D2.3

Final Architecture and Learning Experience Content Model

Editor:
Fridolin Wild (OBU)
Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Contributor(s)</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>5.12.2018</td>
<td>OBU (Wild)</td>
<td>First version assembled</td>
</tr>
<tr>
<td>1.0</td>
<td>14.3.2019</td>
<td>OBU (Wild)</td>
<td>Final version assembled</td>
</tr>
<tr>
<td>1.1</td>
<td>15.3.2019</td>
<td>OBU (Wild)</td>
<td>Post review version</td>
</tr>
</tbody>
</table>

Disclaimer: All information included in this document is subject to change without notice. The Members of the WEKIT Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the WEKIT Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.
Final Architecture and Learning Experience Content Model

WP 2 | D2.3

Editors:
Fridolin Wild (OBU)

Authors:
Fridolin Wild (OBU), Will Guest (OBU), Alla Vovk (OBU), Kaj Helin (VTT), Jaakko Karjalainen (VTT), Roland Klemke (OUNL), Bibeg Limbu (OUNL), Daniele Di Mitri (OUNL), Jordi Hutjens (OUNL), Mark Ransley (RAVE), Carl Smith (RAVE), and external contributors to the standard (see cover pages)

Reviewers:
Christine Perey (IEEE SA P1589), Paul Lefrere (CCA)

<table>
<thead>
<tr>
<th>Deliverable number</th>
<th>D2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level</td>
<td>Public</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Date</td>
<td>15.3.2019</td>
</tr>
<tr>
<td>Due date</td>
<td>M39</td>
</tr>
</tbody>
</table>
# Table of Contents

REVISION HISTORY ........................................................................................................2

EXECUTIVE SUMMARY .................................................................................................5

1. INTRODUCTION ........................................................................................................7

2. ARLEM CONTENT MODEL .......................................................................................8

3. AR TRAINING SYSTEM ARCHITECTURE ................................................................43

3.1. WORKPLACE MANAGER OF THE PLAYER .........................................................44

3.2. ACTIVITY MANAGER OF THE PLAYER ...............................................................45

3.3. SENSOR INTERFACE OF THE PLAYER .................................................................47

3.4. PROCESSING BEHAVIOUR OF THE PLAYER .......................................................48

3.5. ACTIVITY CONTROLLER OF THE RECORDER ..................................................49

   ActivityController.cs .................................................................................................49

   TaskStationController.cs ..........................................................................................49

   AnnotationController.cs ..........................................................................................50

3.6. ANNOTATIONS .......................................................................................................50

   3.6.1. Adding or Modifying Annotation Types .......................................................50

   3.6.2. Giving an Annotation Functionality ...............................................................51

   3.6.3. Changing the View .........................................................................................51

   3.6.4. Linking view to a function in the code ...........................................................51

   3.6.5. Adding a new link to a code function .............................................................51

4. CONCLUSION .............................................................................................................52
Executive summary

Within this deliverable, we report on two key activities of WP2, namely the content model for Augmented Reality Learning Experience Models (ARLEM) and the AR training system architecture.

As for the first, the content standard ARLEM, the following can be summarised.

It should be noted that the content model specification comes in a different format than the rest of the deliverable: The specification is a formal submission to the P15891 working group of the IEEE standards association, with the lead editor at OBU (Dr Fridolin Wild), both chair of the working group and scientific director of WEKIT.

The specification serves as the interchange format, providing the representation needed to collate WP3 recording and sensor data into consumable learning activities and workplace models needed for the WP4 re-enactment system for performance augmentation.

ARLEM went through a major revision from the last version to the present, essentially upgrading large parts of the specification following the feedback from the balloting committee of the IEEE standards association. This has resulted, most notably, in a more strict separation of formal and informal parts of the specification, and a complete update of the definition parts, removing the previous definitions by example. Moreover, the overall processing behaviour, the box model for the activity model and the workplace model, as well as basic system dialogues have been fitted with better, conceptual explanations (and illustrations). In total, these changes satisfy the 126 comments received from the balloting committee.

Table 1 lists an example of such comment and the way the ballot response group reacted to it. Of those 126 balloting comments, 105 were accepted, 17 accepted in kind but a different solution than proposed (‘revised’), and five were rejected. More details can be found in deliverable D8.3.

<table>
<thead>
<tr>
<th>Page: 4</th>
<th>Subclause: 5</th>
<th>Line: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comment:</strong> This entire section is incorrectly presented. It should specify the elements in a normative fashion, without examples, and be followed by an informative section that gives examples.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Must be satisfied:</strong> Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Proposed change:</strong> Separate into normative section defining the meaning of the elements and an informative section giving examples, or, if desired, keep this as a normative section and define the formats, followed by examples that are clearly labelled as examples.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Disposition Status:</strong> accepted</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Disposition Detail:</strong> Reorganisation of the schema from body text to a table; moved examples to example column; marked anything 'informative' as 'Note--'.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Example balloting comment.
The example listed in Table 1 is a particularly big change request. There were a number of smaller comments, mostly grammatical or spelling corrections, but they were in the minority, see also the full list of changes in D8.3.

As for the second, the AR training system architecture, the following can be summarised.

Over the period from the first to the second prototype, the most notable change is that the two disparate subsystems, player and recorder, were fully integrated into a single software prototype, the ‘WEKIT.one unified’.

Functionality for the prototype now falls into several component groups, ‘common’ functionality shared across ‘player’ and ‘recorder, as well as ‘cloud’ repository functionality.

The trials provide evidence that the system landscape successfully achieved interoperability, allowing recordings from one device to be uploaded to the cloud repository, to then be downloaded onto another device to play. The system architecture documented here can serve as a blue print for others’ implementations of an AR training system.
1. Introduction

This deliverable reports on two outcomes of the WEKIT project, the ARLEM content model and the AR training system architecture.

The content model is a specification of an activity description language and a workplace modelling language, capable of describing learning and training workflows and their augmented execution in the environment in which they need to be carried out.

The system architecture is an abstracted description of the system implementation, a work product that has been guiding development and integration, that now can serve as blue-print for other AR training system implementations, allowing developers to re-use and learn from the chosen component organisation and modularisation.
2. ARLEM content model
Abstract: Augmented Reality (AR) promises to provide significant boosts in operational efficiency by making information available to employees needing task support in context in real time. At this time, however, there is no general-purpose conceptual model and data model specification for representing learning activities (also known as employee tasks and procedures) and the learning environment in which these tasks are performed (also known as the workplace).

This document proposes an overarching integrated conceptual model that describes interactions between the physical world, the user, and digital information, the context for AR-assisted learning and other parameters of the environment. It defines a first proposal of the two types of required data models, modeling languages, and their binding to XML and JSON.

Creating such interoperability specification and standard will help to open the market, adding interchangeable component products as alternatives to monolithic Augmented Reality-assisted learning systems. Moreover, it will enable the creation of experience repositories and online marketplaces for Augmented Reality-enabled learning content. Specific attention was given to supporting reuse and re-purposing of existing learning content and to cater to ‘mixed’ experiences combining real-world learner guidance with the consumption (or production) of traditional contents such as instructional video material or learning apps and widgets.

Keywords: Augmented Reality, E-Learning, Workplace Training, Learning Activity, Learning Experience, Performance Support, Immersive Learning Environment, Blended Reality
Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimers.html.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.
Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies
Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

**Updating of IEEE Standards documents**

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Xplore at [http://ieeexplore.ieee.org/](http://ieeexplore.ieee.org/) or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE’s standards development process, visit the IEEE-SA Website at [http://standards.ieee.org](http://standards.ieee.org).

**Errata**

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: [http://standards.ieee.org/findstds/errata/index.html](http://standards.ieee.org/findstds/errata/index.html). Users are encouraged to check this URL for errata periodically.

**Patents**

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at [http://standards.ieee.org/about/sasb/patcom/patents.html](http://standards.ieee.org/about/sasb/patcom/patents.html). Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.
1 Participants

At the time this draft standard was completed, the P1589 Working Group had the following membership:

Dr. Fridolin Wild, Chair
Christine Perey, Vice Chair
Giuseppe Scavo, Secretary

2

3

4

5

6

7

8

Alex Hill

Ingo Dahn

Mark Grant

9

Alexander Komar

Ioannis Brilakis

Matei Dusik

10

Alla Vovk

Jaakko Karjalainen

Maximilian Speicher

11

Anne Knowles

Jason Haag

Mikel Salazar

12

Ashley Fehr

Jayfus Doswell

Mikhail Fominykh

13

Avron Barr

Jim Belanich

Paolo Civardi

14

Barry Po

Joan Woolery

Patrick Selby

15

Carl H Smith

Joe Tojek

Paul Leferre

16

Christopher Brown

Johanna Pirker

Pavel Dvořák

17

Craig Wiggins

John Cook

Pete Wassell

18

Daniel Spikol

Jonathan Goldberg

Ralf Klamma

19

Douglas Knisely

Joseph Howse

Robert McCaffrey

20

Edison Arenas

Judy Bloxham

Roland Klemke

21

Erik Isaksson

Kaj Helin

Stefano Bianchi

22

Geoff Jarvis

Keith Kamikawa

Will Guest

23

Gerry Kim

Louise Yarnall

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

[To be supplied by IEEE]

Balloter1

Balloter2

Balloter3

Balloter4

Balloter5

Balloter6

Balloter7

Balloter8

Balloter9

When the IEEE-SA Standards Board approved this standard on <Date Approved>, it had the following membership:

[To be supplied by IEEE]

<Name>, Chair
<Name>, Vice Chair
<Name>, Past Chair
Konstantinos Karachalios, Secretary

SBMember1

SBMember2

SBMember3

SBMember4

SBMember5

SBMember6

SBMember7

SBMember8

SBMember9

*Member Emeritus
Introduction

This introduction is not part of P1589/D2.6, Draft Standard for Augmented Reality Learning Experience Model.

The next generation of performance support systems and tools for learning at the workplace is likely to be delivered in Augmented Reality (AR). While there are many impressive prototypes and bespoke applications, interoperability has been neglected so far and standards for creating and converting urgently needed AR training content at scale are absent. To address this gap, the IEEE Computer Society provides this specification as a remedy: the working group P1589 elaborated an AR Learning Experience Model (ARLEM).

The specification in development provides a data format for the enrichment and exchange of AR learning content, consisting at its core of an activity description language and a workplace model. The specification is built for describing AR learning experiences such as bringing a handbook to life or hands-free operator training with smart glasses.
Contents

1. Overview ........................................................................................................... 1
   1.1 Scope ........................................................................................................... 1
   1.2 Purpose ....................................................................................................... 1

2. Normative references ......................................................................................... 2

3. Definitions .......................................................................................................... 2

4. Learning in Activities and Workplaces ............................................................... 3

5. An Activity Modeling Language for Activity Scripts ............................................. 4

6. Workplace Modeling Language .......................................................................... 6

7. ActivityML base schema .................................................................................... 8
   7.1 Instruction ................................................................................................... 19
   7.2 Entry, Exit, Triggers .................................................................................... 19
   7.3 Interacting with Tangibles .......................................................................... 20
   7.4 Interacting with actions ............................................................................. 20
   7.5 Triggers ....................................................................................................... 21
   7.6 Logging user activity using the Experience API (xAPI) ............................. 21

8. Workplace Modeling Language Base Schema .................................................... 22
   8.1 Tangibles, Configurables, and Augmentations ........................................... 31

Annex A (Informative) Bibliography ...................................................................... 34
Draft Standard for Augmented Reality Learning Experience Model

1. Overview

1.1 Scope

This is a specification that defines two description languages for expressing Augmented Reality (AR) Learning Experiences. This document specifies how to represent activities conducive to developing or upgrading knowledge, skills, abilities, and other characteristics in a standardized interchange format. The interchange format links the representation of learning activities with their environment and context in the actual workplace in which an AR-enhanced training system can execute them. The specification aims to lower entry barriers for authoring of learning experiences that involve interaction with the real world, sensors, computer vision, and web applications.

NOTE—Notes to text, tables, and figures are for information only and do not contain requirements needed to implement the standard.

NOTE—An example of such learning experience: Astronauts can be trained using an Augmented Reality learning experience while on the ground or when in space, practicing how to perform an assembly procedure of a temporary stowage rack. In this case, the AR training system executes the learning activity represented, using the activity description language set out in this standard in order to provide procedural guidance live and in context. The attention of the astronauts in training can be directed to relevant parts of the rack and wall mount by overlaying visual instruction and 3D animations, explaining step-by-step what needs to be done.

NOTE—Another example: Aeronautics service technicians in training can build expertise in how to rig an aircraft engine. For this, the AR training system provides activity guidance on how to handle and adjust the various components associated with the control system in order to optimize engine performance.

NOTE—Another example: Doctors in training can practice a diagnostic procedure for detecting pulmonary embolism, being guided and receiving explanatory support for different conditions and the way these conditions manifest with different imaging technologies.

1.2 Purpose

ARLEM provides an overarching integrated conceptual model and the corresponding data model specifications for representing activities, learning context and environment (aka 'workplace'), while linking with other data model components needed for AR-enhanced learning activities.
The standard distinguishes slow-changing data for environment description from fast-changing data for step-by-step guidance. It defines the required data models and modeling languages and their bindings to XML [B2] and JSON [B7].

The purpose of this standard is to support the discovery, retrieval, transfer, and execution of AR-enabled learning content, thereby facilitating the creation of repositories and online marketplaces.

The standard supports reuse and repurposing of existing learning content in 'mixed' experiences that combine real-world learner guidance with traditional media such as instructional video material or existing web applications and widgets.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

[B3] Packaged Web Apps (Widgets), https://www.w3.org/TR/widgets/

3. Definitions

3.1 Learning Experience: The cognitive and sensory-motoric effect on the user from performing a Learning Activity in a particular Workplace.

3.2 Activity: Activity is the execution of a planned workflow following a specific process, leading typically from beginning to end, regardless of whether or not the anticipated learning outcome is achieved.

3.3 Learning Activity: A learning activity is an activity (see above) that motivates the development of competence, i.e., knowledge, skills, abilities, and other characteristics.

3.4 Augmented Reality: Augmented Reality refers to enhancing human perception with digital assets to create a new experience including, but not restricted to, enhancing human vision by combining natural with digital images.
3.5 AR Training System: An AR training system is a software application that allows trainees to re-enact predefined learning activities in a given workplace, but can also include authoring functionality to facilitate the creation of new learning activities.

3.6 AR Tracking subsystem: A computer vision component of the AR Training System that detects AR markers, AR image targets, and AR anchors in a room scan.

3.7 Experience API (xAPI): The xAPI is a Learning Analytics API, standardizing communication with a learning record store for logging learner performance. Any AR training system can optionally use the xAPI to keep record of the users’ learning activities, logged in triplet form of (actor, verb, object) statements. If xAPI logging is supported, xAPI statement queries can be used in the if-rules to control the activity.

3.8 Sensor: A sensor is a device that detects or measures physical characteristics and communicates the data generated digitally, such as a heart rate sensor, a gyroscope, or an Internet-of-things sensor that signals whether a specific button has been pressed during operation of a manufacturing machine.

3.9 Workplace: A workplace is a physical environment in which users learn by performing tasks with real world and virtual objects.

3.10 Augmentation: An augmentation is the digital representation of effector outputs that serve to stimulate the sensory experience of the user, which include outputs to visual, audio, haptic, or other modalities.

3.11 Trigger: The trigger is a mechanism firing an event on completion of an action step that allows the statements in the exit section of the action step to be released.

3.12 Validation constraint: A validation constraint checks observed user behavior, recorded in experience API endpoint, in order to control the flow of actions. It includes a query and the corresponding rules on how to react to results.

3.13 Detectable: Detectables are entities that link to fiducial markers, target feature models, or other sensor state properties providing input to computer vision and other sensor processing systems. Detectables have a unique identity and link to data enabling their tracking with the help of the sensor processing system referenced.

3.14 Augmentation primitive: Augmentation primitives define the types of annotations available in the AR training system. They include, but are not limited to, audio, video, images, animations, labels, and vibrotactile patterns.

3.15 Predicate: Predicates are reusable instructional augmentations, configuring a specific augmentation primitive for its use in activities. The set of predicates defined in a workplace model is a domain-specific language for instruction in the workplace under consideration, typically including all required verbs of handling and movement and their visual overlay animations signifying them.

3.16 Safety signs: A graphical symbol that indicates a potential hazard, obstacle, or requirement relating to the identification and management of risks in the workplace.

4. Learning in Activities and Workplaces

Descriptions of learning experiences consist of two types of data. Task-dependent Learning Activities, the first type, contain descriptions of how to interact with real and virtual elements in the sequence of actions. Workplace environment and context descriptions, the second type, consist of data concerning which real and virtual elements exist and are featured within the experience.
The definition of an activity mix for learning by experience includes orchestration of user interaction across multiple devices. Furthermore, it integrates the tracking of and responses to user interaction across devices and sensors. Optionally, if-rules can be defined and evaluated as part of the activity to test for user achievements or action outcomes using data from one or more software and/or hardware sensors.

The activity modeling language (activityML) in this standard is a domain-specific language used to describe the activity mix. The workplace modeling language (workplaceML) in this standard complements the activity representation language to realize interoperability of applications interpreting activityML. It is a domain-specific language for describing the tangibles, configurables, and basic triggers of a workplace.

![Diagram](image)

**Figure 1. The two modeling languages of ARLEM.**

Activities are containers that include a sequence of action steps, progressing from the start to the end of the activity, and metadata that describes the overall activity in human- and machine-readable terms. Activities link to workplace models that define, amongst others, where an activity can take place. Each action step of an activity specifies:

- **Activation and/or deactivation of Augmentations:** Specification and configuration data of media files to play, render, or invoke in other ways, when entering or exiting the current action step.
- **Triggers:** Reference and configuration of the release mechanism instigating the state change from enter to exit.
- **Optional: Messages:** Communication with other devices handled by the user or other users. This can include control commands such as launching an action step.
- **Optional: If-Rules:** Specification of queries and logic for analyzing and reacting to observed user behavior, recorded in an experience API endpoint.

Workplaces describe the tangible things, persons, and places in a concrete workplace. Moreover, they describe software and hardware, with which interaction can take place, including specifying how specific apps can be launched or which particular delivery devices are supported by the workplace. Moreover, how to communicate with the sensors supported in the workplace is described here. An example of a sensor is a device measuring physical characteristics such as energy levels using an Internet-of-Things protocol. Additionally, workplace models define the possible detectables, overlay primitives, and warnings signs supported. An example of a detectable is a marker that can be detected using a computer vision system. An overlay primitive, for example, specifies, how the predicate ‘rotate’ can be visually depicted.

### 5. An Activity Modeling Language for Activity Scripts

Activity scripts hold one activity element each, which contains at least one action step or a sequence of actions steps as the only direct subordinate elements. Since the activity container element defines an
obligatory start action, a minimal activity must contain at least one action referenced in activity metadata as the action to start with.

Each action uses three elements to handle augmentations, and two elements to determine the action workflow:

- The optional *instruction* element contains a human-readable title and description to be displayed or read out to the trainee. This element is optional, allowing the creation of action steps that display only augmentations instead of cluttering the display with instructional prompts. It can be replaced by an *app* element in order to launch specific web apps or widgets instead.

- The mandatory *enter* and *exit* elements are responsible for controlling augmentations, messages between devices and to other users, and simple constraint validation (based on if-rules) of user behavior.

  - Both blocks can contain *activate* and *deactivate* commands, specifying which augmentations and/or actions to launch or remove, respectively. They also define where these augmentations are anchored or what they are attached to in the workplace.

  - Moreover, optional *messages* can specify communication with other devices and/or users.

  - Optional *if-rules* check observed user behavior, recorded in an experience API endpoint, with the help of a query URL and guide, based on the retrieved results, the branching to other actions.

- The mandatory *triggers* element contains at least one *trigger* entry, defining how the user initiates the action’s state change that executes the statements of the exit block and that moves on to the next action step or activity.

![Figure 2. Overview showing the element structure of an activity.](image-url)
NOTE—The sequence of actions is determined by the activate/deactivate statements and/or validation constraints in each action’s enter and exit blocks.

NOTE—The AR training system decides how title and/or description of the instruction element are rendered. For example, the system can read out title and/or description of the instruction, using text-to-speech, instead of displaying a dialogue message, or the display area where the instructions are shown can vary.

6. Workplace Modeling Language

The workplace model describes the environment in which the user re-enacts activities, including properties and configuration data of real and virtual resources with which the user interacts. Each workplace model contains a single workplace, which holds all other data.

Figure 3. Overview showing the element structure of a workplace.

There are four types of resources that are defined within each workplace element: tangibles, configurables, detectables, and augmentations. Thereby, tangibles distinguish three distinct types:
– Things refer to physical objects such as materials, tools, and/or machines.

– Persons define the people in the workplace with whom the user can interact. Note that the mbox attribute holds the identifier of that person that can also be used for xAPI logging statements, if a xAPI is supported by the AR training system.

– Places define locations within the workplace.

For each thing, person, and place, specific points of interest (POIs) can be defined that allow referencing particular parts. Positional offsets along x-, y-, and z-axes as well as x-, y-, and z-rotation must be relative to a default location on the object. If no default point of interest is defined, the default must be the center of the object.

NOTE—Computer vision (or other sensor-processing units) determine the physical location of an object. For example, the center of an object detected with the help of a fiducial marker will be the center of the marker in practice.

There are three types of configurables:

– The entries listed under Sensors define how to communicate with devices or hardware components of the AR Training System that detect or measure physical characteristics. There are various Internet protocols available for communicating with sensors (the mandatory MQTT [B4]; further also CoAP, AMQP, XMPP, OPC-UA, UDP, or TCP), which all have different levels of adoption in different industries. The sensor entries in the workplace model provide the configuration data required to connect to these sensors, define what data keys to listen for, and how to parse them. This allows triggers in the activity script to react to sensor data.

– Devices provide information on what hardware and software resources are available to re-enact a learning experience.

– Apps allow information on how to launch web apps, i.e. Open Social gadgets and widgets [B3]. Apps can be included in action statements of activities, then typically replacing any instruction element. Apps are web apps, gadgets, and widgets.

Detectables are used to provide configuration information for computer vision and other sensory processing systems, instructing them how to detect any of the tangibles introduced above.

Finally, augmentations collect configuration data how to render overlays. They refer to the following sets of elements: augmentation primitives, predicates, and warnings.

– The augmentation primitives enumerate the types of Augmented Reality annotations possible. They include animation, image, video, audio, and label. Thereby, the image primitive instructs the system to create a generic primitive that can be featured as predicate in activate statements in the activity script to overlay a 2D image with the file provided via URL. In analogy, audio and video provide primitives for loading audio and video files. The label primitive instructs the system to provide a generic for overlaying text annotations.

– Predicates are reusable instructional augmentations, configuring a specific augmentation primitive for its use in activities.

– Warnings define the graphical symbols used to provide health and safety instruction (‘safety signs’).

NOTE—The set of predicates in the workplace model is a domain-specific language for instruction and training in that particular workplace. The set typically includes all necessary verbs of handling and movement.
NOTE—Support for multiple devices is required, when an activity has user interaction with, for example, both smart
glasses and a tablet computer.

7. ActivityML base schema

Table 1 defines the base schema for the activity modeling language (activityML).
### Table 1 – ActivityML 1.0 base schema

<table>
<thead>
<tr>
<th>Nr</th>
<th>Name</th>
<th>Explanation</th>
<th>Value Space</th>
<th>Datatype (Max. Length)</th>
<th>Examples</th>
</tr>
</thead>
</table>
| 1  | activity | The activity contains a sequence of action steps, leading the user through the workflow needed to develop competence in the intended learning outcome. **NOTE**—There is only one single activity element per activity script. | Holds action elements only (see 1.1) One activity per script | | XML: `<activity>
   <activity id="assembly120"
   name="Assembly of a stowage rack"
   description="An extensive description"
   language="en"
   workplace="http://wekit.eu/workplace.xml"
   start="step1"
   >
   </activity>` |
| 1a | id | Unique identifier | Alphanumeric | Character String (100) | | JSON: `{ "id": "assembly120", "name": "Assembly of a stowage rack", "description": "An extensive description", "language": "en", "workplace": "http://wekit.eu/workplace.json", "start": "step1" }` |
| 1b | name | Human readable name of the activity | Alphanumeric | Character String (1000) | | |
| 1c | description | Human readable description of the activity | Alphanumeric | Character String (10000) | | |
| 1d | language | Specifies the language [B6] with which the activity expresses human-readable instructional prompts | ISO 639-1 language code | Character String (2) | | |
| 1e | workplace | Reference to the workplace model | workplace.id | URL (2000) | | |
| 1f | start | Reference to the action to start with | action.id (see 1.1a) | Character String (100) | | |
| 1.1 | action | Action elements describe the steps of the learning activity | One or more action elements | | XML: `<actions><action></action><actions>` JSON: "actions": [] |
| 1.1a | id | Unique identifier (unique within the activity), used for referencing | Alphanumeric | Character String (100) | id="step1" |
| 1.1b | viewport | The visual area in which any instruction contained in the action displayed. 

**NOTE**—This may be a fixed area of the display or a named surface of the room. | ‘actions’, ‘reactions’, ‘warnings’ | CharacterString (100) | viewport="actions" |
| 1.1c | type | Specifies the category of the action and is used for styling visual appearance | ‘actions’, ‘reactions’, ‘warnings’ | Character String (100) | type="actions" |
| 1.1d | [device] | Specify on which device to execute the action | device.id in the workplace model | Character String (100) | device="glasses1" |
| 1.1e | [location] | Optional: Refer to a particular place where the action happens | place.id in the workplace model | Character String (100) | location="hallway" |
| 1.1f | [predicate] | Optional: The action may specify what verb to log user interaction with via the xAPI [B1]. When no predicate is provided, logging must use the standard xAPI verb ‘launched’ for each action step. If xAPI is supported by the AR training system, then any predicate id used must correspond to the id of the according xAPI verb statement definition. 

**NOTE**—xAPI support is optional. | predicate.id in the workplace model | Character String (1000) | predicate="rotate" |

### 1.1 Instruction

| 1.1.1 | instruction | Provide human-readable directives and prompts on what to do in this particular action step |  |
| 1.1.1.1 | title | The headline of the instruction | Alphanumeric | Character String (100) |
| 1.1.1.2 | description | More detailed instruction | Alphanumeric | Character String (5000) |
### 1.1.2 enter

Determines which augmentations will be activated or deactivated, and which messages and validation rules will be executed, once the action step is launched.

<table>
<thead>
<tr>
<th>Holds activate, deactivate, message, and if containers (see 1.1.2.1 to 1.1.2.4b)</th>
</tr>
</thead>
</table>

#### 1.1.2a [removeself]

Optional: If set to true, the action triggers the execution of the exit loop once the activate, deactivate, message, and if statements contained here in `<enter>` have been evaluated. If not provided, it is evaluated the same way as if set to false.

- **true**, **false**
- **Boolean**

#### 1.1.3 exit

Determines which augmentations will be activated or deactivated, and which messages and validation rules will be executed, once the action step is exited.

#### 1.1.3a [removeself]

Optional: If set to true, the action itself is removed (most notably: removal of the instruction, see 1.1.1) before the statements of the exit block are evaluated. If not provided, it is evaluated the same way as if set to true.

### 1.1.2.1 activate

Statement to execute effector output: load and display an augmentation to the user.

#### 1.1.2.1a tangible

Reference to unique identifier of a tangible in the workplace model.

- **thing.id** or **place.id** or **person.id**
- **CharacterString (100)**

#### 1.1.2.1b type

Specify which augmentation type the id refers to.

- **'primitive'**, **'predicate'**, **'warning'**, **'action'**
- **CharacterString (100)**

---

*Copyright © 2019 IEEE. All rights reserved.*

*This is an unapproved IEEE Standards Draft, subject to change.*
### 1.1.2.1c

**id**

Instruct the AR training system to play or display the augmentation with the id provided (a predicate.id, primitive.id, or warning.id from the workplace model or an action.id from the activity model).

**NOTE**—To instruct the AR training system to display an ISO 7010 hazard or warning sign (for example: ‘W012 electricity hazard’ or ‘M004 wear eye protection’), there are two possibilities: If the activate statement is of type ‘warning’, the id=w012, and a tangible=’board1’, then the safety sign will be overlaid on board1. If there is no tangible target referenced, type=’warning’, and id=m004, then the warning sign will be displayed at a fixed position in the viewport ‘warnings’.

**NOTE**—To display the predicate augmentation for ‘rotate’ on top of ‘board1’, the statement must include type=’predicate’, id=’rotate’, and tangible=’board1’.

**NOTE**—To attach an audio recording, the statement features type=’primitive’, id=’audio’, with or without tangible target. To attach an image, type=’image’, url=’http://…’.

### 1.1.2.1e

**[poi]**

Optional: Specify the point of interest (poi) of the tangible where the augmentation will be displayed.

 poi.id of tangible’s poi in workplace model

### 1.1.2.1f

**[option]**

Optional: Configuration option for the augmentation

**NOTE**—A predicate ‘point’, for example, may support different directions (‘up’, ‘down’, ‘left’, ‘right’).

### 1.1.2.1g

**[viewport]**

Optional: Specify the area of the display where the augmentation is to be displayed.

‘actions’, ‘reactions’, ‘warnings’

<table>
<thead>
<tr>
<th>1.1.2.1c</th>
<th>id</th>
<th>Instruct the AR training system to play or display the augmentation with the id provided (a predicate.id, primitive.id, or warning.id from the workplace model or an action.id from the activity model).</th>
<th>predicate.id, primitive.id, warning.id, or action.id</th>
<th>CharacterString (100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.1e</td>
<td>poi</td>
<td>Optional: Specify the point of interest (poi) of the tangible where the augmentation will be displayed.</td>
<td>poi.id of tangible’s poi in workplace model</td>
<td>CharacterString (100)</td>
</tr>
<tr>
<td>1.1.2.1f</td>
<td>option</td>
<td>Optional: Configuration option for the augmentation</td>
<td></td>
<td>CharacterString (100)</td>
</tr>
<tr>
<td>1.1.2.1g</td>
<td>viewport</td>
<td>Optional: Specify the area of the display where the augmentation is to be displayed.</td>
<td>‘actions’, ‘reactions’, ‘warnings’</td>
<td>CharacterString (100)</td>
</tr>
<tr>
<td>1.1.2.1h</td>
<td>image/video:</td>
<td>Special case: type='primitive' and id is 'image' or 'video':</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>---------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[url]</td>
<td>The image and video primitives require a URL from which to load a JPEG or PNG image or MP4 video file</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE— URL is optional. If no URL is given, the system must look for a resource packaged with the app with the name of the tangible.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.1.2.1i</th>
<th>animation:</th>
<th>Special case: type='primitive' and id='animation':</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[url] [state]</td>
<td>The animation primitive allows to reference a URL from which to load an animation FBX file and move it to the key frame specified in 'state'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, if no state is specified, the animation must be loaded, but not displayed and can then be moved to a different animation state in a subsequent action statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE— In all cases, the URL attribute is optional. If no URL is given, the system must look for a resource packaged with the app with the name of the tangible.</td>
</tr>
</tbody>
</table>
### 1.1.2.1j
**label:**

- **text**

  - **Special case:** type='primitive', id='label', and 'text' attribute is provided:

    Specify which character string to display at a specific tangible’s default location or any of its specific points of interest (POI)

  - **Character String (100)**

    ```xml
    <activate
        id="board1"
        type="tangible"
        predicate="label"
        poi="left"
        text="This is the left side"
    />
    ```

### 1.1.2.1k
**label:**

- **sensor key [option]**

  - **Special case:** type='primitive' and id='label', and sensor+key attributes present:

    The label primitive can connect to a sensor with sensor.id to display data to display the value of variable key.

    Optional: When option is set to 'stream', the data must be read and displayed continuously

  - **sensor.id:**
    - **Character String (100)**

  - **sensor.key:**
    - **Character String (100)**

  - **option:**
    - **Character String (10)**

    ```xml
    <activate
        id="board1"
        type="tangible"
        predicate="label"
        poi="top"
        sensor="arduino"
        key="voltage"
        option="stream"
    />
    ```

### 1.1.2.1l
**audio:**

- **url**

  - **Special case:** type='primitive', id='audio', and url attribute is present:

    The audio primitive requires a URL from which to load a WAV or MP3 file


      ```xml
      <activate
        id="board1"
        type="tangible"
        predicate="audio"
        url="http://myurl.org/myaudio.wav"
      />
      ```

### 1.1.2.2
**deactivate**
<table>
<thead>
<tr>
<th>1.1.2.2a</th>
<th>tangible</th>
<th>Specify id of a tangible in the workplace model to remove all or specific augmentations attached to it. Use wildcard '*' to remove from all tangibles.</th>
<th>thing.id, place.id, or person.id</th>
<th>Character String (100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.2b</td>
<td>type</td>
<td>Specify which type of workplace element the id references. Use wildcard '*' for all types.</td>
<td>'primitive', 'predicate', 'warning', or 'action'</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.2c</td>
<td>id</td>
<td>Specify id of the augmentation to remove. Use Wildcard '*' for all augmentations.</td>
<td>predicate.id, primitive.id, warning.id, action.id, or '*'</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.2d</td>
<td>poi</td>
<td>Optional: Specify the point of interest (poi) from where to remove the augmentation.</td>
<td>poi.id of the tangible referenced in 1.1.2.2a</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.2e</td>
<td>viewport</td>
<td>Optional: Specify from which viewport to remove the augmentation.</td>
<td>'actions', 'reactions', 'warnings'</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.3</td>
<td>message</td>
<td>Specify message statements to allow sending control commands and communication from device to device and user to user.</td>
<td>Character String (10000)</td>
<td></td>
</tr>
<tr>
<td>1.1.2.3a</td>
<td>id</td>
<td>Reference to person, device, or sensor.</td>
<td>person.id or device.id or sensor.id</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.3b</td>
<td>target</td>
<td>Specify whether to send communication message to a person or control message to a device or sensor.</td>
<td>'person', 'device', 'sensor'</td>
<td>Character String (100)</td>
</tr>
<tr>
<td>1.1.2.3c</td>
<td>viewport</td>
<td>Optional: Specify a particular viewport in which to display the message.</td>
<td>'alerts', 'actions', 'reactions'</td>
<td>Character String (100)</td>
</tr>
</tbody>
</table>

Example:

```xml
<deactivate
  id="start"
  type="tangible"
  predicate="label"
  poi="default"
/>
```

```xml
<message
  target="person"
  id="jake"
  viewport="alerts"
  id="start"
  type="tangible"
  predicate="label"
  poi="default"
/>
```

Job is done!

```xml
<message
  target="sensor"
  id="22ffdac321"
  key="messages"
  launch="action15"
/>
```
<table>
<thead>
<tr>
<th>Section</th>
<th>Key</th>
<th>Description</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2.3d</td>
<td>[key]</td>
<td>Optional: specify which variable key (mqtt 'topic') to use for broadcasting the message contents</td>
<td>Alphanumeric</td>
<td>Hi Fridolin, please start with action 15! &lt;/message&gt;</td>
</tr>
<tr>
<td>1.1.2.3e</td>
<td>[launch]</td>
<td>Specify which action must be launched on the recipient device</td>
<td>action.id</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The AR training system must publish launch ids to other devices or sensors on the mqtt topic target + '/' + id.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2.4</td>
<td>if</td>
<td>Specify a rule for validating user behavior and other characteristics with defined queries to determine branching of the flow of action</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE— As xAPI get queries return exactly none, one, or several statements, this query can only be evaluated for whether it returns statements (or not). If min and/or max are provided, it must evaluate whether the number of statements returned falls into the interval min &lt;= number of results &lt;= max.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2.4a</td>
<td>url</td>
<td>The query URL (for statement retrieval via get), including all parameters</td>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>1.1.2.4b</td>
<td>then</td>
<td>Specify which action to trigger if the query yields one or more (or the specified number) of results</td>
<td>action.id</td>
<td></td>
</tr>
<tr>
<td>1.1.2.4c</td>
<td>else</td>
<td>Specify which action to trigger if the query yields no results</td>
<td>action.id</td>
<td></td>
</tr>
<tr>
<td>1.1.2.4d</td>
<td>[min]</td>
<td>Check whether there are at least min number of results</td>
<td>Integer</td>
<td>0 … 65535</td>
</tr>
<tr>
<td>1.1.2.4e</td>
<td>[max]</td>
<td>Check whether there are no more than max number of results</td>
<td>Integer</td>
<td>0 … 65535</td>
</tr>
<tr>
<td>1.1.4</td>
<td>triggers</td>
<td>Specifies which different triggers instigate the state change from enter to exit of an action</td>
<td>Holds trigger elements</td>
<td></td>
</tr>
<tr>
<td>1.1.4.1</td>
<td>trigger</td>
<td>The trigger is enabled after the statements on the enter stack have been evaluated, starting the execution of the stack of exit statements when released</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4.1a</td>
<td>id</td>
<td>Reference of the action, sensor, or tangible the trigger is listening to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4.1</td>
<td>mode</td>
<td>There are three different types of triggers, including the ones driven by the user ('click' and 'voice' triggers), the ones driven by visibility of tangibles ('detect' triggers), and the ones driven by sensors ('sensor' triggers). Voice triggers must react to the key word 'next'. Click triggers must offer a confirmation button ('checkbox') where the instruction's title and description are displayed to the user.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4.1c</td>
<td>[duration]</td>
<td>For 'detect' triggers, a duration for the gaze lock can to be specified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4.1d</td>
<td>[type]</td>
<td>Optional: Specify type of the entity it is sensitive to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4.1e</td>
<td>[viewport]</td>
<td>Optional: Specify area of the display</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
<trigger id="start" mode="click" type="action"
         viewport="actions"/>
</triggers>

<trigger mode="detect"
          id="board1"
          type="tangible"
          duration="3"/>
```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>operator</th>
</tr>
</thead>
</table>
| Special case: mode='sensor': Specify for which variable ‘key’ to look whether it reaches ‘value’, optionally comparing with a different ‘operator’. The unique key must be defined in the sensor section of the workplace model. Optional: Specify an operator for the comparing value read from value specified. NOTE— If no operator is specified, the equal operator must be assumed. NOTE— For between, the interval must be indicated by a pair of numbers, separated by a semi-colon ‘;’. | key: Character String (100) value: Character String (100) operator: ‘exceed’, ‘below’, ‘equal’, ‘between’ | <trigger mode="sensor" id="arduino" key="connectionA2B" value="1" />
<trigger mode="sensor" id="heartrate" key="rate" value="80" operator="exceed" />
7.1 Instruction

Action containers may include an instruction element in order to provide human-readable directives and prompts on what to do in this particular action step. This element is optional, to allow creating action steps that, e.g., only display image or 3D overlays instead of cluttering the primary display with instructional prompts. Instructions’ title elements contain a short summary of the to do, the description elements contain narrative text describing in more detail what the user is supposed to do.

If provided, the title must be listed in the list of action steps as well as directly when presenting the description to the user once the action step is activated. The list of action steps must provide a confirmation button, typically a checkbox, to allow the user to release ‘click’ triggers (see also Section 7.5).

```
Title of the activity
Title of the first action step ✓
Title of the second action step
Title of the third action step
```

Figure 4. Example list of actions including checkbox trigger and highlighted current step.

The AR training system must support both providing the list of actions (on demand) to the user in order to provide an overview of the activity, as well as delivering the instruction title and description for each action step to the user, once the action step is activated. This can be done, for example, in form of an ‘action card’ (see Figure 5) or as audio using text-to-speech. If an action step does not provide an instruction element, it must be listed as an empty entry in the list.

```
Title of the activity
Description of the activity. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

Figure 5. Example action card with checkbox trigger.

7.2 Entry, Exit, Triggers

To define the flow of action within the activity, each action step specifies what triggers the state change from entry to exit. Moreover, each step defines what must happen when the action is launched (executing
the specified statements in the ‘enter’ element) and what must happen when the trigger moves to the next action (or whatever the ‘exit’ statements define).

Figure 6. Conceptual model of associated processing behavior (example 1).

Augmentations must remain active until explicitly deactivated. For example, if an action step 1 activates an augmentation A and then moves on to the next action 2, the augmentation A must remain visible unless it was explicitly removed in the block of exit statements of step 1 or in the enter section of step 2. For an illustration of this processing behaviour see Figure 6.

NOTE—This ensures that animations with states activated one by one over multiple action steps do not flicker because of deactivation between steps.

7.3 Interacting with Tangibles

The workplace model (see below) defines things, places, and persons, and how they can be detected using the computer vision engine provided by an underlying AR system. The activity script describes merely, what to activate and deactivate in the entry and exit routines: this can, for example, activate or deactivate other actions or overlays once a tangible becomes visible and is detected.

7.4 Interacting with actions

Each action step defines which other action steps are activated (or deactivated) in the statements provided in its ‘enter’ and ‘exit’ sections. This provides sequencing information for the flow of the activity.

The AR system must allow multiple action steps to be launched in parallel from the exit section of an action step in order to support branching within the flow of the activity. Any such action steps launched in parallel effectively offer alternative choices. When the user makes her choice, she triggers the selected action, which launches the target action defined. This target action then can remove the alternative steps offered previously (see Figure 7). Such processing behavior implements multiple-choice branching.
Figure 7. Conceptual model of processing behavior for multiple choice branching.

For further notes on branching and other forms of flow control, see the section on if rules.

7.5 Triggers

Triggers define what induces the state change from after the action was launched (and all enter statements have been executed) to executing the exit statement stack. There are three different types of triggers, including the ones driven by the user (‘click’ triggers), the ones driven by the world of things (‘detect’ triggers), and the ones driven by sensors (‘sensor’ triggers).

The first type is a click trigger. They register an event hook with the necessary launch statements for the follow-up action to be summoned – and react to the user clicking (or air tapping) on the action step checkbox button near where the instruction’s title and description are displayed to the user.

The second type, the ‘detect’ triggers, react to the computer vision signaling to the AR training system that a specific tangible has been detected. When the user stares at the tangible for the specified duration, the trigger is released.

NOTE— It is recommended to indicate to the user the duration required for such gaze lock. For example, circle segments can be filled successively until the target duration is reached.

The third type, the ‘sensor’ triggers, pick up on the data sent from connected sensors (see Section 8.1.2 sensors). Any sensor id used in such a trigger, must be defined in the specified workplace model (see Section 8, Table 1, 1.4 sensors). The AR system must connect at the start of the activity to the sensor system. It must then in each action step that references a sensor trigger monitor the sensor data stream for the appearance of the specified variable keys and check whether the data received meets the trigger conditions.

7.6 Logging user activity using the Experience API (xAPI)

Launching of activities, their actions, and specific overlays (like warnings) can provide valuable insights to analytics, ranging from performance information about training success to ensuring accountability through providing evidence of competence.
Any AR training system may optionally communicate with a so-called Learning Record Store implementing the Experience API [B1] in order to perform logging of user activity. The activity’s action steps provide the semantics required to log an activity trace (a triplet statement of user, the verb listed as predicate, and tangible objects handled or acted upon). If a predicate is provided, it must be used to generate the according triplet statements for the Learning Record Store. When no predicate is provided in an action step of an activity, logging to the xAPI must use the standard xAPI verb ‘launched’.

NOTE—xAPI client libraries are available from [B5].

8. Workplace Modeling Language Base Schema

A workplace model contains information about the real and virtual environment in which the user interacts.
<table>
<thead>
<tr>
<th>Nr</th>
<th>Name</th>
<th>Explanation</th>
<th>Value Space</th>
<th>Datatype (Max. Length)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The workplace element contains all information about a physical environment in which users learn by performing tasks with real and virtual objects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>workplace</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1a</td>
<td>id</td>
<td>Unique identifier of the workplace</td>
<td>Alphanumeric</td>
<td>Character String(100)</td>
<td></td>
</tr>
<tr>
<td>1b</td>
<td>name</td>
<td>Human-readable descriptive name of the workplace</td>
<td>Alphanumeric</td>
<td>Character String(1000)</td>
<td></td>
</tr>
<tr>
<td>1c</td>
<td>origin</td>
<td>The id of a detectable that marks the world origin for the workplace All positions of all other detectables of type ‘anchor’ must be expressed relative to this world origin, so that this single detectable can be used to sync x-, y-, and z-coordinate offsets across devices and platforms. NOTE—Using an image target marker in a fixed position works well, such as, for example, a logo on the nose of a plane or a poster beside the entrance.</td>
<td>Alphanumeric</td>
<td>Character String (100)</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>things</td>
<td>The container object holds an array of definitions of the physical workplace objects such as tools, machines, furniture, or materials</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1</td>
<td>thing</td>
<td>The container element holds the definition of a single physical workplace object</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1a</td>
<td>id</td>
<td>Unique identifier of the thing</td>
<td>Alphanumeric</td>
<td>Character String(100)</td>
<td></td>
</tr>
<tr>
<td>1.1.1b</td>
<td>name</td>
<td>Human readable short description of the thing</td>
<td>Alphanumeric</td>
<td>Character String(1000)</td>
<td></td>
</tr>
</tbody>
</table>

```xml
<workplace
  id="ox"
  name="Oxford Brookes University"
  origin="001"
/>
```

```xml
<thing
  id="helicopter"
  name="AW 109"
  detectable="333"
/>
```
### 1.1.1d detectable

Specify by id which detectable (see 1.7) applies to this thing. 

**NOTE**—The most common types of detectables are image markers and world anchors.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric</td>
<td>Character String (100)</td>
</tr>
</tbody>
</table>

### 1.2 places

Places are tangibles. They refer to locations within the workplace.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>name</td>
<td>Human-readable short description of the place</td>
</tr>
<tr>
<td>detectable</td>
<td>Specify by id which detectable (see 1.7) must be used to recognize this place</td>
</tr>
</tbody>
</table>

### 1.3 persons

Persons refer to the people in the workplace with whom the user can interact.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>name</td>
<td>The individual’s name</td>
</tr>
<tr>
<td>twitter</td>
<td>The individual’s twitter handle</td>
</tr>
<tr>
<td>mbox</td>
<td>The identifier to be used for the xAPI</td>
</tr>
<tr>
<td>detectable</td>
<td>The id of the detectable with which this person can be recognized</td>
</tr>
<tr>
<td>persona</td>
<td>Identifier of the role or group membership of this person, so as to allow for identifying next best alternatives in case of unavailability as well as for the retrieval of additional info about, e.g., access rights</td>
</tr>
</tbody>
</table>

```xml
<place id="platform" name="Platform" detectable="006"/>
<person id="fridolin" name="Fridolin Wild" twitter="fwild" mbox="wild@brookes.ac.uk" detectable="002" persona="/user/learner"/>
```
### 1.1.1.1 poi

Each tangible thing, place, and person (1.1, 1.2, 1.3) by default has at least one point of interest (‘poi’) at the origin, with additional, optional points specifiable. If no default ‘poi’ is provided, the system must assume the geometric center to be the default location.

<table>
<thead>
<tr>
<th>1.1.1.1a</th>
<th>id</th>
<th>Unique identifier</th>
<th>Alphanumeric</th>
<th>Character String (100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1b</td>
<td>x-offset</td>
<td>The offset on the x-axis (in cm)</td>
<td>cm.mm: 0.00 – 1000.00</td>
<td>Float</td>
</tr>
<tr>
<td>1.1.1.1c</td>
<td>y-offset</td>
<td>The offset on the y-axis (in cm)</td>
<td>cm.mm: 0.00 – 1000.00</td>
<td>Float</td>
</tr>
<tr>
<td>1.1.1.1d</td>
<td>z-offset</td>
<td>The offset on the z-axis (in cm)</td>
<td>cm.mm: 0.00 – 1000.00</td>
<td>Float</td>
</tr>
<tr>
<td>1.1.1.1e</td>
<td>x-rotation</td>
<td>Pitch in Euler angles</td>
<td>degrees: 0.000 – 360.000</td>
<td>Float</td>
</tr>
<tr>
<td>1.1.1.1f</td>
<td>y-rotation</td>
<td>Yaw in Euler angles</td>
<td>degrees: 0.000 – 360.000</td>
<td>Float</td>
</tr>
<tr>
<td>1.1.1.1g</td>
<td>z-rotation</td>
<td>Roll in Euler angles</td>
<td>degrees: 0.000 – 360.000</td>
<td>Float</td>
</tr>
</tbody>
</table>

### 1.4 sensors

Sensors are hardware devices that detect or measure physical characteristics and communicates the data generated digitally.

<table>
<thead>
<tr>
<th>1.4.1</th>
<th>sensor</th>
<th>For each sensor, the workplace model describes how to connect and what data streams to subscribe to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.1a</td>
<td>id</td>
<td>Unique identifier</td>
</tr>
</tbody>
</table>

```xml
<pois>
  <poi id="default"
    x-offset="0"
    y-offset="0"
    z-offset="0"
    x-rotation="40"
    y-rotation="-40"
    z-rotation="0"
  />
</pois>

<sensor id="arduino"
  url="mqtt://wekit.eu:1883"
  username="wekit"
  password="****"
/>```
### 1.4.1b **url**

Uniform resource locator (including protocol and port) of the API endpoint for the Internet-of-Things communication. The AR training system must support mqtt [B4], but it can also support additional protocols.

```
{protocol://host.domain[:port]}
```

| Character String (1000) |

### 1.4.1c **username**

Authentication information

| Alphanumeric Character String (100) |

### 1.4.1d **password**

| Alphanumeric Character String (100) |

### 1.4.1e **data**

One or more data elements describe how to parse the data stream.

### 1.4.1e.1 **key**

The variable identifier in the data stream

| Alphanumeric Character String (100) |

### 1.4.1e.2 **[type]**

The data type of the variable

| ‘string’, ‘float’, ‘integer’, ‘boolean’ |

| Character String (50) |

### 1.5 **devices**

Devices are the AR hardware delivery systems that the AR training system can communicate with to render the learning experience for the user.

#### 1.5.1a **device**

The hardware device that can be used to deliver activities or parts of activities created in this workplace.

```
<device id="22ffdab321" type="hololens" name="Fridolin's Smart Glasses" owner="fridolin" url="mqtt://test.mosquitto.org:1883" username="" password="" />
```

| Character String (1000) |

#### 1.5.1b **id**

Unique identifier

| Alphanumeric Character String (100) |

#### 1.5.1c **type**

Type of device (e.g. ‘hololens’, ‘ipad’)

| Alphanumeric Character String (100) |

#### 1.5.1d **name**

Human-readable name of the device

| Alphanumeric Character String (1000) |

#### 1.5.1e **[owner]**

Optional: The person id of the owner of a device

<p>| Alphanumeric Character String (100) |</p>
<table>
<thead>
<tr>
<th>1.6</th>
<th>apps</th>
<th>The apps container element holds definitions of how to interface the AR training system with other apps and widgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.1</td>
<td>app</td>
<td>Widget or app</td>
</tr>
<tr>
<td>1.6.1a</td>
<td>type</td>
<td>The type of application: HTML widget, launch command, or app prefab</td>
</tr>
<tr>
<td>1.6.1b</td>
<td>id</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>1.6.1c</td>
<td>name</td>
<td>Human-readable short description</td>
</tr>
<tr>
<td>1.6.1d</td>
<td>url</td>
<td>The uniform resource locator of the manifest file, launch command, or download link for an app prefab asset</td>
</tr>
<tr>
<td>1.7</td>
<td>detectables</td>
<td>Detectables are entities that link to fiducial markers, target feature models, or other sensor state properties providing input to the computer vision system</td>
</tr>
<tr>
<td>1.7.1</td>
<td>detectable</td>
<td>A detectable instructs the system how to recognize tangibles (things, places, or people)</td>
</tr>
<tr>
<td>1.7.1a</td>
<td>id</td>
<td>Unique identifier (referenced in 1.1.1d, 1.2.1d or 1.3.1e)</td>
</tr>
</tbody>
</table>
1.7b  *sensor*  Optional: The type of AR tracking subsystem to be used for its detection, routing the configuration data (upon parsing of a workplace) to the correct detection engine  
If no sensor is specified, the AR system must use a default subsystem for markers and a default subsystem for world anchors.

1.7c  *type*  Can be either a marker or a workplace anchor relative to the workplace origin

1.7d  *url*  Optional: uniform resource locator pointing to asset bundle with additional data (e.g., data about point clouds of feature targets / about image targets)

1.8  **primitives**  Augmentation primitives define the fundamental types of annotations available in the AR training system

**NOTE**—The workplace model informs the AR training system, which of the primitives are supported in this workplace. Moreover, for each augmentation primitive, the workplace model defines default size and/or volume level.

<table>
<thead>
<tr>
<th>1.8.1</th>
<th><strong>primitive</strong></th>
<th>Definition of a particular type of augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8.1a</td>
<td><em>id</em></td>
<td>Unique identifier</td>
</tr>
<tr>
<td>1.8.1b</td>
<td><em>x-size</em></td>
<td>cm.mm: 0.01 – 1000.00</td>
</tr>
<tr>
<td>1.8.1c</td>
<td><em>y-size</em></td>
<td>cm.mm: 0.01 – 1000.00</td>
</tr>
<tr>
<td>1.8.1d</td>
<td><em>z-size</em></td>
<td>cm.mm: 0.01 – 1000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.8.1e</th>
<th><code>&lt;primitive</code></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id=&quot;animation&quot;</td>
<td><code>x-size=&quot;0.01&quot;</code> <code>y-size=&quot;0.01&quot;</code> <code>z-size=&quot;0.01&quot;</code> <code>/&gt;</code></td>
</tr>
<tr>
<td>1.8.1e</td>
<td>volume</td>
<td>Optional: A volume setting, used for auditory primitives</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1.9</td>
<td>predicates</td>
<td>Predicates are reusable instructional augmentations, configuring a specific augmentation primitive for its use in activities. &lt;p&gt;&lt;em&gt;NOTE— The set of predicates defined in a workplace model should form a domain-specific language for instruction in the workplace under consideration, typically including all required verbs of handling and movement and their visual overlay animations signifying them.&lt;/em&gt;&lt;/p&gt;</td>
</tr>
<tr>
<td>1.9.1</td>
<td>predicate</td>
<td>A particular verb &lt;p&gt;&lt;em&gt;NOTE— If xAPI is supported, each verb listed here must have a corresponding xAPI verb statement.&lt;/em&gt;&lt;/p&gt;</td>
</tr>
<tr>
<td>1.9.1a</td>
<td>id</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>1.9.1b</td>
<td>type</td>
<td>Is an instance of primitive.id (1.9.1a) &lt;p&gt;‘animation’, ‘image’, ‘video’, ‘audio’, ‘label’&lt;/p&gt;</td>
</tr>
<tr>
<td>1.9.1d</td>
<td>scale</td>
<td>A (normalized) scaling factor to be applied to the primitive, across all axes &lt;p&gt;cm.mm: 0.01 – 1000.00&lt;/p&gt;</td>
</tr>
<tr>
<td>1.9.1e</td>
<td>url</td>
<td>Uniform resource locator pointing to a downloadable file with the augmentation</td>
</tr>
<tr>
<td>1.10</td>
<td>warnings</td>
<td>Definition of the supported safety and hazard warnings and instructions &lt;p&gt;&lt;em&gt;NOTE— The AR system must support all warnings listed in [B8], see also [B9].&lt;/em&gt;&lt;/p&gt;</td>
</tr>
<tr>
<td>1.10.1</td>
<td>warning</td>
<td>A graphical symbol that indicates a potential hazard, obstacle, or requirement relating to the identification and management of risks in the workplace</td>
</tr>
<tr>
<td>1.10.1a</td>
<td>id</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>1.10.1b</td>
<td><strong>type</strong></td>
<td>Is an instance of primitive.id (1.9.1a)</td>
</tr>
<tr>
<td>1.10.1c</td>
<td><strong>symbol</strong></td>
<td>Name of the symbol prefab</td>
</tr>
<tr>
<td>1.10.1d</td>
<td><strong>scale</strong></td>
<td>A (normalized) scaling factor to be applied to the primitive, across all axes.</td>
</tr>
</tbody>
</table>
8.1 Tangibles, Configurables, and Augmentations

There are four different types of resources that are described in a workplace model: tangibles, configurables, detectables, and augmentations. Tangibles are things, places, and persons (anything that is ‘real’). Configurables group together all software and hardware information, such as sensors, devices and their styling, as well as apps. Detectables collect information on how to detect tangibles using computer vision. This includes fiducial markers, image targets, point clouds, or world anchors. Finally, augmentations group together information on which augmentation primitives, predicate augmentations, and safety signs are defined in this workplace and how they are signified with visual, audio, or haptic signals.

8.1.1 Tangibles: Things, Places, Persons

Activities consist of action steps, where users interact with the workplace. During each step, this may involve handling, movement, communication, or other forms of interaction with the things, places, or persons in that workplace that matter to the activity. Each tangible is defined with a unique identifier and a human readable name. Moreover, it links to a detectable (see 8.1.3) in order to specify how it can be discovered and identified by the AR training system automatically. Each tangible thing, place, or person has a default point of interest. If none is provided in the workplace model, the AR training system will assume the geometric center of its detectable to be the default location. Additional points of interest can be added, so that they can be referred to via the ‘poi’ attributes in the activity models.

The default point of interest of a tangible is relative to the location of its detectable. This can be the geometric center of the position of a marker (possibly moving around in space) or an anchor defined relative to the workplace origin. The workplace origin is typically configured using a single workplace calibration marker. All additional, non-default points of interest are defined relative to the default location. All anchors are specified relative to the workplace origin.

8.1.2 Configurables: Sensors, Devices, Apps

There are three types of configurables: sensors, devices, and apps. Devices provide information on what hardware is available to re-enact the learning experience. For example, there can be activities that require both smart glasses and a tablet computer to be used in interplay.

Apps define how to launch web apps such as Open Social gadgets and widgets [B3]. Apps can be included in action statements of activities, then typically replacing any instruction element.

Sensors are hardware devices that detect or measure physical characteristics and communicate the data generated digitally. For each sensor, the workplace model describes how to connect and what data streams to subscribe to.

8.1.3 Detectables: Markers, Image Targets, Point Clouds, World Anchors

The detectables are used to provide configuration information for computer vision and other sensory processing systems, instructing them how to actually detect tangibles.

8.1.4 Augmentations: Primitives, Predicates, Warnings

Augmentations in the workplace model refer to three different sets of elements: generic overlay primitives, predicates, and safety warnings. The overlay primitives provide configuration information for augmented
8.1.4.1 Primitives: 3D models/animations, images, videos, audio, labels

The AR training system must implement the following generic augmentation primitives to allow the dynamic inclusion of external resources in the activity models. The minimum set of augmentation primitives that must be supported include: 3D overlays and animations (`animation`), `image`, `video`, `audio`, and `text` `label`.

Animations, the first generic predicate primitive, can either be embedded already within the app, or they can be provided dynamically via a URL. For dynamic animations, the app can download either when launching the activity or on the fly, when the overlay is activated.

All 3D animations are animation sequences with states. Each animation sequence must be addressed via the `state` attribute. By default, if no state is specified, the animation must be loaded, but not displayed and can then be moved to a different animation state in a subsequent action statement.

For videos dynamically included from external resources, the URL attribute must convey the link to an MPEG-4, or equivalent, file.

Similarly, images dynamically included must point to the resource URL with the JPEG or PNG file.

The final generic primitive is `label`. Text labels must be placed at or near a specific tangible’s default location or any of its specific points of interest (POI).

8.1.4.2 Predicates

Predicates contain predicate primitives to define the basic vocabulary for handling and motion in a particular workplace. Selecting predicates for handling and movement will depend on the application area.

NOTE—For example for the manufacturing of helicopters, textiles, and furniture, the following verbs are a good starting point: point, assemble, close, cut, disassemble, drill, inspect, lift, locate, lower, lubricate, measure, open, pack, pick, place, paint, plug, rotate (clockwise), rotate (counter clockwise), screw, unfasten, unpack, unplug, unscrew. This list is certainly not complete, but contains key predicates that can be expressed directly with visual instruction.

Where necessary, additional parameters must be provided on instantiation in activate and/or deactivate statements (for example: option="up").

There are other, more generic primitives that every system must implement: forbid and allow. Forbid thereby signifies what the user is not allowed to touch, whereas allow can be used to highlight explicitly, what the user is expected to touch.

8.1.4.3 Warnings

The AR training system must implement an interface for displaying the ISO 7010 safety and hazard warnings [B8]. The supported warning signs must be specified in the according workplace model. The activity model then can specify, where needed, whether the safety sign is placed in space near the relevant location/object or into a fixed viewport area. To display a safety sign near a tangible, the activity model’s activate statement must reference a type='tangible' and the according tangible id in the id attribute. To show a safety sign in a fixed position of the display area, the type must be type='warning' and the id must be the id of the safety sign.
NOTE— As a general recommendation, in most situations, elements relevant to the safety of the person (e.g., ‘wear protective gloves’) should best be displayed in a fixed position of the viewport, whereas object- or location-related warnings (e.g., ‘do not touch’) should better be placed near, but not occluding the actual object.
Annex A

Informative

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

3. AR training system architecture

The distributed component architecture is depicted in Figure 1. It distinguishes seven different key components.

Three of these are server applications: the single sign on (SSO), the community platform, and the cloud repository. The WEKIT.one unified app is the smart glasses application running on the Microsoft Hololens. The Hololens app communicates via mqtt to a message broker, which dispatches messages between the Hololens app, the hardware Vest (an ESP32 arduino implementation) and the Sensor Processing Unit (SPU).

The Hololens application possesses some common components, most notably, the data model implementations for activity and workplace model. These two classes implement the ARLEM data model standard described above in Section 2.

For the recorder part, the Activity Controller takes care of managing data collection from the TaskStationController (operated by the TaskStationContextMenu) and the AnnotationController (collecting data from and managing the Annotations).

A SensorController dispatches data received from particular sensors (like the vestSensors) received from the broker.

For the player functionality, the ActivityManager is responsible for instantiating Triggers and Augmentations, the latter via the ObjectFactory. Moreover, it instantiates also the Workplace using a WorkplaceManager.
On the side of the Vest platform, each hardwired sensor is connected via a SensorWrapper (typically this is just a C-function) to the SensorDataHandling (the Arduino project's `loop()`). Data are communicated via wifi to the mqtt broker.

The broker again distributes the data to the SPU's DataProvider Apps subscribed to it, which then again feed into DataProcessing components to generate processed output – and manage local storage as well as upload via the RecordingInterface to the Cloud Repository (using the Storage API).

From the repository, data can be downloaded again to be played in the player part of the WEKIT.one unified.

### 3.1. WorkplaceManager of the Player

The Hololens application layer is configured using ARLEM files (with JSON binding). The workplace model is used for setting up all the things, persons, and places together with detectables and sensors. The activity model is used for expressing the augmentations to be revealed in each action step.

When the player is started, the Unity scene contains just the empty parent objects for different workplace object types. The WorkplaceManager parses the ARLEM workplace model file and creates the child objects described. Figure 2 shows an example of such configuration for a thing object and the result in player Unity scene. The same approach applies to all workplace object types.

![Figure 2. Left: Example thing configuration. Right: Thing configuration result in player Unity scene](image)

Workplace objects are linked to the physical 3D space via the detectable objects. Detectable objects can be either Vuforia targets (Figure 3 and Figure 4) or Spatial Anchors (Figure 5) used by the HoloLens’s spatial mapping.

In the trials, this feature is most visible when approaching the workplace calibration marker (see d2.5). Once the calibration marker is detected, all other fixed locations contained in the workplace are recalibrated relative to the calibration marker. This way, it is possible to change the location of the plane, the rover, or the caddy with the ultrasound machine, while moving the relative locations attached to it along with the marker.
By linking the things, persons, and places to detectables, they pass their transform values to object they are linked with, and they tell the linked object to either show or hide based on the detectable’s state. The position is combination of the position of the detectable feeding its transform to the linked parent, and then adjusting the linked object position in relation to the thing poi configuration.

![Detectable configuration for Vuforia image target](image)

**Figure 3. Detectable configuration for Vuforia image target**

![Vuforia image target for a device and the device itself](image)

**Figure 4. Vuforia image target for a device and the device itself**

![Detectable configuration for spatial anchor](image)

**Figure 5. Detectable configuration for spatial anchor**

### 3.2. ActivityManager of the Player

The ActivityManager parses the ARLEM activity model file and handles the activity flow based on the configuration. The activity JSON array contains a list of actions that contain the configuration of each
action steps (Figure 6). The main components of the configuration are the enter and exit loops with activation and deactivation in both, the triggers and the instruction.

```
"id": "step",
"idParent": "actions",
"type": "action",

"enter": {
  "message": [
    "activate": [
      {"type": "trigger",
       "predicate": "start",
       "poi": "assit",
       "text": "The voice command 'SHOW TASK LIST' to start, and 'NEXT' to move between steps. Re-align out-of-place visualizations by looking at the NEXT button.",
       "scale": 2.50}
    ],
    "activate": [],
    "exist": [],
  ],
  "activate": [],
  "exit": [],
  "nextSteps": [],
  "triggers": [
    {"mode": "click",
     "type": "action",
     "id": "start"
    },
    {"mode": "click",
     "type": "action",
     "id": "stop"
    }
  ],
  "interaction": {
    "title": "Welcome to NEXT MKT trial activity",
    "description": "Welcome to NEXT MKT trial. Here you will learn how to assemble a TIR rack.
  }
}
```

Figure 6. Configuration for a single action step

The enter loop is activated when an action step is activated. The enter activation loop then activates objects based on the configuration. The configuration defines what type of content should be activated, show should it be initialized and where the content should be placed.

The activation “id” field is mapped to a thing, person or place object defined in the workplace JSON file. This means that the “id” is a game object in the player Unity scene created by the workplace manager. The field “poi” defines in which point of interest inside the parenting thing, person or place object should the content be placed. Field “type” can be either “tangible”, “action” or “reaction”. Tangible types are for activated content and the “predicate” field defines the content type. The rest of the fields are used for configuring the content type as needed (Figure 7 and Figure 8). The deactivation loop deactivates content, actions or reactions. If the content is activated, it remains in the scene until it is deactivated. This way it is possible to easily have some warnings etc. active during the whole activity. The exit loop is activated when an action or reaction is deactivated.
The triggers are used for triggering the exit loop of the currently active action. The possible trigger types are click, voice and IoT. Click triggers enables the user to simply click the next card on the activity card stack or click the check box in the tasklist view to launch the trigger. The voice trigger enables the “next” voice command to launch the trigger. The IoT trigger can be used with various sensors to launch the trigger when a desired sensor state is achieved. To enable a sensible logic flow, the action exit activation loop should contain activation for the next action step.

3.3. Sensor Interface of the Player

The player supports real-time sensors that can be used for simply providing and visualizing data or to trigger action steps. Sensors are divided in three categories, which are human, environment, and device sensors. Human sensors can be linked to workplace person objects, environment to workplace place objects and device sensors to workplace thing objects. The human and environment sensors are used for
providing data to the main UI sensor display and the different limit values are configured in the workplace JSON file. This way the system can support e.g. different heart rate ranges of the users. Device sensor visualization is presented on the linked thing object. Also, the device sensor is configured with the workplace file, where the variable types, ranges and limits are defined. Figure 9 shows a device and human sensor display in HoloLens view. Supported sensor protocols are MQTT and UDP.

![Figure 9. Device and human sensor display showing the result of the configuration](image)

### 3.4. Processing Behaviour of the Player

The overall process starts from the selection and loading of an activity file. If the user is loading an activity from an URL, all the content is downloaded and stored on the local file system. Once the activity is loaded, it is added to the list of available actions. The user can then load an activity by simply clicking the button showing the title of the desired activity.

Once an activity is loaded, the workplace JSON defined in the activity JSON is parsed by the workplace manager. The parsing of workplace JSON creates all the content placeholder objects, detectables and sensors to the player scene. Once the workplace JSON is parsed, the workplace manager informs the activity manager that everything is ready. This brings out the main UI window with the activity overview view active.

If the workplace file contains detectables that rely on the spatial anchors and the anchors are not yet in the device specific anchor database, the system informs that the anchors has to be calibrated first. This is done by using a predefined calibration target. The workplace JSON detectable configuration for the anchor types should contain offset values that are calculated from the origin of this calibration target. This means that there should be a predefined fixed location used for the calibration target and every time the calibration is done, the marker should be in exactly the same place. The calibration routine picks up the location of the calibration marker in the HoloLens coordinate system and creates the anchors to the positions defined by the workplace JSON offset definitions. The calibration needs to be done only once per device, since the created anchors are stored on the device anchor database that is persistent. By using this calibration process, it is possible to replay activities on different HoloLenses that were not used for recording the content.

When the user starts the activity by clicking the activity start button or by using the voice command “start” the action step defined as the starting action in the activity JSON file is activated. This activates the enter activation loop of this starting action. If there are content activations defined in this step, all the content is instantiated on the scene based on the activate configuration. The configuration tells where this content
should be placed and the content is instantiated as a child of the target object and point of interest. The transform of this target is updated by the detectable object linked to the target object.

### 3.5. ActivityController of the Recorder

TaskStations are created using a double tap while the gaze pointer is hitting the spatial map. Once created, air-tapping a task station (TS) will open the TaskStationContextMenu. Tapping a menu item will create a copy of the object, though distinguishable by a ring surrounding it (menu item = no ring).

Each annotation has its own menu that will manage the content stored in each. Task stations and annotations also have configuration menus (containing object-level functionality, such as delete and copy), which will have their functionality set in the top-level manager objects (in Unity).

The component structure follows a model-view-controller approach, see Table 2.

<table>
<thead>
<tr>
<th>Model Element (ARLEM)</th>
<th>View Component</th>
<th>Controller</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>My Activity</strong></td>
<td>ActivityModel (None)</td>
<td>ActivityController.cs</td>
<td>Create Task Stations, Open Task Station Menus</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Task Station (prefab)</td>
<td>TaskStationController.cs</td>
<td>(De)activate Task Stations, Create Annotations</td>
</tr>
<tr>
<td><strong>EnterExit/ ToggleObject</strong></td>
<td>Annotation (prefab)</td>
<td>AnnotationController.cs</td>
<td>(De)activate Annotations, Invoke Functions</td>
</tr>
</tbody>
</table>

Table 2. Model-View-Controller pattern.

The controllers are the interfaces that communicate to the data handling classes as well as dealing with changes to the views.

**ActivityController.cs**

This script is attached to the ActivityModel gameobject in the root of the 'Recorder' scene. It requires input to the following public variables:

1. MyWAM: Link to the World Anchor Manager, a component of the 'HoloManager' gameobject (same scene).
2. World Origin Marker Prefab: this is the visualisation of where the person started the app. Should not be null.
3. Task Station Prefab: the visualisation to use for all task stations.
4. Debug Text: this is a text object found in the 'HUD' root gameobject. For development, though can be used to display any information that should be user-locked.

**TaskStationController.cs**

This controller is attached to the Task Station prefab. Every task station has its own controller. A number of elements are accessible through the Unity inspector:
• Gaze focus colour and magnification - the change in colour and size when the gaze pointer is hovering over the task station.
• Menu Animation Settings - this sets set the speed of menu openings and the distance of the menu items (radially) from the centre object.

**AnnotationController.cs**

Every created annotation has a controller, which acts as a go-between for the various functions associated with capturing, replaying and storing data. The process of creating and modifying annotation types is explained below.

### 3.6. Annotations

Each annotation has a controller that handles common controls (e.g. 'remove annotation) and a unique element (script) that handles the functions specific to that annotation.

<table>
<thead>
<tr>
<th>Annotation Controller (Script)</th>
<th>AnnotationController</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config Menu Item Prefabs</td>
<td></td>
</tr>
<tr>
<td>Context Menu</td>
<td></td>
</tr>
<tr>
<td>Editor And Export</td>
<td></td>
</tr>
<tr>
<td>Stop Recording</td>
<td></td>
</tr>
<tr>
<td>Script</td>
<td></td>
</tr>
<tr>
<td>AudioAnnotation (Script)</td>
<td>AudioAnnotation</td>
</tr>
<tr>
<td>Record Audio</td>
<td>recordAudioPrefab</td>
</tr>
<tr>
<td>Menu Item Name</td>
<td></td>
</tr>
<tr>
<td>Menu Item Prefab</td>
<td></td>
</tr>
<tr>
<td>Is Visible Initially</td>
<td></td>
</tr>
<tr>
<td>Function To Call ()</td>
<td></td>
</tr>
</tbody>
</table>

The code functions for each annotation (e.g. start audio recording, replay video recording etc.) are linked to prefabs, which will determine what is displayed when the user clicks an instance an annotation.

#### 3.6.1. Adding or Modifying Annotation Types

The list of available annotations is specified inside the task station prefab, which can be found in the 'WEKIT / Recorder / Resources' folder. Adding new annotation types is done by extending the array in the Unity inspector and dragging the new prefab in to the component.

When creating new annotation types, the 'AnnotationController' component should be added as a component in Unity. An annotation-specific script is then added and should inherit the 'AnnotationBase' class, allowing its context menu to be set in Unity.
Updating prefabs is done by simply dragging the new prefab into the component in place of the previous one. The new prefab should be put in the 'WEKIT / Recorder / Resources / Annotations' folder and, if updating, the obsolete prefab should be moved to another folder or deleted.

### 3.6.2. Giving an Annotation Functionality

In order for an annotation to be useful, it must have a visual component, or icon that represents a function (we are not using text menus). For example, the 'start recording' function could be represented by a red cube. This object must have a collider so it is interactable using air taps.

### 3.6.3. Changing the View

Each annotation prefab has an 'icon' child object that contains the main mesh renderer component and a 'ring' child object that is activated or deactivated depending on whether the prefab represents an instance of an annotation or a menu item, respectively. Currently, the 'ring' component is searched for by name, so should be present in any new prefabs created.

### 3.6.4. Linking view to a function in the code

The unique script for each annotation work as a library of functions. There is a class called 'AnnotationMenuItem' which contains 4 elements and is specified in the script 'Annotation Base', allowing it to be inherited by each of the unique annotations, which are:

- **Menu Item Name:** Human readable name of the menu item.
- **Menu Item Prefab:** The prefab, described above.
- **Is Visible Initially:** A boolean that determines whether a newly created annotation will show this option.
- **Function to Call:** This is where the link is made between the prefab and the function. Described below.

### 3.6.5. Adding a new link to a code function

This is done in four steps, all of which can be accomplished in Unity.

1. Add a new element to the 'Context Menu' array, in the inspector, which will, in most cases, duplicate the last one already in the list
2. Specify when to use this: pick 'Editor and Runtime' to enable testing of the code in Unity before compiling
3. Specify the file: This can be any script in the code project, though the convention here is to use the annotation script itself, so drag the component we are editing into this box (see above screenshot)
4. From the drop-down list of functions, choose the script (same as 3) and the function inside this that you wish to invoke when tapping.

Remember that, if the IsVisibleInitially property is false, the prefab must be activated ('gameObject.SetActive(true)') within some other function, else it will never be displayed.
4. Conclusion

This deliverable reports on the ARLEM content model standard and the system architecture.

The content model standard is included as it is presented to the p1589 working group of the IEEE standards association. In the elaboration of the standard from its previous version to the version presented here, the specification has changed fundamentally – both in terms of its presentation as well as with regards to the models. Amongst the changes introduced for the presentation, this includes the following:

- Added clarity by separating normative and informative sections
- Added precision in language through stringent use of ‘must’, ‘may’, and ‘should’
- Added definitions for all key terms
- Added normative definitions for all elements and attributes, specifically removed the style of ‘introducing by example’
- Added value spaces and data type specification
- Separated examples from normative definitions

Amongst the changes introduced to improve the models, this included the following:

- Removed confusing dependency to HTML and CDATA markup
- Removed URN definitions (potentially to be done properly and re-introduced in a version 2)
- Improved the xAPI connection
- Improved the interface to ISO warning signs
- Improved the sensor interfaces (with normative support for mqtt)
- Introduced conceptual models of processing behaviour
- Introduced conceptual models for standard dialogues
- Improved if-rules for xAPI interface
- Improved message handling and sensor communication
- Improved anchor handling and relative locations/points of interest
- Improved sizing

More detail about these changes can be found in deliverable d8.3.

The architecture is described in overview first, then more details are added for specific components, explaining how they are implemented – in order to allow others to learn from our implementation.
WEKIT project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 687669. [http://wekit.eu/](http://wekit.eu/)