the output is not supported, the application developer will need to extend the system appropriately.

6.16. Components not covered in this report

6.16.1. Super Render Computer (WP6T2)

WP6T2 covers the building of a high-performance low latency “super render computer”. Given advances in GPU technology since the proposal was written we no longer anticipate that this approach will be required.

7. HIGH LEVEL AGENT CREATION ARCHITECTURE

A rig defines the interface between animation and the movement of the agent geometry. Animation is therefore tied to the rig and geometry to which it is targeted. The agent creation architecture is therefore split into two rigs. The 'High Fidelity' and 'Lightweight' rigs.

Framestore will create the 'High Fidelity' Rig. Cubic Motion will create the 'Lightweight' rig. A common interface will be defined between Framestore and Cubic Motion which will allow animation generated from the Cubic Motion performance component to drive both rigs.

As outlined in the methodology, Framestore will deliver a single, high resolution agent for the project on the 'High Fidelity' rig. Cubic Motion will produce the same agent on the 'Lightweight' rig.

Cubic Motion will create a 'Lightweight' rig and will licence out the plugins necessary to utilise this rig for the duration of the project.

Framestore and Cubic Motion will design and supply an interface specification that clearly lays out the rig structure required for system compatibility. This will enable other partners to create agent rigs in future that can be driven by the system using the same user interface or via a retargeting module.

8. CHANGE MANAGEMENT

Over time, changes to the interface specifications are expected. This section outlines the process for how such a change is managed.

8.1. Identification and Proposal

A need arises - usually in the process of utilizing a component in an application. As a consequence, the relevant partners liaise and discuss a proposal for how the interface could evolve. If relevant, a collaborative prototype is carried out in order to validate that the interface change is relevant. A written proposal explaining the details of the change as well as why the change is needed is drafted and distributed to the partners.

8.2. Approval


The change is presented to the architecture board - ideally along with the practical prototype to demonstrate the need as well as the implementation. The architecture board reviews and approves the change.

8.3. Re-Release

The architecture specification is re-released to all partners. The component interface version number is incremented. Framestore re-releases the reference implementation to implement the interface in a way that demonstrates the added functionality in a relevant way.

8.4. Component Update

Lastly, the partner responsible for providing the component re-releases the component with support for the new interface. The version number of the component should reflect the version number of the component interface.

 **Example:** If the component interface was version 12 and the most recent component release was v12.3.1, an interface change would result in the component interface version being incremented to become 13. The initial release of a component to support this interface would be v13.0.0.

9. REFERENCE IMPLEMENTATION

The current version of the reference implementation - v2.1.1 - is released to all partners via the Framestore sFTP.

9.1 Components

Reference implementation v2.1.1 divides the components according to Figure 8.

The following components are represented in v2.1.1:

- Emotional Assessment
- Audio Processing
- Text to Speech
- Action Response
- Motion Generation

The components associated with integrating Brainstorms InifinitySet software as well as InfoCert's security API's are not yet implemented. These components will be supported in future releases.

Each component is represented as a Third Party Plugin in the Unreal Engine reference implementation project. Part of each plugin is therefore the library (dll) that each partner can edit as they see fit, and the other part is the integration of that library (dll) with a mock implementation. The plugins are integrated through a wrapping actor component, and linked in the project between actors.

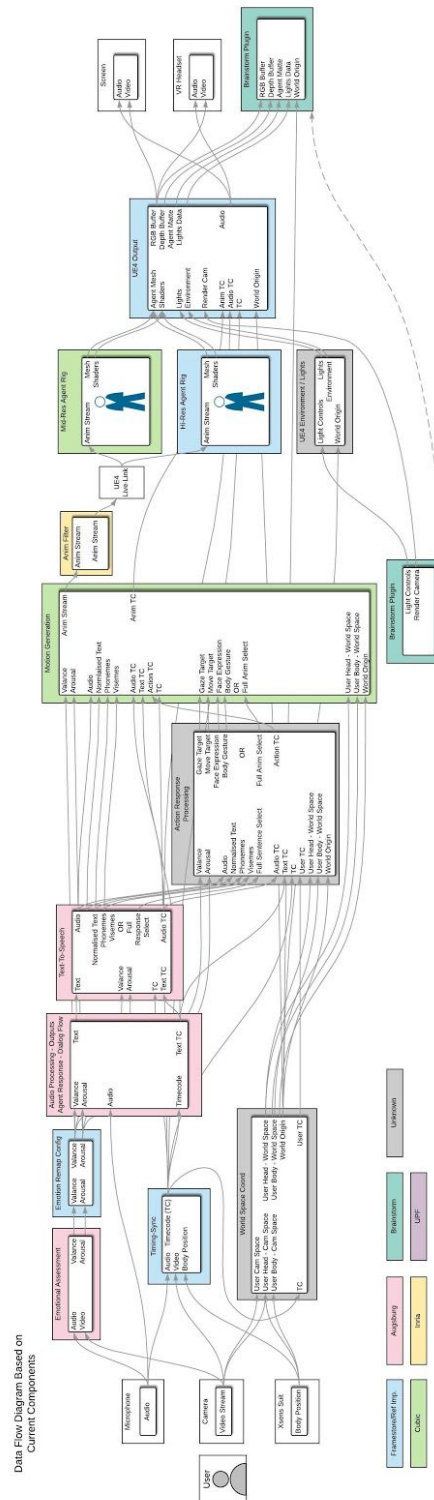
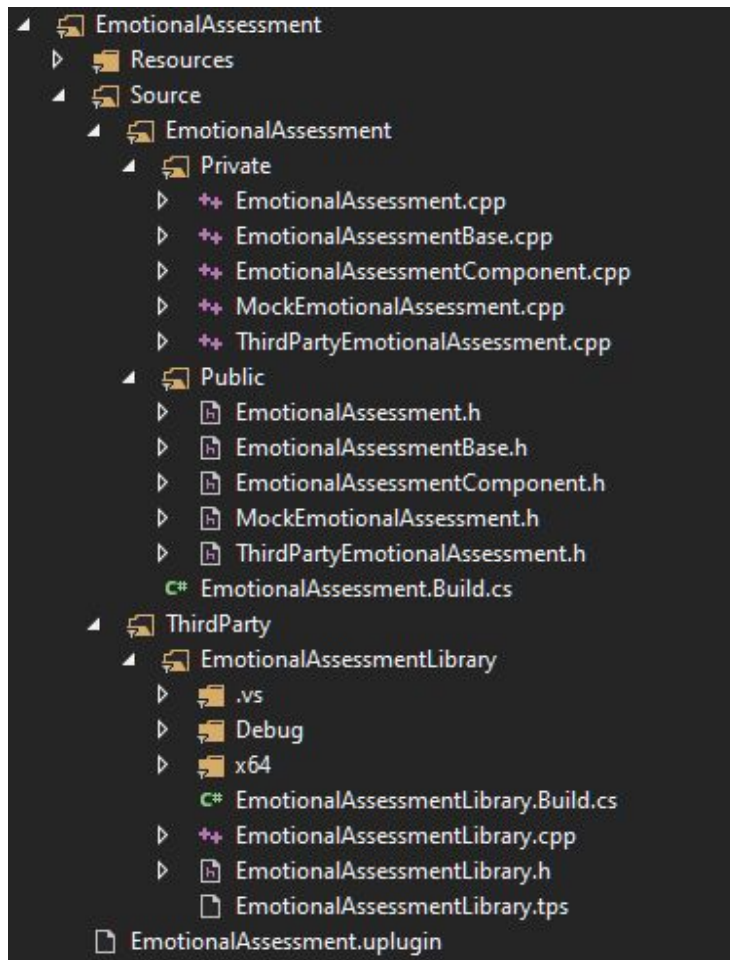


Fig. 8. Reference Implementation Schematic

Below is a typical breakdown of the components - the example given is the Emotional Assessment plugin :



> Everything under there is the integration of the library and any other code required for the reference implementation project to use the partner's library, as well as the mock implementation.

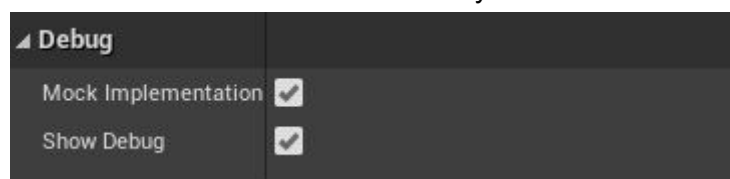
> Everything under ThirdParty belongs to the Partner. It can be any kind of project/sources, as long as it a generated dll with the correct signature under x64/Release/. That dll must also be found in the Binaries/ThirdParty folder of the plugin

The aim with this structure is to have a base class that can be derived from, to allow for any implementation required. In this case, the first is the Mock Implementation, which fakes the feature, and the second one is the ThirdParty Implementation which uses the partner dll.

The integration is done through an Actor Component, which controls the switch between each implementation when the engine is running. It is a wrapper of the implementation integration, to expose it to the blueprint scripting language in the editor.

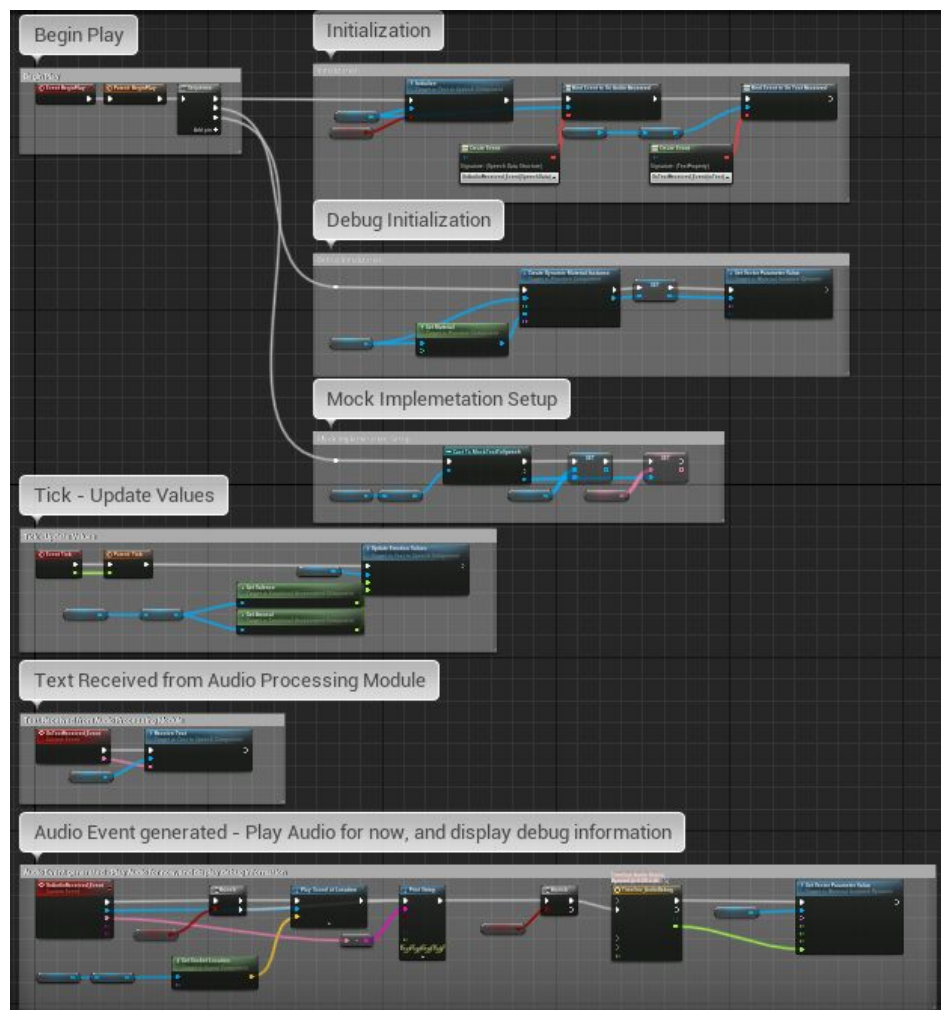
Each actor using these components derives from BP_ModuleBase, which contains variables for switching from mock to real implementation, and logic for debug information.

You have access to 2 variables to control that on every module actor.



Each module actor does its integration of the module c++ component, as well as linking with other modules through scene references and events.

The component is initialized first, along with its required events. Then steps are added to control the mock implementation variables, and the debug information. These steps will be ignored when variables described above are unchecked. Here's an example of the TextToSpeech Actor setup.



9.2 Mock Variables

These are the exposed variables used to change the behavior of the reference implementation for each module.

Emotional Assessment:

No variables are required for this component, it uses sin and cos functions over time to generate valence and arousal.

Audio Processing:

A list of text sentences that are broadcast with Unreal events once every “MinDelay” to “MaxDelay” seconds.

Mock	
Sentences	11 Array elements + -
Min Delay	5,0 -
Max Delay	8,0 -

Text To Speech:

A list of sound wave audio files associated with its translation in text. The mock implementation choses a random entry in the list when it receives text from the audio processing module.

Mock	
Audio Files	6 Array elements + -
Audio to Text	6 Array elements + -

And an additional variable controlling whether this module should play the audio file or not.

Debug	
Play Audio	<input checked="" type="checkbox"/>

Action Response:

The offset from the user position where the digital human will move.

Mock	
Move Target Offset	X 150,0 - Y 0,0 - Z 0,0 -

Motion Generation:

A list of animation sequences that are used randomly when a response is triggered.

Mock	
Animation Sequences	3 Array elements + -

9. 3 Third Party Signatures

These are all subject to change. The reference implementation is in its early stage releases and all variables are subject to requests and feedback from the partners concerned with the components implementation. Framestore fully expect to be iterating on the exact configuration of the component interfaces in the reference implementation throughout the project.

Every module was created with a method that is being called when the module is loaded in the editor. It is here for convenience if dlls need single time initialization.

```
void OnModuleLoaded();
```

Emotional Assessment:

Gives access to a valence and arousal value at any time.

```
float GetValenceValue();
float GetArousalValue();
```

Audio Processing:

This module keeps track of the valence/arousal values and should send an event when a sentence is recognized from the audio feed and an answer is found..

```
typedef void (__stdcall *EventCallback)(const char*);
void UpdateEmotionValues(float valence, float arousal);
void RegisterEventCallback(const EventCallback& func);
```

Text To Speech:

This module listens for text sentences sent from the audio processing module and generates audio data to be played to the user, based also on the emotion status of the user. It sends that audio event in the form of a SpeechData object.

```
typedef void (__stdcall *EventCallback)(SpeechData);
void ReceiveProcessedText(const char* text);
void UpdateEmotionValues(float valence, float arousal);
void RegisterEventCallback(const EventCallback& func);

struct SpeechData
{
    const char* AudioFile;
    const char* NormalizedText;
    const char* PhonemesData;
    const char* VisemesData;
};
```

Action Response:

This component listens for the speech data event in order to choose from a range of actions to give life to the digital human response.

```
typedef void (__stdcall *EventCallback)(ActionData);
void RegisterEventCallback(const EventCallback& func);
void UpdateEmotionValues(float valence, float arousal);
```

```
void UpdateTargetValues(Vector3d headPosition, Vector3d bodyPosition);
void ReceiveAudioEvent(SpeechData speechData);

struct ActionData
{
    Vector3d GazeTarget;
    Vector3d MoveTarget;
    const char* FaceExpression;
    const char* BodyGesture;
};
```

Motion Generation:

This module is the last one in the chain to create a reaction. It receives information from the other modules to generate the final animation of the character. The output is yet to be determined based on feedback from Cubic Motion.

```
typedef void(__stdcall *EventCallback)(const char*);
void RegisterEventCallback(const EventCallback& func);
void UpdateEmotionValues(float valence, float arousal);
void ReceiveAudioEvent(SpeechData speechData);
void ReceiveActionEvent(ActionData actionData);
```

10. TESTS, VERSIONING AND DISTRIBUTION

This section describes the various processes for alignment which will be operating during the project in order to ensure that all partners can establish the right expectations on other partners and be able to collaborate, iterate and mitigate risk.

10.1. Functional tests

The reference implementation will be the main functionality alignment vehicle. The validity of the functionality within a given release of a software component is practically demonstrated by its integration into the most recently released reference implementation.

10.2. Unit tests

Accountability for unit testing resides with each component provider. Partners authoring test plugins will share their test plans but partners will have the choice of choosing the testing process they choose fit for the project and their needs. It is not a requirement that unit testing has to happen within Unreal Engine.

10.3. Versioning

Each software component and the API for each software component is versioned using semantic versioning, as outlined at <https://semver.org/>. The version of the interface specification should be reflected in the major version number, e.g. if a software component implements version 3 of a component interface, it should be versioned as v3.x.x.

10.4. Component Distribution

Each component provider is responsible for making available all the released versions of their software components to all other project partners in such a way that integration is reasonably straight forward:

- Each release includes details of features added and bugs addressed.
- Each release includes installation instructions, outlining the practical steps for integration.
- If a software component contains complex services, a [dockerfile](#) and/or docker compose file is included, allowing any partner to easily stand up all services required.

11. USE CASE REFLECTIONS

In order to ensure the architecture is relevant and appropriate, this section discusses a number of the PRESENT use cases and how each one of them would be realised given the components and principles introduced in this document.

11.1. Adam 0.1

Use Case Leader: CREW

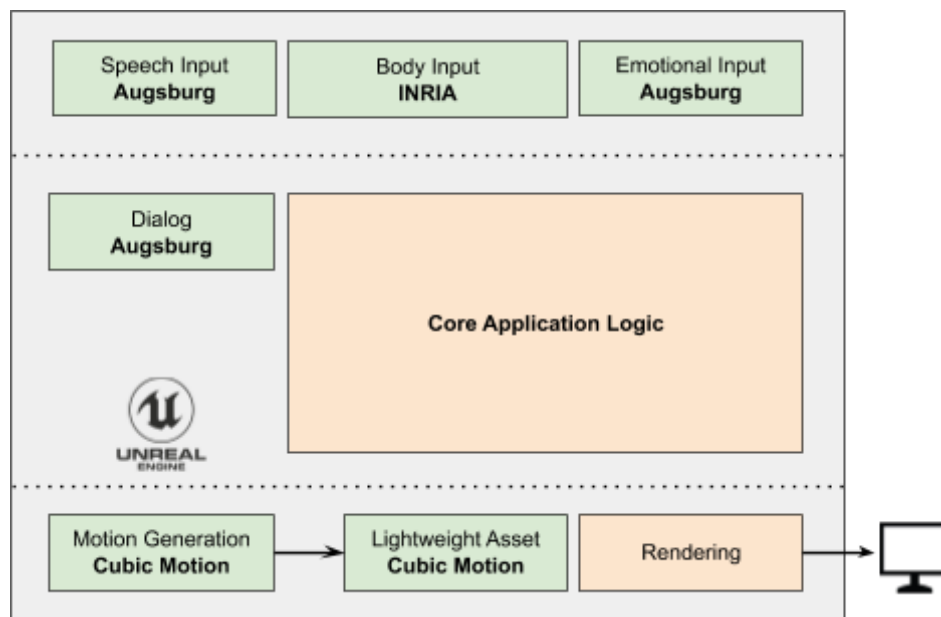


Figure 9: Adam 0.1

- Adam receives speech, body and emotional input from cameras and audio inputs.
- Adam's application logic implements gesture recognition.

Note: Adam's dialog generation may be too complex to be able to utilize the basic google dialog flow based component implemented by Augsburg. In this case, the same dialog interface can be utilized but with an augmented component.

11.2. Complex Social Situation

Use Case Leader: CREW

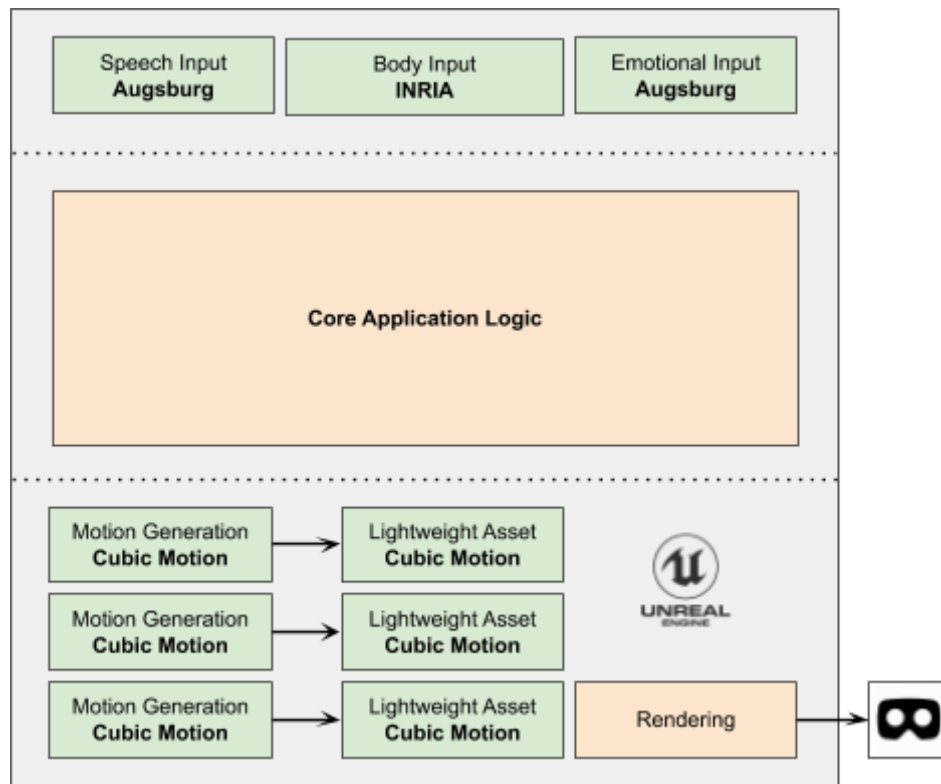


Figure 10: Complex Social Situation

- Posture and hands based non-verbal input is generated from the body input plugin
- Speech and emotional input augments the non-verbal input but is used by the application to a lesser degree.
- Multiple lightweight characters are rendered for VR output

11.3. Greek Chorus

Use Case Leader: CREW

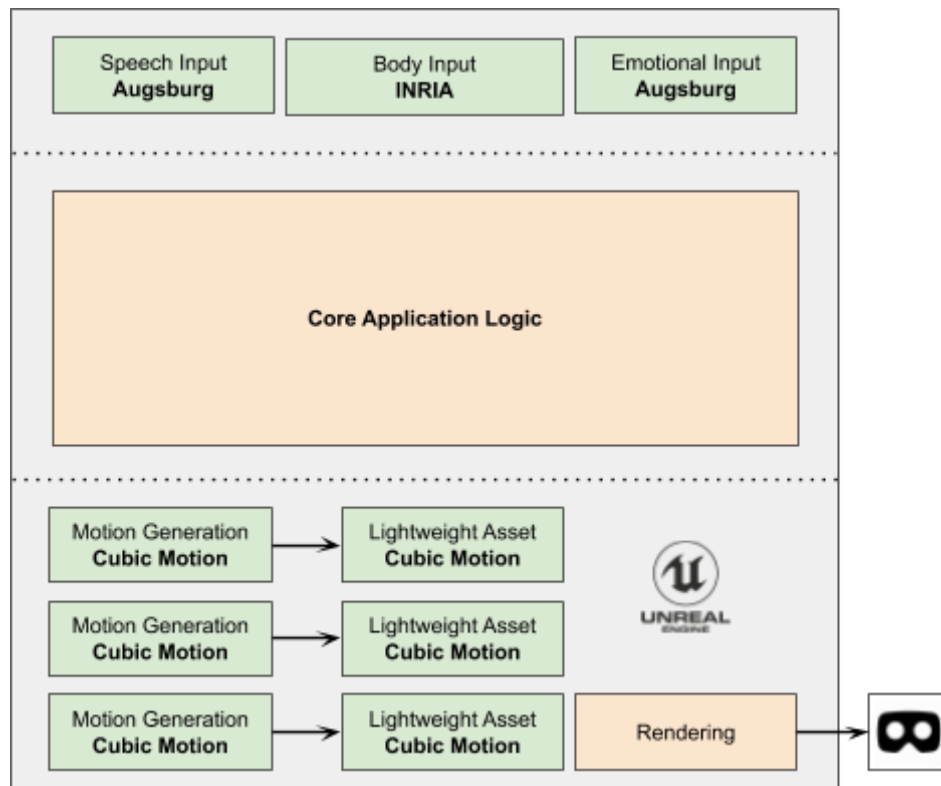


Figure 11: Greek Chorus

- User speech is input in both audio and text form along with emotional input and body position.
- Core application logic generates synthesized speech as well as physical expression and positions for multiple characters.
- The user experiences these multiple avatars in VR.

11.4. Sports Broadcast

Use Case Leader: Brainstorm

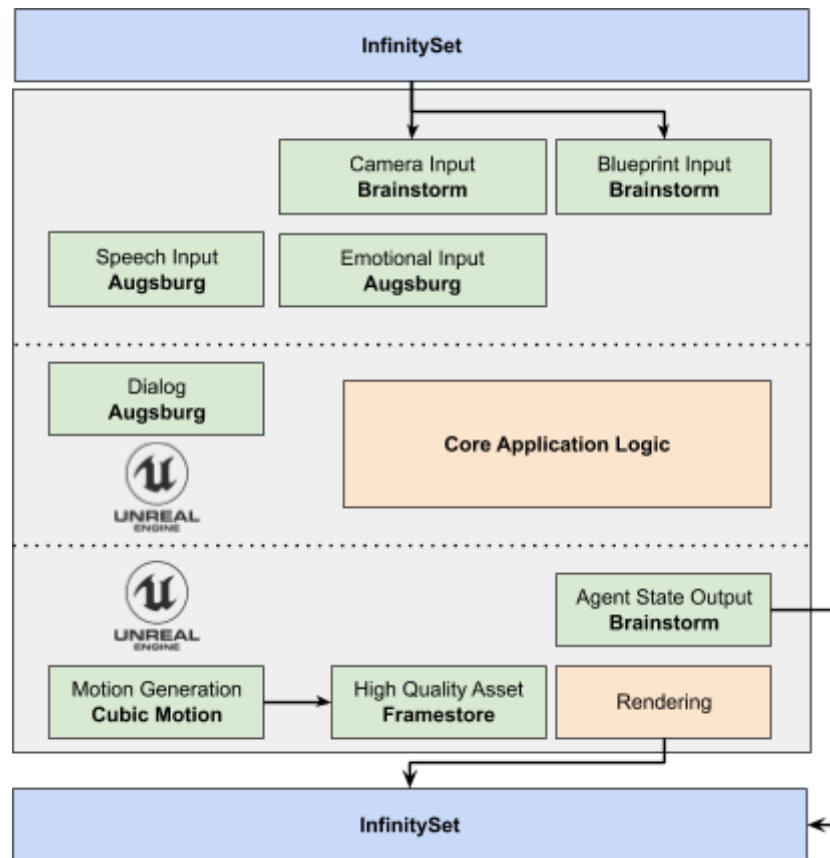


Figure 12: Sports Broadcast

- Camera input, positioning of key 'points of interest' in the virtual studio as well as triggers for graphics are input from infinitySet to the application via camera and blueprint inputs.
- Audio and video feeds of a user are turned into speech and emotional input.
- The application reacts to the emotional state of the user, uses the interpreted audio to respond with one out of a fixed responses that were prepared in advance of the session.
- High level agent states (key position, emotional state, etc), as well as events, are flagged to InfinitySet via the agent state output component. This will allow controlling Aston graphics in InfinitySet.
- The application generates rendered output in a way that is suitable to stream back into InfinitySet.



Note: Streaming video from InfinitySet into Unreal engine's rendering loop may require a bespoke solution. From an architecture point of view, this is considered to be within the application logic.

11.5. Virtual Clerk

Use Case Leader: UPF

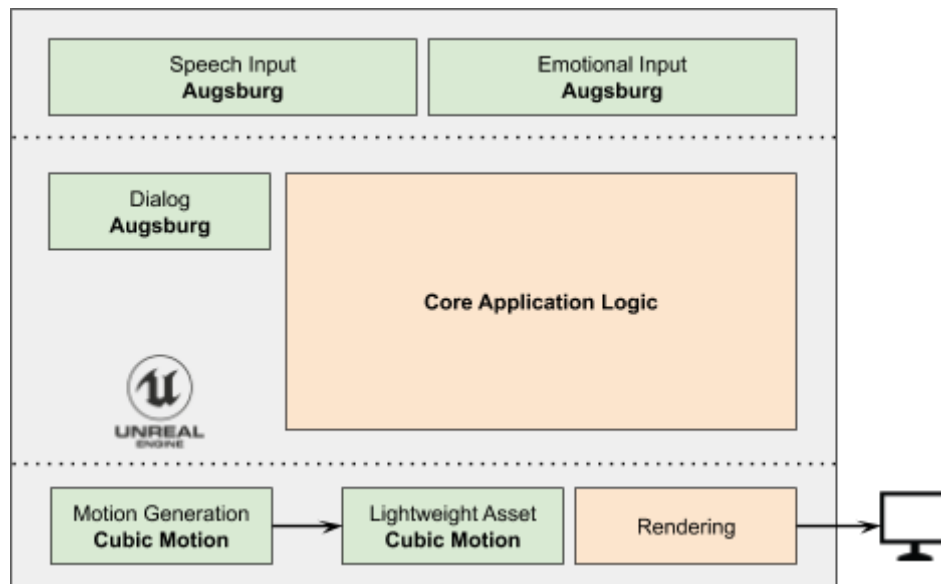



Figure 13: Virtual Clerk

- User speech and emotional input is analyzed by the application logic
- A logical as well as audio response is generated via the google dialog flow based dialog component.
- Logic for displaying images, maps and related media is handled by the application.
- The response is rendered via the lightweight asset in order to be able to run the application on affordable hardware.

 **Note:** In this use case, the dialogue component may need to be augmented in order to provide specific instructions

11.6. Virtual Production

Use Case Leader: Framestore

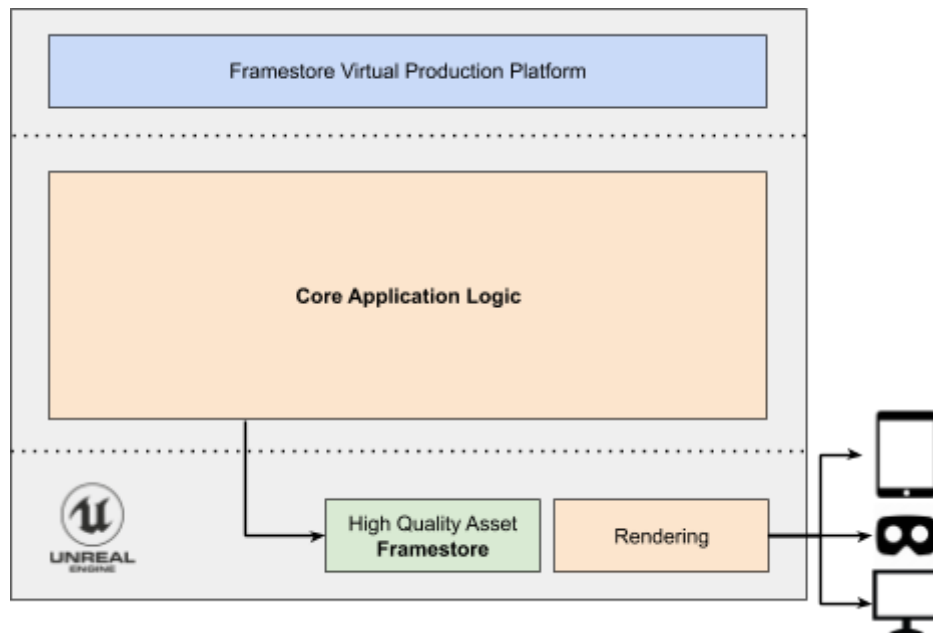



Figure 14: Virtual Production

- Framestore's proprietary virtual production technology is used for all inputs into the application.
- The performance of the high quality asset is driven directly from the application logic, stemming directly from the performance capture.

 **Note:** In this use case, the user and the agent are the same person. The user's motion is used to drive the agent's behaviour directly, rather than the agent being a different individual who is responding to what the user is doing.

12. CONCLUSIONS

The architecture presented here forms a solid foundation to the project and clearly defines partner responsibilities and interfaces. The reference implementation complements this and provides a strong platform from which partners can build. A number of components are yet to be completed within the reference implementation and many more iterations are anticipated over the course of the project based on partner feedback. However foundational progress has been made and key elements of the architectural design laid out.

The design evolution of individual components, interfaces and system architecture will be monitored in the ongoing bi-weekly meetings between partners. There are also defined points in the schedule called out to assess updates to the Unreal Engine version.

Whilst not to be overstated, it should be acknowledged that there is a tension in the project between modularity and performance. Whilst the use of REST APIs and live links in Unreal Engine to link work packages are deemed to be the best interfaces for the project goals, it is also at odds with the active pursuit of overall system performance (overall system latency and human computer interactivity). This trade off is accepted as the best compromise. It gives partners freedom to work in their own established development environments and workflows to create the best components and functionality. Whilst we acknowledge this as an overall architecture and interface compromise, the relative largest latency in the system is not within the interfaces, but instead carried in the individual components.

Whilst gaps have been identified in the work packages, there are proposed solutions to fulfil the aims of the project, and given the active research by other parties in this area, the consortium will be looking for potential third party solutions that can further improve the outcomes of the project. The modular design architecture compliments this philosophy and provides flexibility to benefit from external research and components.

APPENDIX A - PROJECT TERMINOLOGY

- A**
- **Agent** : Potential term for the proposed photoreal, realtime computer generated character that is the subject of this grant
- B**
- **Brainstorm** : **Partner** and developer of Virtual Studio broadcast software
- C**
- **Call And Response** : Initial instantiation of conversation and beginning of a single session
 - **Christina EU project** : Previous grant project Uni of Augsburg and UPF
 - **Cubic Motion** : **Partner** and facial and body motion capture and animation specialists
 - **Crew Online** : **Partner** and Belgium live performance based entity specialising in the use of VR in live environments
- D**
- **DCC** : Digital Content Creation tools used to create the base modeling, rigging, animation, lighting and compositing aspects necessary to create the CG character and environment
 - **DeepMind** : Google AI powerhouse
 - **Dialogue Management** : Process of capturing conversational persistence beyond the initial “call and response” (see: Google Dialogflow)
 - **Dialogue Persistence** : A subset of Dialogue Management providing response consistency across multiple sessions, including indexing into a *fuzzyDB* where the information is delivered in an intentionally general way. I.e; remembering too much detail could prove disconcerting to the User
- E**
- **Emotion Engine** : A set of modules designed to take human user input, process that input for meaning, index the result into a response criteria and package that response ultimately for presentation to the CG Agent.
 - **Emotional Metadata** : NEED to define input vs output. Emotional Metadata(in) is derived from Speech Emotion Recognition with influence from dialogue content and as EM(out) it represents a payload delivered from the Emotion Engine that influences the voice and facial response potential for the Agent combining with output from the Knowledge Mining module.
- F**
- **FPGA** : Field Programmable Gate Arrays : a computer processing technique that allows for the developer to produce a domain specific hardware architecture in order to increase the computational ability of a specific computing node. A potential alternative to GPU technology.
 - **Framestore** : **Partner** and specialist Film, Advertising and Experiential visual effects company
- G**
- Google **Dialogflow**: An end-to-end, build-once deploy-everywhere development suite for creating conversational interfaces.
- I**
- **Infiniband** : A high throughput, low latency network designed to facilitate node based parallel processing
 - **InfoCert** : **Partner** and pioneer in Sovrin Trust network specialising in digital identity verification and persistence
 - **INRIA** : **Partner** and French university and research institute.
 - **IPR** : Intellectual Property Rights. In the context of the grant process this is represented as “foreground” (IP that existed before the project began) and “background” (IP generated as a product of the grant process)
- K**
- **Knowledge Component** : The Agent response to User Interaction is encapsulated in a Response Envelope comprising two main deliverables: the Emotion Metadata and Knowledge Component
 - **Knowledge Discovery** : In the context of the proposal, these are the methods applied to the collected data gathered via multiple User <> Agent exchanges and used to provide insight and guidance into further dialogue sessions

- **Knowledge Mining** : Variation on data mining, knowledge discovery, knowledge extraction. An approach to indexing into domain specific knowledge to provide the Knowledge Component of the Response Envelope
 - **Knowledge Representation** : Guidelines for structuring initial database lookups for performance and indexing
- L
- **Letter(s) Of Support** : from non-grant parties endorsing proposals. Currently: Google, Nvidia, Epic Games, Datatonic
- M
- **Modeling (Character)** : The process of generating computer based geometry to match the physical properties of the object in question
 - **Motion Capture** : The process of capturing human or animal motion via a variety of methods and digitizing that motion with the purpose of retargeting the physical motion on to a CG (computer generated) object
- P
- **Photoreal** : In the context of the proposal this is a computer generated character whose skin quality, emotional characteristics and overall lighting are indistinguishable from a photograph of the same target individual.
- R
- **Ray Tracing** : A graphics rendering technique based on computing rays of light in order to produce physically plausible representations of computer generated scenes
 - **Real-time Rendering** : In the context of the proposal this is the ability for the CG character to visually respond to user input in a natural manner and with minimal noticeable latency
 - **Reinforcement Targets** : Potential positive outcomes for a dialogue session as proposed by Bjorn Schuller. (ie: to cheer someone up)
 - **Response Envelope** : The accumulated payload from both the Emotion Engine (metadata) and
 - **Rigging** :
 - **RTX Graphics** : A next generation GPU from Nvidia that incorporates ray tracing techniques significantly improving photorealism
- S
- **Sentiment Analysis** : valence recognition from text or other human signals as related to an object of sentiment
 - **Smart Prompt** : TODO design for idle loop periods or the “awkward silences” present or potential in all conversations. Depending on URT may prompt user for more engagement or wait patiently.
 - **[Sovrin Network](#)** : Digital Identity framework based on blockchain and digital ledger (DLT) technologies. Within the proposal, Infocert are members of the consortium.
 - **[Speech Emotion Recognition](#)** : recognition of emotion in classes (such as the “big 6”) or dimensions (such as arousal or valence) from acoustic and linguistic properties in the speech signal
- T
- **TaxiDriverMode** : “Are you talking to me?”
 - **[Time Well Spent](#)** : a movement based on shifting the User <> Computer interaction from the current Time On Site model to a more user-centric Time Well Spent model as advocated by [Tristan Harris](#).
 - **Turing Test** :
- U
- **University of Augsburg** : **Partner** and European leader in Speech Emotion Recognition
 - **Unreal Engine** : popular game engine from Epic games
 - **UPF** : **Partner** and project administrative lead. Shorthand for Spanish Universitat Pompeu Fabra located in beautiful Barcelona
 - **User Expression Template (UET)** : A set of questions designed to prompt the human user in order to better understand that users preferences for dialogue and interaction. At a base level, it represents an evolution of the [Myers-Briggs Type Indicator](#) test. See also: [Rogerien Psychotherapy](#))
 - **User Preferred Response Envelope (UPRF)** : Variation on the URT below.
 - **User Response Template (URT)** : A software filter representing the users preferred communication methods and style

V

- **VFX** : general term for film visual effects
- **VizRT** : Virtual Studio broadcast software
- Volumetric Solver :

W

- **Wavecrest** : **Project coordinator** and specialist in preparation of grant funding proposals
- **WP** : Work Package. The unit of work assigned to a subcomponent of the overall grant process

Z

- **Zero Density** : Virtual Studio broadcast software