WHOLODANCE

Whole-Body Interaction Learning for Dance Education

Call identifier: H2020-ICT-2015 - **Grant agreement no**: 688865

Topic: ICT-20-2015 - Technologies for better human learning and teaching

Deliverable 4.3

Analysis and Integration of Generic Application Framework

Due date of delivery: January 31st, 2018

Actual submission date: February 23rd, 2018

Start of the project: 1st January 2016
Ending Date: 31st December 2018

Partner responsible for this deliverable: POLIMI

Version: 4.0

Dissemination Level: Public

Document Classification

Title	Analysis an Integration of Generic Application Framework
Deliverable	4.3
Reporting Period	M1-25
Authors	Massimiliano Zanoni, Michele Buccoli, Augusto Sarti, Fabio Antonacci
Work Package	WP4
Security	Public
Nature	Report
Keyword(s)	Software platform, software libraries, application framework, software integration

Document History

Name	Remark	Version	Date
Massimiliano Zanoni	TOC	0.1	24/01/2018
Michele Buccoli	Data Exchange	0.2	30/01/2018
Stefano Piana	Added Feature extraction engine		05/02/2018
Stefano Piana	Added contribute to unity-based integration		08/02/2018
Katerina El Raheb, Akrivi Katifori, Aristotelis Kasomoulis, Marianna Rezkalla, George Tsampounaris	Contribution to Sections 3, 4 and 5 related to the work of Athena RC		8/02/2018
Oshri Even-Zohar	Contribution to the motion capture and the unity integration sections (3&5) related to the work of Motek entertainment.		08/02/2018

Augusto Sarti, Massimiliano Zanoni	Final Review	1.0	17/02/2018

List of Contributors

Name	**Affiliation**
Massimiliano Zanoni, Michele Buccoli, Augusto Sarti	Polimi
Katerina El Raheb, Akrivi Katifori, Aristotelis Kasomoulis, Marianna Rezkalla, George Tsampounaris	Athena
Oshri Even-Zohar	Motek
Vladimir Viro	Peachnote
Stefano Piana	Unige

List of reviewers

Name	**Affiliation**
Vladimir Viro	Peachnote
Antonella Trezzani	Lynkeus
Anna Rizzo	Lynkeus
Edwin Morley-Fletcher	Lynkeus

1 Executive Summary

This Deliverable is based on the outcomes of task *T4.2.2 Generic middleware architecture design and implementation* and *T4.2.3 Component specification and implementation for application scenarios*.

Within WhoLoDancE, several tools have been developed to fulfil the project requirements. Since WhoLoDancE considers several application scenarios, the use of different programming languages, development environments and technologies is mandatory. Nevertheless, in the development process an integration policy has been adopted in order to build a unique integrated WhoLoDancE framework.

The framework is designed to be scalable and platform independent. For this reason, a layered architecture is considered and tools that compose the framework are designed to be modular and reusable in various applications.

In order for all the components to be able to interact with each other, and in order to be able to interact with applications and engines outside the framework (VR visors, etc.) effective interfaces and protocols for data exchange are needed.

Table of Contents

2 The global scheme

In figure 1, the global schema of the WhoLoDancE framework is presented.

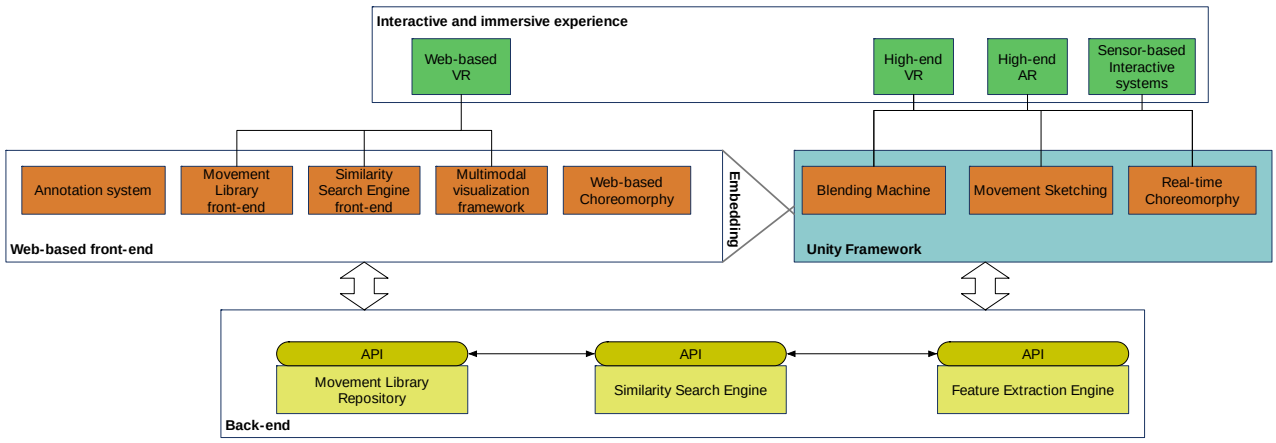

Figure 1 Global Scheme of the WhoLoDancE Framework

The WhoLoDancE project addresses different learning scenarios and dance practices, and this leads to several possible application scenarios. Moreover, the project has the goal to be easily expandable to scenarios that are not considered here. For this reason, the application framework we developed within the project is thought to cover different state-of-the-art technologies, easily re-usable in different contexts and expandable.

With this in mind, the framework is designed in three layers. The first is the **back-end** and represents the foundation layers. It implements a number of services like the Storage Layer, the Feature Extraction Engine and the Similarity Search Engine that are the base for most of the WhoLoDancE applications.

In order to be flexible and expandable, tools in the back-end layer need to be cross-platform and easily accessible from all the applications in the layers above and located in different regions, since the project Consortium is composed by different partners spread across Europe. For these reasons, tools are provided over as Internet services. Each service runs in a different server. With this architecture, the back-end can be easily expanded by adding new servers running new services. A set of efficient APIs are developed to access each service.

The second layer is the **front-end**. The front-end includes all the applications and tools that will be available to users. They can be an interface through which users can access back-end services, like the Movement Library front-end, or applications used in a specific learning scenario like the Blending Machine.

Due to the number of different scenarios to cover, we implemented the front-end considering two main technologies: Web-based and Unity-based technologies. The first is appropriate for distributed tools that do not need to be installed on a computer, they need to run on several platforms and they do not need to provide real-time responses. Whereas the second is mainly considered for real-time, interactive and immersive applications.

The various components of the front-end are developed to be modular and reusable. As an example, the Multimodal Visualization Framework is used in most of the implemented web-based tools.

Web-based and Unity based-frameworks can interact via Internet through back-end services or through direct Internet calls. However, as we will describe, in some cases an embedding procedure is needed (see section 5.2.2).

Since the project concerns real-time interactive applications, at the top level there is the **interactive and immersive layer**. The layer includes all the engines suitable for interactive and immersive experience like VR and AR engines, as well as sensor-based interactive systems. Front-end applications implement methodologies to interface with the upper layer that are specific for each engine.

In order for all the tools to interact within the framework we designed effective interfaces and protocols for **data exchange**.

All the components are described in the next sections in more detail.

3 Data Exchange

Due to the numerous tools developed for the project, the need to choose or design a common file format for data exchange arose. WhoLoDancE concerns the use of multimodal signals captured from dance performances including video, audio (music or environmental noise) and motion capture recordings. The dance experts in WhoLoDancE then *annotated* these recordings to provide manual descriptions of performances. In parallel, we also designed algorithms to extract from recording signals those properties, namely *features*, which provide an automatic description of performances.

In this Section, we describe file formats we used to describe recording signals, annotation data and features. We mostly relied on the JSON (JavaScript Object Notation) text-based file format. JSON is language-independent, hence an ideal data-interchange language, and is a standard application-level protocol for Internet-based application. The use of JSON was particularly suitable for the applications developed using web technologies (see 5.1 Web-based framework). JSON is built on two kinds of structures: a collection of name-value pairs and an ordered list of values. The combination and nesting of these two kinds allows to realize universal data structures with JSON. While text-based file formats are naturally more space-demanding than binary formats, we adopt some common compressing approaches to minimize the demand of Internet bandwidth when transmitting files.

In the rest of this Section, we will explain the data structure of files, and specify when such structure is exported using JSON. For the sake of brevity, we will avoid a technical explanation of the actual implementation of the structure.

3.1 Recording signals

The recording of dance performance sessions (see ***D1.5 Data Acquisition Plan*** and ***D2.3 Outcome of the Capture Process***) produced a high number of multimodal signals, including high-end and low-end video recordings, audio recordings of environmental sounds, performer's breath, music tracks used for accompaniment and motion capture signals. The various signals may not be synchronized (e.g., the video recording started 3 seconds before the motion), so there is also the need of creating and exchanging data regarding the synchronization (see ***D3.6 First Report on Software Platform and Libraries***). Many multimodal signals ultimately refer to the same dance performance; we therefore need a unified data format for saving the performances metadata.

3.1.1 Audio and video data

Audio and video data are widely employed and then use stable and shared file formats. For video signals, we used AVI and MPEG-4 video files that are supported by the HTML5 standard for web technologies. For audio signals, we used lossless WAV files and lossy MP3, which are both widely employed in literature and fully supported in HTML5.

3.1.2 Motion capture data

The motion capture (MoCap) system employs a set of 3D tracking markers that are placed on the dancer's suit, resulting in a *cloud of points* representation of the performance, which we store in a standard C3D file. Cloud-of-point representation is affected by noise from occlusion, mixing of markers, and slight change of muscular tension. For this reason, we converted them into a *skeletal* representation, which cleans the data and exposes the kinematic chain of movement by applying inverse kinematics techniques.

For exchange of skeletal data among project partners, we used the standard FBX file format, owned by Autodesk. On the one hand, FBX files are widely used for 3D model and animations, due to their compactness when expressed in binary format. On the other hand, the FBX structure requires computational effort to be processed; for this reason, we developed a Python script to convert FBX files to a simpler and explicit data structure that is best suitable for web-based applications, and we exported the converted structure into JSON files. The MoCap structure is shown in the following table.

Variable	Type	Description
Name	String	Name or identifier of the recording
filename	String	Name of the file from which the MoCap has been extracted
Np	Integer	Number of limbs in the MoCap representation
Nf	Integer	Number of frames in the MoCap animation
samplerate	Integer	Number of frames to be executed in one second (fps)
properties	Name-pairs	Properties as defined in the FBX file
labels	String [Np]	Name of each limb
conn	Int [][2]	Connections between limbs, following the kinematic chain, e.g., the LeftShoulder is connected with the LeftElbow
dancers	Name-Int[] pairs	For each dancer in the performance, which limb refer to him/her
GlobalPos	Float [Nf, Np, 3]	Global Position of each limb, in each frame, in the 3D world
LocalEuler	Float [Nf, Np, 3]	Euler Angles of each limb, in each frame, with respect to its parent connection
GlobalEuler	Float [Nf, Np, 3]	Euler Angles of each limb, in each frame, with respect to the 3D world

The MoCap JSON file represents an offline-processed, ready-to-use and web-friendly version of the FBX mocap files. The Position and Angles are expressed following the Unity reference system for 3D world.

3.1.3 Synchronization data

Multimodal signals from the same performance need to be synchronized with each other to guarantee simultaneous visualization, as in the WhoLoDancE Movement Library (see Section ***5.1.1 Web-based applications – an overview***).

In order to express the value of synchronization, we assume the existence of a time reference t_0 and express the starting point t_x of the generic recording *x* as an offset *sync[x]* with respect to such t_0, i.e., *sync[x]*= t_x-t_0. We have *sync[x]>0* if t_x>t_0 and *sync[x]<0* if t_x<t_0. We represent this strategy in 2.

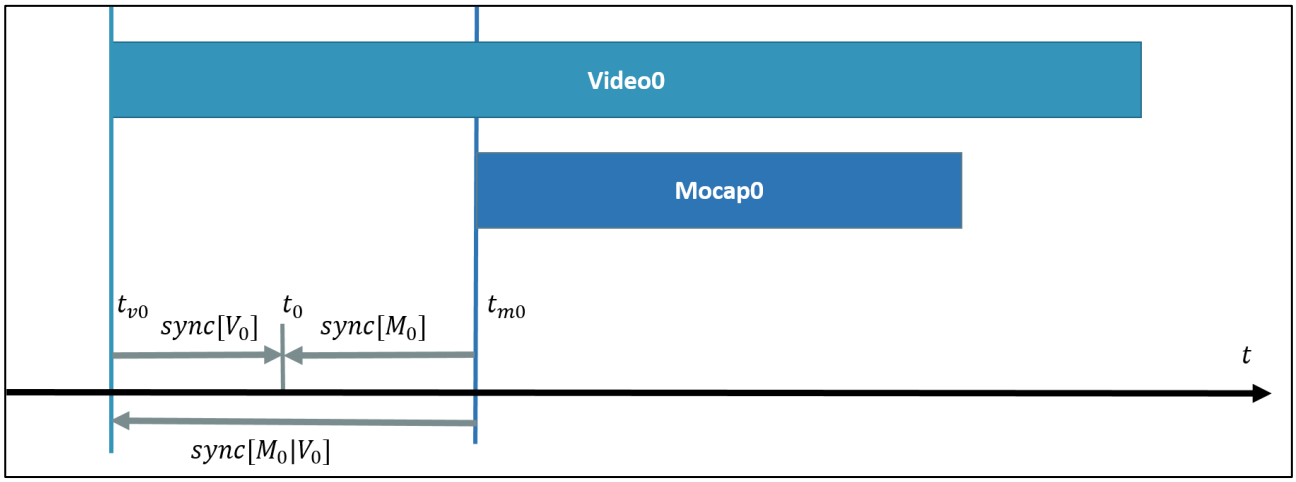

Figure 2. A representation of the synchronization strategy

To compute the synchronization between two recordings, for instance **Mocap0** and **Video0** (Figure 1), we compute the offset of **Mocap0** with respect to **Video0** (i.e., *sync[M0|V0]*) as:

$$sync[M0|V0] = sync[Mocap0] - sync[Video0] = (t_{M0} - t_0) - (t_{V0} - t_0) = t_{M0} - t_{V0},$$

and vice versa for $sync[M0|V0]$.

We set $t_0 = \min_{x \in Recordings} t_x$, in order to have $sync[x] \geq 0$ for all the recordings. The key-value pairs recording-synchronization are then stored in the metadata regarding the corresponding session, as detailed in the next subsection.

3.1.4 Metadata

The metadata concerning each performance are stored in the CKAN database described in ***D5.1 Data modeling, Data Integration and Data Management Plan***. The database employs the standard CKAN server to provide stability and maintenance. CKAN provides a web-API for database querying and retrieval, exporting results in standard the JSON format.

We use the following structure to describe dance performances. While the structure was designed for the project, it is flexible and open for external contributions.

Variable	Type	Description
name	String	Name of the performance
id	String	Identifier of the performance as assigned by CKAN

url	String	URL in the WhoLoDancE Movement Library where the performance can be watched
performer	List of String(s)	Name of the performer(s)
Num_performers	Integer	Number of performer(s)
Dance Genre	String	Genre of the performance
Company	String	Performers' company
Capture Date	Date	Date of capture of the performance
Capture Venue	String	Place (city) where the performance was recorded
Resources	List of recordings	Recordings that have been captured during the performance (video, audio, MoCaps, etc.), with information on synchronization
Num_resources	Int	Amount of captured recordings

3.2 Annotations data

The WhoLoDancE Movement Library (WML) allows users to simultaneously watch the MoCap and video recordings of a performance, and to add annotations to it. These annotations may refer to movement qualities (graded between 0 or 10), actions that appear during performance, other elements define in the ontology described in ***D3.1 Report on Semantic Representation Models***, or even free-text labels (*tags*). Each annotation may refer to the whole performance or to a narrow fragment and apply to all performers or focus on a specific limb.

These annotations are useful for performance retrieval in the WML itself (e.g., *all performances with a jump*), and to train models for the extraction of data-driven features (see next section and ***D3.5 Report on Data-driven and Model-driven Analysis Methodologies***). We designed a simple structure for annotations, that are stored in the CKAN repository and exportable in JSON format.

Variable	Type	Description
name	String	Name of the annotated performance
id	String	Identifier of the annotated performance as assigned by CKAN
user	String	Annotating user's name
Motion Capture	Boolean	Whether the annotation refers to the MoCap recording
Video	Boolean	Whether the annotation refers to the Video recording
Dance Genre	String	Genre of the performance
Category	String	Type of annotation (Movement Quality, Action, etc.)
Label	String	Name of action /movement quality annotated

Value	Integer	If graded annotation: grade value (0-10), otherwise is -1
Starting Time	Float	Starting time of the reference segment for annotation
Ending Time	Float	Ending time of the reference segment for annotation

3.3 Features

We can extract many features from multimodal recordings, and techniques vary depending on several factors:

- the recording signals -and the underlying information- vary from 1-dimensional audio times-series to 3-dimensional video and MoCap signals; different signals lead to different extraction techniques and extracted properties; in the project, we focused on MoCap recordings for the richness of information;
- we can extract different levels of features, from physical (low-level) features, to semantic (high-level) features; the higher the level of the features, the longer the window of observation needed for computation, from frame-level to tens of seconds, or even not fixed *a priori*, hence the dimensionality may vary (see ***D3.5 Report on Data-driven and Model-driven Analysis Methodologies***);
- we can aggregate information over different dimensions: features may be computed for each joint in the MoCap, for each direction, or aggregated over axes, to have a global descriptor for each joint, or aggregated over joints, to have a descriptor for each main limb (e.g., torso, left arm) or even for the whole body (especially for higher-level features).

Combining the aforementioned factors, we may have 4-d features as well as scalar values for the description of a performance. Due to the high variety of features we can extract, we did not design a fixed structure for them and let it change freely depending on the feature. We export extracted features in JSON and CSV text-based files.

4 The back-end servers

4.1 The Storage Layer (Movement Library Repository)

The WhoLoDancE storage layer is part of the data management system and is described in detail in *Deliverable 5.1 Data modelling, data integration and data management plan report* (Section 4). It is a file-based object store for depositing binary data objects. The repository is implemented over a redundant store with one delayed replica and is accessible via a number of standard protocols such as FTP and HTTP, while special protocols are also available depending on the data type (e.g., streams for media objects). Items in the repository obtain URLs that can be disseminated via standard web means, yet access may be provided only with granted credentials.

The storage layer includes also a relational database management system (PostgreSQL) for managing dataset metadata, behind the CKAN repository and pilot-specific services.

4.2 The similarity search engine

The search engine is described in detail in *Deliverable 4.1 Report on Data Integration, Algorithm and System Analysis, and Framework Description*, in particular the section "Service Implementation" which describes the system architecture.

The search and similarity platform supports automated submission of recordings and time series data by authorized users. It supports flexible querying scenarios, in which users can specify the features to be used for searching and their weights by themselves. The functionality is exposed via a REST API that consumes and produces data in JSON format. The API is documented and easily consumable in a variety of environments. A simple authentication scheme is employed. Multiple deployment instances of the engine are available to technical partners for experimenting with the API, the low-, mid- and high-level features that they can generate and submit to the search engine on their own, and the custom weighted templates without interfering with other partners or the production deployment. The deployments are easily upgradeable to support short development and deployment cycles.

The Search and Similarity Engine is implemented as a stand-alone Java application that can be interacted with over a REST API. The API implements a Swagger API definition, from which client implementation in multiple languages can be automatically generated.

For real-time applications WebSocket connections can be used for streaming motion and HLF data to the search engine and receiving a stream of pointers to the search results. WebSocket connections are supported for both Unity and Web-based clients.

4.2.1 Self-hosted documentation

The application hosts its own documentation, available at the root endpoint of the API, e.g. http://search.WhoLoDancE.peachnote.com. The documentation describes all available REST endpoints and the data structures consumed and returned by the API.

4.2.2 Access control

The access to the API is secured with HTTP Basic Authentication mechanism, which requires the user to provide a username and password before accessing the service. The authentication is transmitted using an HTTP header of any request sent to the service.

The API supports the Cross-Origin Resource Sharing (CORS) mechanism that makes it easy to consume the API from within any web page that wishes to make use of it.

4.2.3 Data management

The search engine ingests submitted time series and meta data and persists them in a PostgreSQL database and on the file system. The configuration of the database access and the file system storage is provided using a configuration file.

4.2.4 Deployment

The application is automatically updated on every change in its source code repository resulting in a successful build.

4.3 The Feature extraction engine

In view of WhoLoDancE's purpose of giving accessibility to the various tools over internet, we are developing a service that will provide online accessibility to movement feature extraction from recordings. The service will embed the feature extraction modules developed so far in the project (See D3.4 and D3.5 for details) and provide an API to access the functionalities from remote clients.

The first version of the service will require FBX files to be analysed, and the analysis will be performed asynchronously. An extension of the service to add compatibility to live streams and different file formats is under investigation, the API of the service is detailed in Section 4.3.2.

4.3.1 Feature Extraction Engine functionalities

The Feature Extraction Service will initially allow two types of usage: direct analysis of a capture sequence (i), fetching and analysis of a sequence from the WhoLoDancE repository (ii). In the first case the user will provide an existing recording that will be analysed while for the latter one the system will fetch an existing recording from the WhoLoDancE repository and analyse it. In both cases, the user will be able to download the features from the Feature Extraction Service.

4.3.2 Feature Extraction Engine web API

This section describes the API that will be available on the feature extraction engine that is currently under development, the service will provide a REST API that will enable feature extraction from FBX recordings, an extension to CSV, c3d and JSON recordings is under investigation. The current version of the API is detailed it Table 1 and Table 2.

Table 1: API methods related to feature extraction modules

Features				
Method	Address	Description	Parameters	Returns
GET	/features	Lists available features on the server	Limit: the maximum number of retrieved entries	List of retrieved features as a JSON array
GET	/features/{recordingId}	Lists extracted features for recording with given Id	RecordingId: the id of the desired recording	List of extracted features for the given recording as a JSON array
GET	/features/{recordingId}/{featureId}	Download a file containing	RecordingId: the id of the desired recording featureId: the id of the	A CSV file containing the extracted feature

			desired feature	

Table 2: API methods related to feature analysis

Analysis				
Method	Address	Description	Parameters	Returns
POST	/analysis/analyse	Analyses a recording	Expects A Json containing the link to the file of the recording	An Id given to the request
POST	/analysis/analyse_from_repository	Analyses a recording fetching it from the WhoLoDancE repository	RecordingId: the id of the desired recording	An Id given to the request
GET	/analysis/status	List the status of all the requests enquired so far	Limit: the maximum number of retrieved entries	
GET	/analysis/status/{requestId}	Gives the status of an analysis request	requestId: the Id of the desired request	The status of the desired request

4.3.3 Extensions of the Feature Extraction Engine

The first version of the Feature Extraction Engine will provide asynchronous data analysis; one of the extensions that are under investigation is real-time data streaming-analysis. In this scenario, a client would connect to the Analysis Service, stream coordinates and receive analysis results in real-time (with reasonable delay). This process would require the client to provide a stream of data to the service and receive the stream of results from it, thus requiring a pretty wide available upstream/downstream bandwidth and a synchronization mechanism (i.e., timestamping) between client and server.

5 The front-end applications

Front-end applications are the interface with which users can access services and data provided or stored in the back-end service, in a typical client-based architecture. There are two main approaches for the front-end

development: web-based and stand-alone. Both approaches offer pros and cons and, for this reason, in the project we decided to follow both. In the following, we present the application framework and integration among applications.

5.1 Web-based framework

Web-based applications are very common nowadays, due to numerous pros. Firstly, they do not require a dedicated installation, since they are accessible by using a modern web browser, and are therefore naturally cross-platform. Moreover, they offer more control to the owner of the application, including authentication or geographical limitations. Lastly, the deployment of new versions is instantaneous, as soon as the new version is uploaded to the server, overcoming issues related to backward compatibility. In the rest of the Section, we describe each implemented web-based application, highlighting their interaction with back-ends servers and other applications. For a detailed description of the applications, please refer to *D5.3 Integration and interoperability with external services, systems and applications report*.

5.1.1 Web-based applications – an overview

For each application we provide a brief overview of the functionalities.

Visualization tool

First, we implemented the visualization tool to visualize MoCap performances as 3D movements. The tool contains a 3D scene (using WebGL API) that can be manipulated to rotate the scene, zoom in/out or move the scene. This allows users to watch a MoCap dance performance from every angle they desire. The tool loads MoCap JSON files as described above, and it shows it using a simple stick-man visualization. In case two performers are involved, the tool renders them with different colours to ease discriminability between the two.

The execution of the performance (starting, pausing, resuming, seek at a given moment, etc.) is controlled by Javascript functions accessible from the outside. For instance, one can design a progress bar to show the execution instant, insert HTML buttons for playing/pausing the performance, or using voice command to control it.

The visualization tool is in fact the very first web-based module designed for the project, complying modularity and reusability requirements. It is indeed included in the Annotation Tool, in the Similarity Search Engine front end and in the WhoLoDancE Movement Library front end (see below), and its code is the core of the Web-Based VR tool. The Choreomorphy Web-based version can be seen as the natural evolution of the tool for more artistic purposes.

Annotation tool

Annotations on the recordings describe and analyse the dancer's motion. The "Annotation Table" allow users to quickly add, edit or delete annotations inline through the table structure. The steps of adding and editing annotations takes place gradually. The system guides the user step by step, including several useful mechanisms, such as searching with keywords, regulating the number of annotations that will emerge in each page, as well as sorting the columns of the table. Last but not least, undo and redo methods have been implemented.

Regarding the need of visualizing annotations, a timeline structure has been created, under the player's box.

Users will be informed of the existing annotations, both from the annotation table and the timeline. The timeline structure is synchronised with the media player. During the playback of the recording, a vertical, red line moves through the timeline segments, in order to present the annotations that correspond in each time spot. The timeline is also combined with many useful functionalities. Filtering, zoom in and out, slide right or left, move in specific timestamp are some of the provided functions. However, the timeline structure could also be used as an effective means for managing annotations. By hovering over a timeline annotation, options for editing and deleting will appear, while a button is used to add new annotations instantly on the timeline.

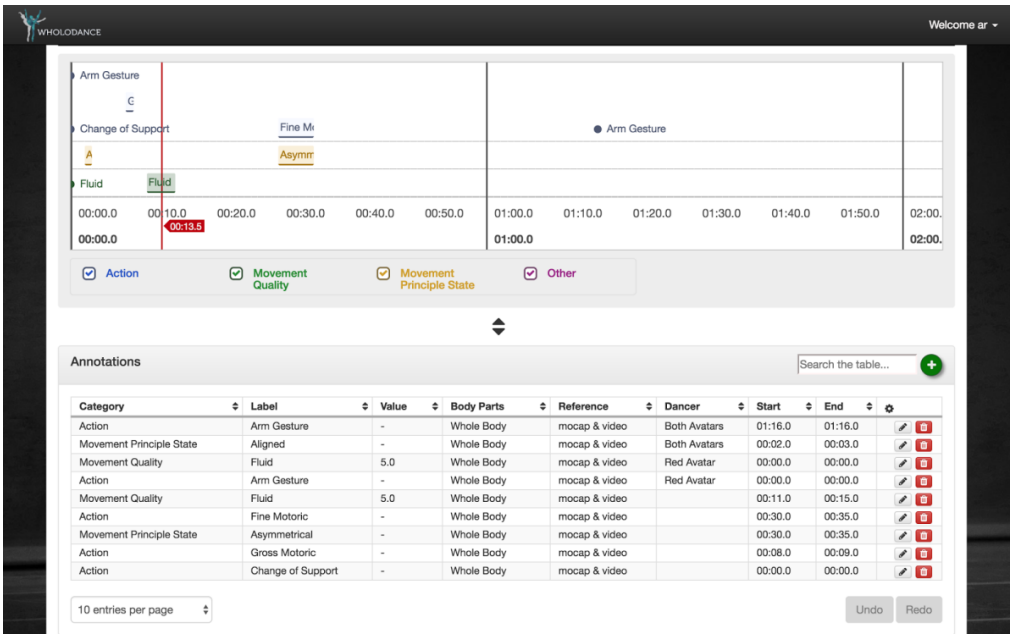

Figure 3 Timeline and table of Annotations

Similarity Search Engine front end

The Similarity Search front-end is a web-based prototype developed to test and showcase the **Similarity Search Engine back end**. In order to use it, we had to extract features from the MoCap recordings using the **Feature Extraction Engine** and upload them to the **Similarity Search Engine** for offline optimization.
The Similarity Search shows performances metadata and corresponding MoCap recording accessing the **CKAN repository**. By interacting with the **Similarity Search Engine**, and with the **CKAN repository**, the front-end returns and show the most similar fragments of performance. The query fragment and the results are played simultaneously and shown side-by-side, to help comparison.
A more detailed description of the front-end is provided in *D4.2 Similarity Search Framework and Components*.

WhoLoDancE data management system

The main objectives of the WhoLoDancE data management system interface is to organize the files in the FTP server by grouping them into recordings and adding metadata information. It also provides a useful API in order to access the stored data and metadata from other applications. CKAN Action API exposes all of CKAN core features to API clients. All CKAN website core functionality can be used by external code that calls the CKAN API. Moreover, it provides a vast number of tools and libraries for the CKAN API, such as Python, Java and Javascript. The WhoLoDancE data management is based on the CKAN metadata which is a repository tailor-made with regards to configuration and plugins, to fit WhoLoDancE project datasets and metadata

servicing needs. It offers full web interface for managing and accessing metadata and a rich set of tools and libraries for consuming/exploring projects datasets.

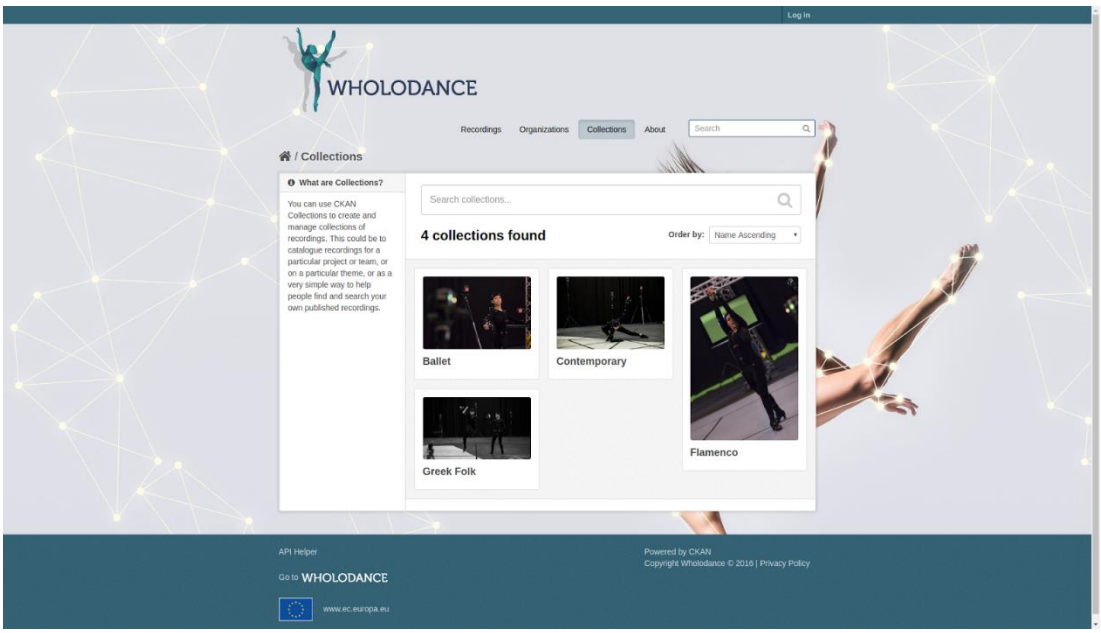

Figure 4. The Data management system front end tool

The WhoLoDancE Movement Library

The main objective of the WhoLoDancE Movement Library (WML) application is to provide access to the WhoLoDancE repository, through a usable interface with browsing, searching, visualization and annotation functionalities for the multimodal recordings.

More specifically, the user can browse the recordings by dance genre, and search by using keywords that are included as metadata of the recordings. A special player has been developed, to allow the synchronised playback of a video, as well as its corresponding motion capture file. Moreover, not only do users have the opportunity to view the recordings but also to annotate them. Finally, a timeline that operates as viewer for the annotations has been developed.

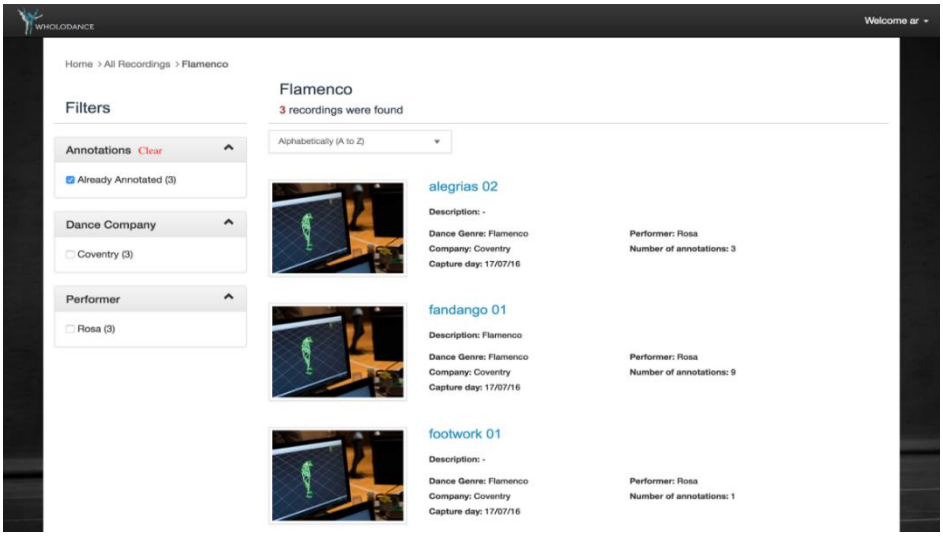

Figure 5. Search Results page

The WML tool is a web-based application, compatible with every OS system, built according to the principles of the MVC architecture. Model-View-Controller (MVC) software design pattern, works by dividing an application into three interconnected parts, in order to separate internal representations of information from the ways that information is presented to and accepted from the user.

Spring Web MVC framework was selected. Like many other web frameworks, it is designed around the front controller pattern where a central Servlet, the DispatcherServlet, provides a shared algorithm for request processing while actual work is performed by configurable, delegate components. This model is flexible, scalable and supports diverse workflows.

The view component has been developed by utilizing the JSP script based templating system. JavaServer Pages (JSP) technology has the ability to create dynamically generated web pages. Web pages development is based on the HTML mark-up language and CSS determines how those HTML elements should be displayed.

Bootstrap has been used in order to design and implement the Wholodance Movement Library application. The current open source toolkit contains HTML and CSS based design templates, as well as optional JavaScript extensions.

AJAX, which is a set of Web development techniques on the client side that creates asynchronous Web applications, has been used. With Ajax, WML application can send and retrieve data from the server asynchronously (in the background) without interfering with the display and behaviour of the current page.

Through the WML application, several different functionalities, allow users to interact with the tool. Those functionalities were implemented with the scripting language, Javascript. Its use is primarily for DOM Manipulation, AJAX Calls and Validation. Features such as the player functions, synchronisation between the video and the mocap, as well as services supported by the annotations table, have been all developed with Javascript.

Web-based VR

We implemented a web-based VR application using modern Javascript WebVR APIs and dedicated libraries. The application works with low-end devices (such as Google Cardboard) as well as high-end compatible devices (including HTC Vive). Due to the need to provide an enjoyable experience to users regardless of their device, our web-based VR are designed to use a modest amount of computational and graphical resources.

In the current implementation of web-based VR, users load a performance from the **CKAN repository** and watch it with a VR device in an immersive scene. Users can move through the VR space, watching the performance from every angle and view. Moreover, since the web-based VR is able to show any performance which complies with the aforementioned JSON MoCap structure, it is also possible to load a performance created with the **blending engine** (see next section). This aspect makes the web-based VR a useful application for choreographers to easily and immediately share their creations over the web. The current prototype offers a set of pre-defined environments - an empty space, a virtual rehearsal space, a Laban cube – among which the user can choose.

Choreomorphy web-based version

The Choreomorphy web-based tool allows the visualization of a recording in WML through the use of different avatars. The WebGL build option allows Unity to publish content as JavaScript programs which use

HTML5 technologies and the WebGL rendering API to run Unity content in a web browser. Choreomorphy WebGL edition is supported by all major desktop browsers to some degree. It operates as a beautified viewport of the WML 3D visualization. When the user has selected a choreography, Choreomorphy retrieves with Ajax the motion capture data from the remote repository and then visualizes it on the avatars.

5.2 Unity-based framework

5.2.1 Unity-based applications – an overview

For each application we provide a brief overview of the functionalities.

Real-time Choreomorphy

Choreomorphy is a whole-body interaction interface that allows a user to visualize their movement in real time using motion capture technologies. The interface allows a user to change avatars and different visualizations in real time, in order to focus on specific aspects of their movement such as traces, trails, and volumetric space, and improvise while seeing themselves as different avatars and shapes and interact with virtual objects. Choreomorphy has been implemented in Unity®, a cross-platform game engine developed by Unity Technologies, which is primarily used to develop both three-dimensional and two-dimensional video games and simulations for computers. Mono, the open source development platform based on the .NET Framework is used for the development of the app. Mono .NET implementation is based on the ECMA standards for C# and the Common Language Infrastructure. The dynamic shaders are developed using Cg, that is a modified version of Microsoft's High-Level Shading Language.

Choreomorphy is created in such a way as to be able to work with any motion capture equipment that includes a Unity Plugin. In our case, we used Synertial IGS-C420 motion capture equipment, which is an inertial motion capture suit with 42 IMU sensors.

The pipeline for a successful setup and motion capture session with Choreomorphy is the following:
1. **Router:** establish a local network that will enable communication between computer and motion capture suit via its wireless HUB.
2. **PC:** launch Animate.exe, which is a Synertial software that fetches the motion capture data that is streamed by the suit ‘s HUB and then broadcasts it to Choreomorphy app.
3. **PC:** in Animate software, enable the broadcast option to broadcast the motion capture data anywhere. In our case to Choreomorphy.
4. **Motion capture suit:** turn on HUB to connect with Animate software, to stream the motion capture data to it.
5. **PC:** calibrate the suit. (~3 seconds)
6. ready to start the session.

The avatar library consists of 3D models that are available at the Unity Asset Store, some of them have been designed by 3D Artist Maya Lara and Motek. More details on the tool can be found in deliverable D5.3 Integration and interoperability with external services, systems and applications report.

Movement sketching and feature analysis integration in the unity environment (UNIGE)

Modules for feature extraction and the movement sketching application were developed in the EyesWeb XMI[1] environment, to provide a single environment to end users, especially in real-time use cases, the tools will be integrated in Unity. The integration is implemented using a communication mechanism between the

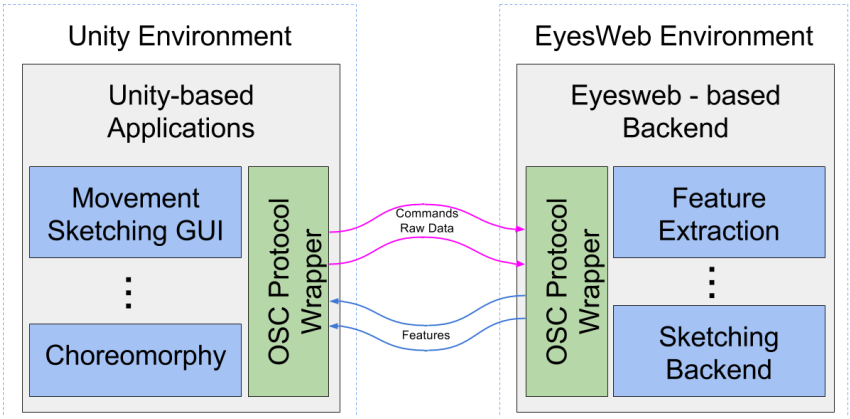

Figure 6. Unity Integration schema and communication protocol between Unity-based and Eyesweb-based applications

Unity Environment and the EyesWeb-based implementation; using OSC protocol, in particular, commands and raw MoCap (positions, rotations) data streams will be sent to the Analysis modules by the Unity application, and results of the analysis will be received from the EyesWeb-based Application. The current Graphical user interface will be ported and integrated into Unity: **Errore. L'origine riferimento non è stata trovata.** shows a schematic view of the communication between Unity and Eyesweb-based applications.

Blending machine

The Blending engine is a standalone PC desktop application that enables blending of every motion capture sequence from the repository to every other sequence, while allowing separation and different blend types for each body part. It also enables the creation of long choreographic pieces by assembly of sequences. The blending machine will be integrated into Unity as an FBX stream source. Inside Unity, the users may request the direct output in runtime from the blending engine, or just load into the scene blended sequence.

5.2.2 Embedding of web-based tools into the Unity-based framework

Web-based tools can easily interact with the Unity-bases tools by using the throughput of data via Web socket, or Tcpip or Udp protocols to input and output data stream. Nevertheless, in some scenarios it will be useful to embed web-based applications into the Unity-based framework, preserving the advantages of the web technology and without the need to re-develop the application into the Unity environment. Operations that stay in the 2d domain can be visualized as a full Web browser inside a window in the AR or VR view. This is the case, for instance, when a dancer dances with an avatar that should be loaded by from the Movement Library.

[1] http://www.infomus.org/eyesweb_eng.php