D3.3 Software Prototype with Sensor Fusion API Specification and Usage Description

Editor:
Roland Klemke (OUNL)

Wearable Experience for Knowledge Intensive Training
Project No 687669

Revision History

Version	Date	Contributor(s)	Modification
0.1	06-10-2016	Roland Klemke	Initial Outline
0.2	10-10-2016	Roland Klemke, Fridolin Wild, Daniele Di Mitri	Methodology and refined structure
0.3	19-10-2016	Roland Klemke, Daniele Di Mitri, Bibeg Limbu, Jan Schneider	Design requirements and existing prototype
0.4	10-11-2016	Puneet Sharma, Bibeg Limbu, Fridolin Wild, Roland Klemke, Daniele Di Mitri, Tre Azam, Jan Schneider	API Specification, Recommendations, and Conclusions
0.5	18-11-2016	Kaj Helin, Istvan Koren	Review
0.6	25-11-2016	Roland Klemke, Daniele Di Mitri, Jan Schneider, Bibeg Limbu	Final version
1.0	30-11-2016	Cinzia Rubattino	Final edits

Software Prototype with Sensor Fusion API Specification and Usage Description

WP 3 | D3.3

Editors:

Roland Klemke (OUNL)

Authors:

Daniele Di Mitri (OUNL)
Bibeg Limbu (OUNL)
Jan Schneider (OUNL)
Puneet Sharma (UiT)
Fridolin Wild (OBU)
Tre Azam (MP)

Reviewers:

István Koren (RWTH)
Kaj Helin (VTT)

Deliverable number	D3.3
Dissemination level	Public
Version	1.0
Status	Final
Date	30.11.2016
Due date	M12

Table of Contents

Executive summary

This deliverable reports on the components of the first functional prototype of the WEKIT sensor fusion API and the experiences made with it. The development of these components is based on previous work on the WEKIT framework and methodology, requirements and scenarios, as well as technological selections and limitations. The deliverable specifies the key interfaces between the software component and the hardware, the backend infrastructure, and the front-end application modules. Furthermore, it contains usage recommendations. This deliverable represents the first of two iterations. The second iteration is due in month 27.

1. Introduction

As outlined in the WEKIT project description, the objective of this deliverable (D3.3) is to describe the software prototype and specify the sensor fusion API. It involves reviewing existing deliverables and transform their outcomes into formal specifications:

- D1.3 WEKIT Framework and Training Methodology
- D1.4 Requirements for Scenarios and Technological platform
- D3.1 Requirement Analysis and Sensor Specifications

This deliverable is delivered together with further technology-oriented deliverables:

- D2.1 Functional and Modular Architecture: Requirements and Specification
- D2.2 Learning Experience Content Model Draft
- D3.2 Hardware Prototype with Component Specification and Usage Description

The results of this deliverable will be used to further develop the experience capturing API (iteration of this deliverable), to provide further input to hardware specification and development activities (D3.2) and to guide the development of application modules, which are based on the API (WP2, WP4).

The rest of this deliverable is organised as follows. In Section 2, we outline the methodology used. In Section 3, we analyse the starting points for this deliverable: input from previous deliverables, coordination with accompanying deliverables, and input from related research activities. Section 4 contains the core specification of the experience capturing API split into three parts: (1) API for interaction with the hardware, (2) API offering high level functionality to application modules, (3) Backend considerations for the API. In Section 5, we condense a set of recommendations for the usage of the experience capturing API.

2. Methodology

In order to tackle the challenges WEKIT faces in technology selection and development, data storage and analysis, user interface design, and pedagogic instruments, the project relies on

several pillars in its technology development activities, which are represented by parallel deliverables to be delivered along this one:

- Deliverable D2.1 (functional and modular architecture) specifies the overall bird's eye view on the project's infrastructure, components, and their connections.
- Deliverable D2.2 (learning experience content model draft) describes how data formats and protocols can be put in place to allow for storage, communication, and retrieval of captured experiences.
- Deliverable D3.2 (hardware prototype with component specification) specifies the hardware components to be used and connected as the wearable solution for WEKIT.

This deliverable is connected to these parallel deliverables, in that it specifies a software component (Experience Capturing API, short XCAPI), which connects the hardware components (D3.2, sensors and their integration and abstraction, wearable devices) to the backend storage (D2.2 & D2.1). Additionally, this deliverables offers methods for the development of high-level end-user applications.

To minimize the impact of interdependent work aspects, the first version of the technical prototype has been developed in an agile, rapid prototyping manner. In several iterations, sensor integration component, storage formats, user interface prototype, capturing and re-enactment functionality, as well as editing functionality have been developed and tested.

These development cycles lead to experience with respect to:

- sensor communication
- data amounts and formats
- interaction requirements and limitations
- functionality requirements and constraints

Based on these experiences, the following development cycles will work towards improved, stabilized, usable and functionally more complete versions of the XCAPI and its connected components. Figure 1 shows an overview over this structure.

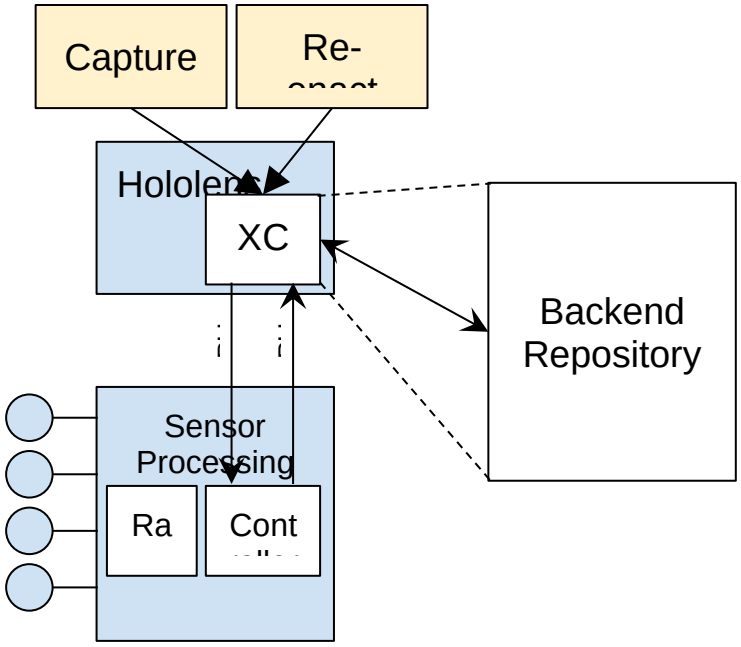

Figure 1. XCAPI and its Connections to Frontend, Backend, and Hardware

3. Design Requirements

3.1 Framework and Methodology (WP1: D1.3 & D1.4)

The WEKIT framework at its core collects attributes of an expert and methods to train these to novices using AR & WT. The general attributes of the expert, termed "Expertise" here on, are entities that are demonstrated by experts in different domains. Expertise can be regarded as higher form of skills in experts which is domain specific and need to be learnt through experience. The methods to train this expertise, termed "Transfer Mechanisms", are more than mere instructions to practice the skills (see Table 1). In addition to skills, the Transfer Mechanisms allow trainees to experience the experts performance by augmenting the skills with expertise. [*Refer to document "Framework Information Collector" from D1.3*]. The framework information collector document consist of a classification of the Transfer Mechanism based on focal areas of expertise. Of the 7 identified categories, Gaze Behaviour, Point of View, Audio & Motor Performance are relevant for the XCAPI and the context of this document. This is not to say we do not address the rest of the categories. They do not rely on captured data from expert and thus are outside the scope of XCAPI specifications. These are addressed in the "pre/post recording phase". This section elaborates on how we can provide an API to the UI designers, Instructional designers and front end software developers so that they meet our framework standards to achieve the best results with their application.

Table 1. Template for Transfer Mechanisms

Category Based on Resource of Expertise

● Classification of the Transfer Mechanism based on focal areas of expertise. It should be noted that while a task may have many forms of expertise involved, capturing them would require focus on different aspects of the expert.
General Descriptor ● How can the feature be described?
Requirements for Recording ● How is this mechanism enabled during recording? ● Which conditions need to be met to enable this feature? ● Which functionalities does the feature have to offer?
Requirements for Training ● How is this feature enabled by/for the learner? ● What does this feature do? ● Which conditions need to be met to allow this feature to be present? ● Which interaction means does the learner have?

It should be noted that each transfer mechanism represents an instructional approach in its own right, thus having a design and use case. Additionally, the requirement sections in the Transfer Mechanism list any possible types of requirements from technical to human aspects. Each Transfer Mechanism in the document has a "Requirement for Recording" and "Requirement for Training" section. It seems only logical to separate the APIs based on these categories even though there might be instances where some of them might overlap. Since the front end application will need to record the expert data and then create a training application as well, APIs need to be provided for both instances. Therefore APIs will be designed to meet these requirements stated by Transfer Mechanisms and the requirements envisioned from the elaboration of use case of each Transfer Mechanism.

The process of capturing expertise is done in the 3 steps of *pre-recording*, *recording* and *post recording*. As stated, the pre-recording phases consist of planning tasks such as task analysis and sensor setup. It involves the expert and the person performing the task analysis. Task analysis provides us with a structure of the complex task that is broken down into subtasks along with crucial explicit information that can assist in proper recording of the expert such as the granularity of breaking down the task. The expert records each subgoal individually by breaking each subgoal into a number of linear steps. During the recording, the expert will be allowed to perform the task fluidly with no interruptions of having to explain or divide the task into a number of steps. When the recording is done, the expert will be allowed to review his recording to further improve the instructions. During this phase, he may chose to annotate particular hardware, provide a verbal explanation, choose to keep the video data or delete it based on the type of data that was captured. In order to facilitate the phase of post recording, the sensor data must be provided in a simplified manner to the expert and allow any common modifications on the data that seems relevant. It also means that it must be recorded in a sensible manner. For example, the eye tracker could provide only the areas that were focused for longer duration and frequency rather than having to play the whole video through and manually deleting the noise. Therefore, each packet of data should be tagged with the ID that follows "goal/subgoal/step_number….". The packet should also handle empty calls for data of sensors that are not used in that particular step. For this, the same type of recording such as eye tracking

should share the same naming convention in all the packets. Each type of data should also be individually accessible.

During the training phase, the trainee will follow a natural sequence of activities unless explicitly needed in which case he may have the option to search for the required step from the task hierarchy provided by the expert. He/she should start the task in the same settings, environment and orientation. The picture of how the end user system will look like depends on the application developer. Table 2 summarizes the functional requirements followed by non-functional requirements but the APIs will satisfy the requirements for re-enactment for each Transfer Mechanism.

Table 2. Functional requirements

Functional Requirements	
Capture	Training
● Synchronized Fusion of sensor data ● Each sensor data will be stored in packets that correspond to the annotation (see below) based on experts task analysis. ● Each packet must be able to perform functions of their parent's sensor which might be required for the editing or re-enactment. ● Each packet must handle empty calls for data of sensors not available in that packet. **Gaze Behaviour** ● Must allow deletion of points of focus. ● Must be coupled with the video relevant to the captured data. **Point of View** ● Must allow video to be cut into clips. ● Must allow deletion of clips. ● Allow cropping of the video. ● Should allow taking pictures which might be a clue to understanding the complex step and tag it. **Audio** ● Allow clipping of audio. ● Allow tagging audio in dimensional space and time. **Motor Performance** ● Augmenting the performance of the expert. **Task Analysis**	**Gaze Behaviour** ● Only provide points of significance which has higher gaze duration and gaze rate. ● Allow each point to be tagged to a physical object location in the scanned 3D model of the room. **Point of View** ● Provide video control options such as speed **Audio** ● Provide audio control options. **Motor Performance** ● Define tolerable range based on expert movement to provide haptic feedback. ● Allow enriching the replay of expert 3D model with augmented instructions.

• Allow artefacts to be tagged. • Define artefacts' individual characteristics.	

3.2 Requirements Derived from Sensor Specifications (D3.1)

For the WEKIT project, we aim to use binocular augmented reality glasses. Taking into consideration all the factors (mentioned in the sensor specification deliverable D3.1), **Microsoft Hololens** (Hololens, 2016) with features including: environment capture, gesture tracking, mixed reality capture, Wi-Fi 802.11ac, and fully untethered holographic computing, is the best candidate for the project.

While the main design of the prototype will be based around Microsoft Hololens, to keep the prototype open for other augmented reality glasses and emerging technologies such as: Snap glasses or the ODG R-7 augmented reality glasses.

For EEG, the **MyndBand and Neurosky chipset** (MyndBand, 2016; Neurosky, 2016) are favoured for the WEKIT project due to the processed data, ease of use, simple setup, low cost and ability to provide research grade data on attention, relaxation levels, and eye blinks.

For eye tracking, we need a wearable eye tracking component that can work in conjunction with the augmented reality/smart glasses. However, there are no eye-tracking solutions available for Microsoft Hololens or ODG R-7 (ODG, 2016). To this end, we plan to use **Hololens' gaze**, which is not based on the eyes, but, on the position and orientation of the user's head. It acts as a laser pointer originating from the center of two eyes, and by intersecting with a spatial mapping mesh (provided by Hololens) gives a good estimate on the gaze direction of the user.

For capturing the point of view of the expert or trainee in the the Industrial scenarios, we aim to use the **point of view camera** associated with augmented reality glasses (e.g., Microsoft Hololens). In doing so, we use more sensors from the existing augmented reality glasses, thereby, reducing the complexity of the overall system design.

For capturing the voice of the expert or novice in industrial training scenarios pertaining to aviation, space and medical use cases, the **built-in microphone** of the augmented reality glasses can be used.

For detecting hand movements and gestures, camera based sensors such as: **Leap Motion** (Leap, 2016) and **Intel RealSense** (Intel, 2016), can be used. For sensor-based arm (and, to a limited degree also hand-gesture) tracking, the **Myo armband** (Myo, 2016) can be employed.

For detecting the posture, we plan to use **Lumo Lift** (Lumo, 2016) **or Alex posture tracker** (Alex, 2016) (sensor-based system).

Table 3 summarizes the selected sensors and the corresponding requirements.

Table 3. Selected Sensors and Requirements for Interaction, Capturing, and Re-enactment (see also table 1 in parallel deliverable D3.2)

Sensors	Technical specifications	Requirements for interaction	Requirements for capturing	Requirements for re-enactment
Augmented Reality Glasses (Microsoft Hololens)	See-through holographic lenses (waveguides, 2 HD 16:9 light engines, Automatic pupillary distance calibration. Holographic resolution: 2.3M total light points, holographic density: >2.5k radiants (light points per radian), 4 environment understanding cameras, inertial measuring unit, depth camera.	Track location of user in the environment, mapping the environment for optimal placement of virtual objects.	Track location of user in the environment, track objects in the environment.	View instructions, activity, videos, and virtual post-its, application, in the AR display.
Augmented Reality Glasses (ODG R-7)	Dual 720p Stereoscopic See-thru displays at up to 80fps, 80% See-through transmission, Magnetic Removable Photochromic Shields.	Same as Hololens (above).	Same as Hololens (above).	Same as Hololens (above).
Point of view camera (Microsoft Hololens)	1 2MP photo, high definition video camera	None	Start and stop video recording, take digital pictures, enable and disable point of view camera, capturing current point of view.	Capturing current point of view, enable and disable point of view camera.
Built-in microphone (Microsoft Hololens)	4 microphones	Voice as input for interaction	Start and stop the microphone, enable and disable microphone.	Start and stop the microphone, enable and disable microphone.
Gaze (Microsoft Hololens)	Gaze cursor	Estimate gaze direction, select objects in the environment, place virtual post-its.	Estimate gaze direction, select objects in the environment, place virtual post-its.	Estimate gaze direction, select objects in the environment, place virtual post-its.
MyndBand and Neurosky chipset	Raw-Brainwaves, EEG power spectrums (Alpha, Beta, etc.), Attention, Meditation, and other future metrics, detect poor contact, and whether the device is off the head.	None	Estimate attention, focus eye blinks, and other metrics, enable and disable EEG.	Estimate attention, focus eye blinks, and other metrics, enable and disable EEG.
Intel RealSense	Range: front-facing: 20-180 cm, rear-facing 50-500 cm. Depth camera	Use hand movements and gestures to interact with the	Recognize hand movements and gestures.	Recognize hand movements and gestures.

	with 640x480 resolution (30 fps).	environment and the application.		
Leap Motion	Range: 1m, Frame rate: 200 fps, precision: 0.7 mm.	Use hand movements and gestures to interact with the environment and the application.	Recognize hand movements and gestures.	Recognize hand movements and gestures.
Myo	Sampling data rate for electromyography is 200. Hz, sampling date rate for Inertial Sensor is 50 Hz.	Use gestures to interact with the environment and the application	Recognize gestures,	Recognize gestures, use vibrations as feedback on some activities.
Lumo Lift	Vibrational posture feedback, Bluetooth communication.	None	Recognize posture.	Vibration feedback.
Alex posture tracker	Vibrational posture feedback, Bluetooth communication.	Same as Lumo (above).	Same as Lumo (above).	Same as Lumo (above).

3.3 First Prototype: Documentation, Experiences and Requirements

Several components of the WEKIT system have been developed. In this section, we describe these components, their structure and the experience gained. The first version of the prototype aims to explore the following essential functions of the WEKIT system:

1. **Sensor fusion:** with this prototype, we aim to explore the combination of different kinds of sensors within one infrastructure.
2. **User interface:** the sensor fusion component should be supplied with an experimental user interface for configuring the sensor usage, recording, editing, and visualizing sensor data in order to test use case coverage.
3. **Backend infrastructure:** we aim to explore architectural insights into storing/retrieving high amounts of recorded data.

3.3.1 Sensor Fusion Component

For the first prototype of the sensor fusion component, we decided to take a set of off-the-shelf sensor products and combine them into one infrastructure. These sensor products are selected according to the recommendations derived from Deliverable D3.1 (Sensor Specification) and comprise:

- Microsoft Kinect (motion and posture tracking, full-body) (Kinect, 2016)
- Leap Motion (motion and posture tracking, hands only)
- Myo (motion and gesture tracking, arm movement)
- audio
- video

As a first step, we explored the amount of data delivered by each sensor per frame and calculated storage requirements for uncompressed/compressed storage of raw sensor data. We assumed a data resolution of 60 frames per second for each sensor. Table 4 lists the raw and compressed data amounts for various sensors as measured per frame, second, and minute.

Table 4. Amount of Data (in Kilobytes) for Individual and Combined Sensors

	Device						
Duration	Kinect	Myo Armband	Leap Motion (No hands)	Leap Motion (1 hand)	Leap Motion (2 hands)	1 file (Kinect, Myo, Leap (no hands))	1 file (Kinect, Myo, Leap (2 hands))
Raw Data							
1 Frame	7	0.8	1	2	2.5	10	11
60 Frames	343	4	3.5	19	31	352	380
3600 Frames	20,000	205	148	1,000	1,800	21,000	22,500
Compressed Data							
1 Frame	1.5	0.5	0.5	0.8	0.9	2	2.4
60 Frames	41	1.2	1	3	5	43	48
3600 Frames	2,400	37	28	133	222	2,500	2,600

With this information, we designed a file format to combine sensor data from the sensors selected into a combined data stream. This format is XML-based and follows the logic of multi-track recordings in order to allow to select, replace, or edit tracks individually. For each sensor, the original sensor data stream is included as binary data into the sensor's track inside the file.

To opt for extensibility of the sensor fusion component, an architecture has been defined, which specifies a general base class (WEKITPlayer). This base class defines the infrastructure for capturing and replaying sensor data. For each specific sensor device, a special subclass is available (AudioPlayer, KinectPlayer, MyoPlayer, etc.), some of which define own specialised data types (MyoData, CapsuleHand). This architecture allows for easy extension (new sensor types) while maintaining a common interface (see figure 2).

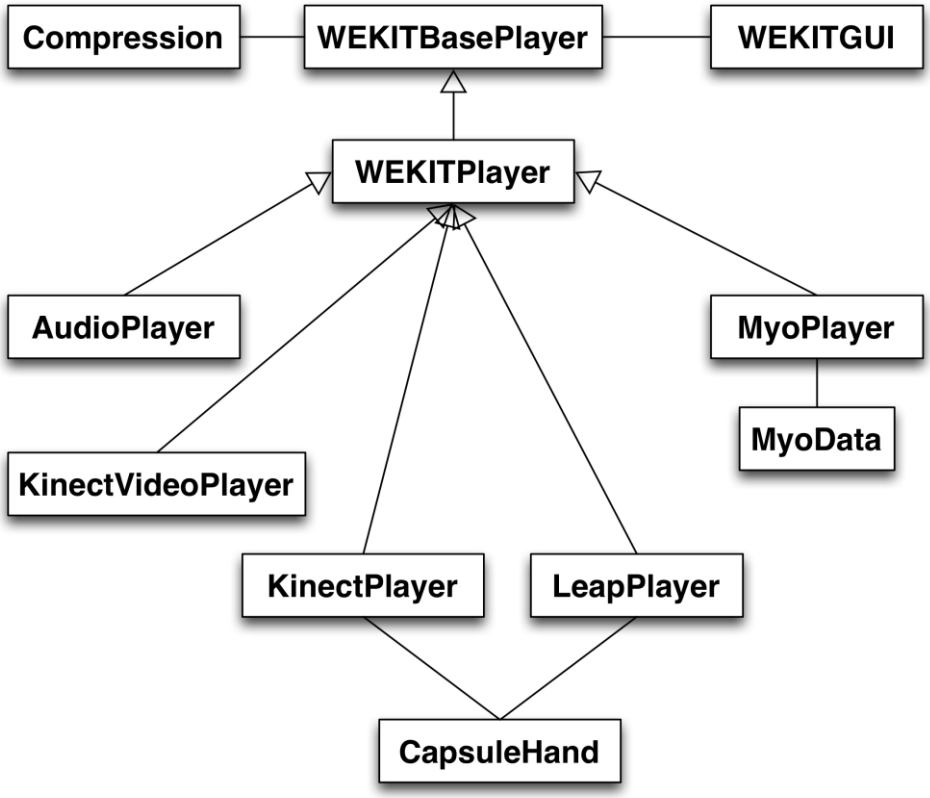

Figure 2. Class Diagram of First Prototype

3.3.2 User Interface

The first version of the user interface is designed for using the sensor fusion component as a standalone system on a PC. It's main aim is to serve as a functional prototype, which enables to access and test the sensor fusion component's functionality allowing to experiment with the recording and re-enactment of the various supported sensors.

The first prototype offers the following use cases (see Figure 3 for a screenshot):

- Configuration of sensor components and specification of different recording and re-enactment setups. This functionality refers to the design of various different transfer mechanisms: each transfer mechanism as specified in deliverable D1.3 (WEKIT Framework and Methodology) corresponds to specific sensor requirements (see D3.1, sensor specification).
- Starting a new recording session. The system visualises the captured sensor data during recording for all sensors selected to be part of the session.
- Loading and replaying a recorded session. The system re-visualizes the sensor data from the recording for all recorded sensors.
- Editing a recorded session. The editing feature allows to replace individual tracks in a recording with re-recordings. To do so, it visualizes all recorded tracks (except for the

one to be re-recorded) and records and visualizes the selected sensor synchronised to the existing tracks.

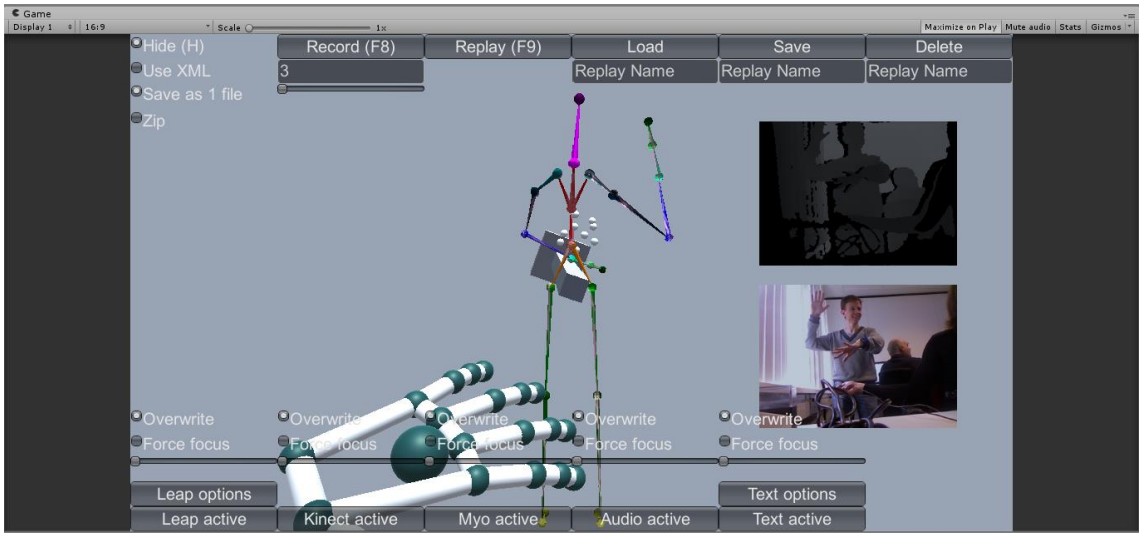

Figure 3. Screenshot of the Sensor Integration Component

We conducted first trials (see Figure 4) where we recorded different assembling actions for Lego Mindstorm using the described framework. Findings from these tests show that it is really difficult to make sense out of the sensor data alone. This means that using the sensor data alone in order to identify actions and give feedback on them does not seem to be a practical solution. Therefore, we propose to follow a very structured approach to capture the learning experiences, as described in the process model definition (section 4.3.1).

Figure 4. First Wearable Enhanced Learning Experience Capturing Experiment

3.3.3 Backend infrastructure

The current version of WEKIT's backend infrastructure is a *Repository of Learning Experiences* which is based on the ECO Learning Analytics infrastructure. It implements a version of the Learning Record Store (LRS) developed by the OUNL for the ECO project (Ternier et al., 2016) and also used for a similar multimodal data collection study called *Learning Pulse* (Di Mitri et al., 2016). This LRS is able to pull the data from the third-party APIs, transform them into learning records and handing out their identifiers. The learning records are stored into a "fact table" (Learning Locker) and assigned Universally unique Identifiers. As shown in Figure 5, ECO Learning Analytics (ECO LA) has three components running on the Google Cloud: an xAPI master a BigQuery and a Learning Locker. This setup allows to balance the load of data on a distributed architecture for scalability purposes.

From the Learning Locker the data are synchronised into a Big Query index which, contrarily to the Learning Locker, allows to query the distributed learning statements with SQL language. The synchronisation between the Learning Locker and the BigQuery Index happens using a queue, such that no learning records get lost. As the amount of data being received from the sensor could get really big, the computational approach used to develop the ECO LA is MapReduce, a framework developed by Google for executing very large amounts of computation in a short time. At the core of MapReduce there are two functions: a *map* function which starts from an object and generates a set of key/value pairs for this object. A *reduce* function merges all values that are associated to one key.

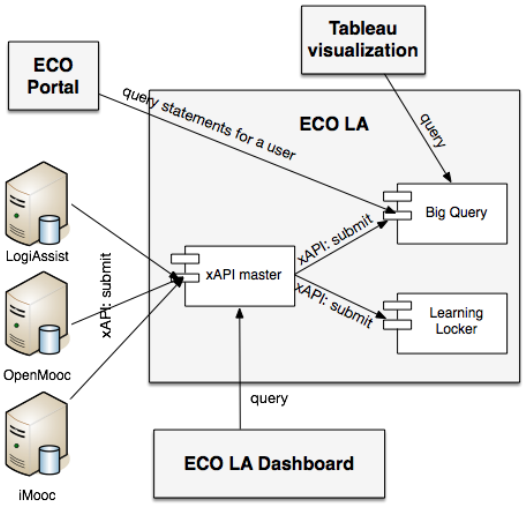

Figure 5. ECO Learning Record Store Architecture

4. Experience Capturing API Specification

While chapter 3 reports on the existing components and infrastructures of the WEKIT XCAPI prototype, this section specifies the XCAPI for further development into the final version. In important aspect of moving towards the final prototype is its transfer into the final architecture,

which includes the technical architecture for the wearable part as depicted in figure 1 and detailed in the parallel deliverable D3.2 (Hardware prototype with component specification and usage description), but also refers to the overall system architecture as described in the parallel deliverable D2.1 (Functional and modular architecture).

This section is organised along the three main connection points of the Experience Capturing API (XCAPI):

- *API to Hardware* describes how the XCAPI interacts with the low-level API provided by the sensor component. Here, function calls to retrieve/accept sensor updates are bundled and methods to activate/deactivate specific sensors are provided.
- *API to Application* Modules describes the high-level functionality as offered to end-user application level software. These build on the general functionalities offered in the first prototype mapped to the wearable scenario.
- *Backend API* describes the interaction of the XCAPI with the backend infrastructure for storing/retrieving recorded data.

4.1 API to Hardware

The core functionality of the XCAPI is to capture sensor data for the selected sensor component in real-time or close to real-time. Hence, data from multiple sensors needs to be gathered, processed, and prepared for storage.

Additionally, previously captured and stored sensor data needs to be synced with current sensor data to allow for editing, comparison, and guidance (this concept is similar to "ghost tracks" as known from e.g. racing games).

Furthermore, the XCAPI relies on an abstracted access to each sensor device category provided by the sensor controller component. This means, that e.g. independent from the concrete manufacturer of an EEG sensor, the XCAPI receives the attention level from the sensor controller, rather than the raw sensor data. Some sensors, where raw data is essential (e.g. audio data) may be exceptions to this approach.

Based on these core requirements, the XCAPI defines a number of PUSH (the sensor controller actively informs the XCAPI about events and updates) and PULL (the XCAPI queries information from the sensor controller) methods, to interact with the sensors.

Consequently, for each high-level function (attention, stress level, heart rate, etc.) the following methods are defined:

- *PUSH methods*: activation, deactivation, updating, events/errors. Below we list these functions using the attention (EEG) sensor as an example:
    - *activateAttention*: called by the sensor controller, as soon as an attention sensor (EEG) is connected and ready to be used.
    - *deactivateAttention*: called by the sensor controller, when the attention sensor (EEG) is disconnected and can no longer be used.
    - *updateAttention*: called by the sensor controller to indicate a significant change in the attention value.

    - *attentionEvent*: called by the sensor controller to indicate specific events such as low battery.
- *PULL methods*: querying availability of a sensor, querying sensor state, opening/closing a sensor input stream:
    - *isAttentionAvailable*: called by XCAPI to check, if an attention sensor is available in the infrastructure.
    - *getAttention*: called by XCAPI to get the most current attention value from the sensor controller.
    - *openAttentionStream/readAttentionStream/closeAttentionStream*: if the sensor controller ensures data for a sensor to be available in a specific frequency (e.g. 10 values per second), stream based access can be used.

Accordingly, other sensors and corresponding high-level functions are mapped to the XCAPI. Table 5 lists these different high-level functions and the key functions (PUSH and PULL) to be provided for them.

Functionality: at lowest levels the drivers will be presented as a C++ library with a standardised API plus device specific extensions. Upon this a language-agnostic abstraction layer should be built. One suggestion is protocol buffers from Google (proto-buf) https://developers.google.com/protocol-buffers/ which is light-weight, but only supports C++, C#, Go, Java, and Python. Another suggestion is https://thrift.apache.org/ . Thrift has a much larger list of supported languages. Both are well supported, have permissive licensing, and are able to run both locally and and across network connections, thereby offering flexibility of architecture.

Both proto-buf and Thrift use an underlying communication method, most usually RPC, which allows cross-platform support. The details of the operating system on either end is hidden by the framework. It also allows the API to be accessed over a network. As detailed in D3.2, Apache Thrift has been chosen as the communication library.

Table 5. XCAPI Calls from/to hardware modules

HighLevel Value	**Pull Functions**	**Push Listeners**	**Return Value Type and Range**	**Sensors**
Attention	getAttention isAttentionAvailable openAttentionStream closeAttentionStream	activateAttention deactivateAttention updateAttention attentionEvent	Integer, 0-100	EEG
HRV	getHRV/getHR	activateHrv deactivateHrv updateHrv hrvEvent	Integer	HR tracker
StressLevel	getStressLevel	activateStressLevel deactivateStressLevel updateStressLevel stressLevelEvent	Integer, 0-100	Skin conductance
Location	isRelLocationAvailable	activateRelLocation	Coordinates	GPS, Depth

(relative to a specific reference location / absolute to geographic location)	getRelLocation openRelLocationStream closeRelLocationStream isAbsLocationAvailable getAbsLocation openAbsLocationStream	deactivateRelLocation updateRelLocation relLocationEvent activateAbsLocation deactivateAbsLocation updateAbsLocationListener absLocationEvent		Camera
Posture	getPosture isPostureAvailable openPostureStream closePostureStream	activatePosture deactivatePosture updatePosture postureEvent	3D Model, Skeleton of Joints	Depth Camera, Kinect
HandPosture (separate for left / right hand)	getRightHandPosture getLeftHandPosture isRightHandPostureAvailable openRightHandPostureStream closeRightHandPostureStream isLeftHandPostureAvailable openLeftHandPostureStream closeLeftHandPostureStream	activateRightHandPosture deactivateRightHandPosture updateRightHandPosture RightHandPostureEvent activateLeftHandPosture deactivateLeftHandPosture updateLeftHandPosture leftHandPostureEvent	3D Model, Skeleton of Joints	Leap, Depth Camera
Direction	getDirection		3D Vector	
Gaze	getGaze		Relative Coordinates (x,y,z)	
Video	isVideoAvailable openVideoStream closeVideoStream		Video	
Audio	isAudioAvailable openAudioStream closeAudioStream			

4.2 API to Application Modules

XCAPI will offer functionalities that will allow flexible implementation by end-user applications regardless of which Transfer Mechanism is implemented. These APIs will allow standard mechanisms for the application to access the stored interpreted data. These APIs are guided by the WEKIT framework which defines the core functionalities of each type of sensor data. The functionalities partly ensure successful adherence to the framework at the level of end-user application. XCAPI defines standard data storage formats for each data type. Each type of sensor data in every packet will inherit from their parent methods. A parent class will include all

common methods for a particular sensor data type, for example the “Gaze Behaviour” class would have a method DeletePoint() which would allow deleting certain points in any packet of expert data regardless of the device used to capture the gaze behaviour data.

Table 6. XCAPI to Application

High-level Value	Functions	Return Type
Gaze behaviour	getData() isDataAvailable() getPacketId() deletePoints() gazeCompare(node, current gaze)	Gaze co-ordinates (x,y,z)
Point of View	getData() isDataAvailable() getPacketId() CreateClips(time1,time2) tagClips() replay(speed)	Video
Audio	getData() isDataAvailable() getPacketId() tagAudio() replay()	Audio
Motor Performance	getData() isDataAvailable() getPacketId() ghostReplay() makeTaggable(3djoints) compareTolerance()	3D model recording in free space

Through the user interface, users can get access to the different sensors’ information and recorded learning materials. For this, the user interface needs to have an API to access the data streams provided by the different sensor components as well as the recorded material.

The API should also provide:

- Selection of sensors to be used for the recording.
- Retrieve recorded material from the cloud.
- Levels of authentication, this deals with different restrictions and priorities that the different type of users can have. For example, experts by default have their recordings and annotations public, while learners might have them private by default.
- A feature to create, store and retrieve annotations.

4.3 Backend API

Process Model Definition

The process will allow the capture and reenactment of expert's experience, so that it can be transferred to the learner later. One important step in capturing the expert's experience is to identify and define the actions performed by an expert in order to complete some predefined task. Experts are usually not good at identifying all the actions needed in order to perform a task, therefore it is important that experts together with educational designers define the tasks. One problem in the definition of actions is to define a consistent and useful level for granularity of them. The definition here is inspired by the emerging ARLEM standard (see also parallel deliverable D2.2, Learning Experience Content Model Draft). ARLEM defines a standard for abstracted, high level experiences, which will be complemented by the low level, sensor data specific model and storage concept as defined here.

Our current proposal is to define an action as following:

Action: Atomic meaningful procedure performed by the learner or expert.

Example: Following the 1st step of an instruction Manual X (we used Lego Mindstorm assembly as a first example in our trials and here refer to it as Manual X). See Table 7 for a reference of the information stored as Actions.

Once the actions are defined, experts and learners can record them using the WEKIT framework. These recorded actions are stored as Action Runs. The definition of an Action Run is the following:

Action Run: It is the instantiation of an action performed by an expert or a learner.

The action runs contain a sensor recording that contains the data streams of all of the sensors that were used to capture the action performed by the user. Before the recording of each action run, users can select the type of sensors needed for it. We recommend that experts together with educational designers come up with a list of the recommended sensors to be used for the recording of each of the actions, in order to use this list of sensors as default for the recordings of the action runs.

Sensor Recording: A saved stream of sensor data extracted by one or multiple sensors used for the recording of an activity or annotation.

Each sensor recording is built by one or many sensor data streams. Sensor Recordings are captured to memory, stored locally on the wearable device and communicated to the backend storage asynchronously.

Sensor Data Stream: It is the data stream of one particular type of sensor for a sensor recording.

One important part of the WEKIT is to use AR to provide learners with information that will help them to learn and to perform the actions in a correct manner. This information should be

available to the users while performing an action run. In order to do that we propose to add Annotations to each of the actions.

Annotations: Extra information that can help the learner to perform the action correctly. These annotations can be referred to the transfer mechanisms already identified.

Examples of annotation types are: post-it notes, videos, audios, sensor recordings, etc.

Annotations have some condition on when or where to be shown, for example: showing arrows pointing the direction where to look at when the user is facing the wrong way. Experts and learners can add annotations to every action, therefore by default the recommendation is to have expert annotations as public and learners' annotations as private. A whole action run could also be annotated to an action, serving a similar function to the learner as a ghost track in racing games as mentioned above.

Examples for the Action-Annotation process model:

- Expert together with educational designers define an action, such as following first step of Manual X.
- Expert with the help of educational designers start creating annotations for the Action.
    - The annotations can be the sensor recording of an Action Run (the whole action performed by the expert).
    - Video fragments with voiceover
    - Post it notes
    - Recorded gestures using leap motion.
- Learner selects to perform the first step of Manual X.
- Learner creates an action-run and starts performing the action.
- While performing the action, the learner through the system starts presenting to the learner the annotations that belong to the action.
- The learner can pause the recording and create a new annotation to the action (post-it "remember to hold Tool X with left hand").
- After finishing the action run and saving it. The learner adds this action run as an Annotation to the action so that the next time he performs the action he can compare its current performance against his previous one (similar as the "Mario Kart Ghost").

Table 7 exemplifies how actions could be defined for assembling the wheels of the Lego Mindstorm. The Action_id represents the steps in the manual. As seen in the table the first step Mindstorm_Tracker_Wheels_1 has no previous actions, and it is followed by the action Mindstorm_Tracker_Wheels_2. For each action there are some required materials, in the example shown in the table for the first step a 13-holes black long piece is needed. The table also shows examples on how annotations can be added to the specific actions. As an example the first row of the table in the field of annotations points out that all the assembled connectors should point to the same side.

Table 7. Example Actions Required to Assemble the Wheels of the Lego MindStorm Based on the Manual

Action_id	Previous Actions	Following Actions	Required Material	Description	Annotations

Mindstorm_Tracker_Wheels_1	none	Mindstorm_Tracker_Wheels_2	13-holes black long piece…	First place the 2…	Avoid placing connectors pointing to different directions
Mindstorm_Tracker_Wheels_2	Mindstorm_Tracker_Wheels_1	Mindstorm_Tracker_Wheels_3	Two 3-wholes black piece	…	…
Mindstorm_Tracker_Wheels_3	Mindstorm_Tracker_Wheels_2	Mindstorm_Tracker_Wheels_4	Rin, 8-connector	…	…
Mindstorm_Tracker_Wheels_4	Mindstorm_Tracker_Wheels_3	Mindstorm_Tracker_Wheels_5	yellow stopper	…	…
Mindstorm_Tracker_Wheels_5	Mindstorm_Tracker_Wheels_4	Mindstorm_Tracker_Wheels_6	8-connector, 2-hole black bar, 2-hole black bar	…	…
Mindstorm_Tracker_Wheels_6	Mindstorm_Tracker_Wheels_5	Mindstorm_Tracker_Wheels_7	2-hole bar, black connector	…	…
Mindstorm_Tracker_Wheels_7	Mindstorm_Tracker_Wheels_6	Mindstorm_Tracker_Wheels_8	5-connector, red-stopper, thin-rin, thin-rin	…	…
Mindstorm_Tracker_Wheels_8	Mindstorm_Tracker_Wheels_7	Mindstorm_Tracker_Wheels_9	5-connector, yellow-stopper, rin	…	…
Mindstorm_Tracker_Wheels_9	Mindstorm_Tracker_Wheels_8	Mindstorm_Tracker_Wheels_10	13-hole black long piece	…	…
Mindstorm_Tracker_Wheels_10	Mindstorm_Tracker_Wheels_9	Mindstorm_Tracker_Wheels_11	black connector	…	…
Mindstorm_Tracker_Wheels_11	Mindstorm_Tracker_Wheels_10	Mindstorm_Tracker_Placing_Wheels	Rubber band	…	…

Storage Architecture

In order to implement the previously presented process model definition the following list of tables is required:

Actions		
Action_id	Name	Description

Action-Flow		
Action(action_id)	Follows(Action_id)	Condition

Action-material			
Action_id	Material	Quantity	Picture

Actions-Run			
Action_Run_id	Action_id	User_Id	Sensor_Recording_Id

Annotation					
Annotation_id	Type	Link	User_id	Action_Id	privacy

Sensor_Data_Stream		
Sensor_Data_id	Sensor_type	Link

Sensor_Recording		
Sensor_Recording_id	Sensor_Data_id	User_id

Users		
User_id	Name	Level of expertise

By looking at the tables it is possible to see that we have only defined links to access the sensor data or other types of data used for the annotations. The idea is to store them as blobs or as files that can be accessed through a link.

Experience API Integration

The instances of the actions which are the Action Run can be mapped one-to-one to an Experience API (xAPI) statement[1]. xAPI is an open source RESTful web service through which systems send learning information to the Learning Record Store. The xAPI was inspired by the Resource Description Framework xAPI is made by triples of having the format *actor-verb-object* which are generated and exchanged in JSON format, opportunely validated by and stored in the LRS. The main advantage of using xAPI is interoperability: learning data from any system or resource can be captured and eventually queried by third party authenticated services. The JSON code shown in Listing 1 shows a sample xAPI statement taken from the Dutch Specification of Learning Activities (Scheffel et al. 2016) of the type: “user likes a blog-post”. We think an Action Run like “user assembles wheels” can have similar structure. The *context* place would be the section where which points to some specific sensor data.

```
{
   "timestamp":"2015-06-01T08:30:48Z",   "id":"abcdefghijk123456789",


   "actor":{
      "objectType":"Agent",
      "account":{
         "homePage":"http://URL_of_the_Source_LMS?With_The_UserID=0388437472",
         "name":"0388437472"
      }
   },
   "verb":{
      "id":"http://activitystrea.ms/schema/1.0/like",
```

[1] xAPI (Experience API) must not be confused with XCAPI (Exeperience Capturing API)

```
      "display":{
         "en-US":"Indicates the learner liked something"
      }
   },
   "object":{
      "objectType":"Activity",
      "id":"http://URL_of_the_Source_LMS/exampleblogpost.html",
      "definition":{
         "name":{
            "en-US":"name of the blogpost"
         },
         "description":{
            "en-US":"This is a blog post"
         },
         "type":"http://www.ecolearning.eu/expapi/activitytype/blogpage"
      }
   },
   "context":{
      "extensions":{
         "http://activitystrea.ms/schema/1.0/place":{
            "definition":{
               "type":"http://activitystrea.ms/schema/1.0/place",
               "name":{
                  "en-US":"Place"
               },
               "description":{
                  "en-US":"Represents a physical location."
               }
            },
            "id":"http://vocab.org/placetime/geopoint/wgs84/X-15.416497Y28.079203.html",
            "geojson":{
               "type":"FeatureCollection",
               "features":[
                  {
                     "geometry":{
                        "type":"Point",
                        "coordinates":[
                           -15.4164969,
                           28.0792034
                        ]
                     },
                     "type":"Feature"
                  }
               ]
            },
            "objectType":"Place"
         }
      },
      "contextActivities":{
         "parent":{
            "id":"http://URL_of_the_Source_LMS/exampleMOOC.html",
            "objectType":"Activity",
            "definition":{
               "name":{
                  "en-US":"name of the MOOC"
               },
               "description":{
                  "en-US":"This is the originating MOOC"
               },
               "type":"http://adlnet.gov/expapi/activities/course"
            }
         }
      }
   }
}
```


Listing 1. Example xAPI Statement

Retrieval Processes

Considering the previously defined process model and corresponding storage architecture Table 8 presents the interfaces needed to store, manipulate and access the recorded data.

Table 8. Interfaces to access, store and manipulate the recorded data

Method	Input	Output	Explanation
getActions		List of all Actions	

getAction	id_action	Action	
insertAction	Description Name	true/false	
insertActionMaterial	id_action <List of Material>	true/false	
getActionMaterial	idAction	List of materials	
insertActionFlow	id_action(current action) id_action(previous action) condition	true/false	
getFollowingAction	id_action	Action	
getPreviousAction	id_action	Action	
insertActionRun	Action_id User_id <list>SensorData	true/false	This method should also upload the sensorData and create the corresponded entries for the SensorRecordings
getActionRun	id_Action_run	Action Run	
insertSensorRecording	<list>SensorTypes <list>SensorData user_id	true/false	
getSensorRecordings		<list>Sensor Recordings	
getSensorRecording	sensorRecording_id	Sensor Recording	
getSensorRecordingBySensor	sensorRecording_id sensor	Sensor_data_stream	
insertAnnotation	User_id Action_id Type Annotation_data(sensor, videos, audios, text,etc.)	true/false	This method should upload the Annotation_data into a file or a blob, and save the link to the file or blob in the new created entry.
getAnnotationsByActions	Action_id	<list>Annotation	
getAnnotationsByActionUser	Action_id User_id	<list>Annotations	

getAnnotationsByActionUserExperience	Action_id User_Experience	<list>Annotations	

5. Recommendations & Usage

XCAPI provides to different developers (e.g. application developer, infrastructure provider, and hardware specialist) methods to implement the WEKIT framework efficiently. However, effective implementations should consider recommendations that the WEKIT framework has established. Strict adherence to the methodological approaches toward capturing and enacting data is required to extract the highest potential of XCAPI. Expert, Application developers and Instructional designers have equally important and closely tied roles in framing the learning platform.

The capturing of the expert's performance is complemented by the task analysis during which experts, application developers and instructional designers need to collaborate. The expert should take the lead in defining the granularity of actions and validating each captured packet. While application developers are required to support the expert and instructional designers by creating tools that allow easy handling of the data packets. We strongly recommend having two-fold recording phases where during the initial phase the expert records his performance. This recording does not need to be executed with the WEKIT framework, a simple video recording can do the work. This recording should be reviewed by experts in coordination with educational designers and programmers in order to identify aspects such as the granularity of the actions and the important aspects to be annotated in each of them. Once this first analysis is conducted, we suggest to record the experience of the expert once more using the WEKIT framework following the previously defined structure and taking in consideration the findings from the analysis of the first recording.

It should be noted that XCAPI does not handle high-level pedagogic methods such as feedback and reflection. These elements are defined in the task analysis and it is in the best interest to adapt these to the particular problems being addressed. We also recommend that the recording and re-enacting to be performed in similar if not exact environments and that the starting orientation of the user in that the relative 3D space (room) be aligned by certain methods. In that perspective the 3D space captured during the expert recording may be re utilized for the ideal implementations. Therefore the editing of the packet data may also be performed in the same 3D space for consistency. Allowing an expert to view his recording from a third perspective, while being able to annotate assets into the 3D space will make editing easier and innovative.

6. Conclusions

In this document we present the first iteration of the defined interfaces to be used in order to access the data from the available sensors that capture the user experience. To create these

interfaces, a corresponding data structure that allows to store the learning materials is defined. This learning materials include:

- A data structure to define the actions performed by a learner or expert to complete a task.
- The captured learning experiences
- Annotations to the actions

Following this data structure the document also provides a proposed interface to store, retrieve and manipulate the learning material. This procedure is outlined along the complete procedure from capturing the data to making it available for implementation. An accompanying set of recommendations for using XCAPI makes the whole implementation more rigid. Thid deliverable furthermore elaborates on how the XCAPI adapts the WEKIT framework and requirements generated in D1.3 and D1.4.

Together with the parallel deliverables D2.1 (Functional and Modular Architecture: Requirements and Specification), D2.2 (Learning Experience Content Model Draft), and D3.2 (Hardware Prototype with Component Specification and Usage Description), this deliverable sets the foundation for the next iteration of the WEKIT technology development and thus immediately guides activities towards the first integrated version of the WEKIT.One solution.

References

Articles

Di Mitri, D., Scheffel, M., Drachsler, H., Börner, D., Ternier, S., & Specht, M. (2016). Learning Pulse: using Wearable Biosensors and Learning Analytics to Investigate and Predict Learning Success in Self-regulated Learning. CEUR Proceedings, 1–6. Retrieved from ceur-ws.org/Vol-1601/CrossLAK16Paper7.pdf

Scheffel, M., Ternier, S., & Drachsler, H. (2016). The Dutch xAPI Specification for Learning Activities (DSLA) – Registry. Retrieved from http://bit.ly/DutchXAPIreg

Ternier S., Loozen K., Viñuales J., Tejera S., Tomasini A., Unruh S. (2016) D3.6 Report on implementation of the ECO Federated Search infrastructure. Retrieved from http://project.ecolearning.eu/wp-content/uploads/2016/03/ECO_D3.6-Report_of_ECO_Federated_Search_Infrastructure.pdf

Related WEKIT Deliverables

D1.3 WEKIT Framework and Training Methodology (v1, M5)

D1.4 Requirements for Scenarios and Technological platform (v1, M6)

D2.1 Functional and Modular Architecture: Requirements and Specification (M12)

D2.2 Learning Experience Content Model Draft (M12)

D3.1 Requirement Analysis and Sensor Specifications (M12)

D3.2 Hardware Prototype with Component Specification and Usage Description (M12)

Other references

Alex (2016). Alex posture tracker product page: http://alexposture.com/

Hololens (2016). Microsoft Hololens product page: https://www.microsoft.com/microsoft-hololens

Intel (2016). Intel RealSense developer documentation: https://software.intel.com/sites/landingpage/realsense/camera-sdk/v1.1/documentation/html/index.html

Kinect (2016). Microsoft Kinect developer page: https://developer.microsoft.com/en-us/windows/kinect

Leap (2016). Leap Motion product page: https://www.leapmotion.com/

Lumo (2016). Lumo lift posture coach product page: http://www.lumobodytech.com/

MyndBand (2016). MyndPlay MyndBand + MRT research toolkit product page: http://store.myndplay.com/products.php?prod=10

Myo (2016). Myo armband product page: https://www.myo.com/

Neurosky (2016). Neurosky homepage: http://neurosky.com/

ODG (2016). ODG R-7 product page: http://www.osterhoutgroup.com/products-r7-glasses

Wearable Experience for Knowledge Intensive Training
Project No 687669

WEKIT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687669. http://wekit.eu/