ISO/IEC JTC 1/SC 24/WG 9 **N 116**

**ISO/IEC JTC 1/SC 24/WG 9 "Augmented reality continuum concepts and reference model"**
Convenorship: **KATS**
Convenor: **Kim Gerard Jounghyun Mr**

# Information Model for Mixed and Augmented Reality (MAR) Contents Part 3: Live actor and entity (Presentation)

| Document type | Related content | Document date | Expected action |
|---|---|---|---|
| Meeting / Presentation | Meeting: VIRTUAL 21 Jul 2021 | 2021-11-03 | |

# Information Model for Mixed and Augmented Reality (MAR) Contents Part 3: Live Actor and Entity

ISO/IEC JTC1 SC24 Plenary Meeting

July. 21, 2021

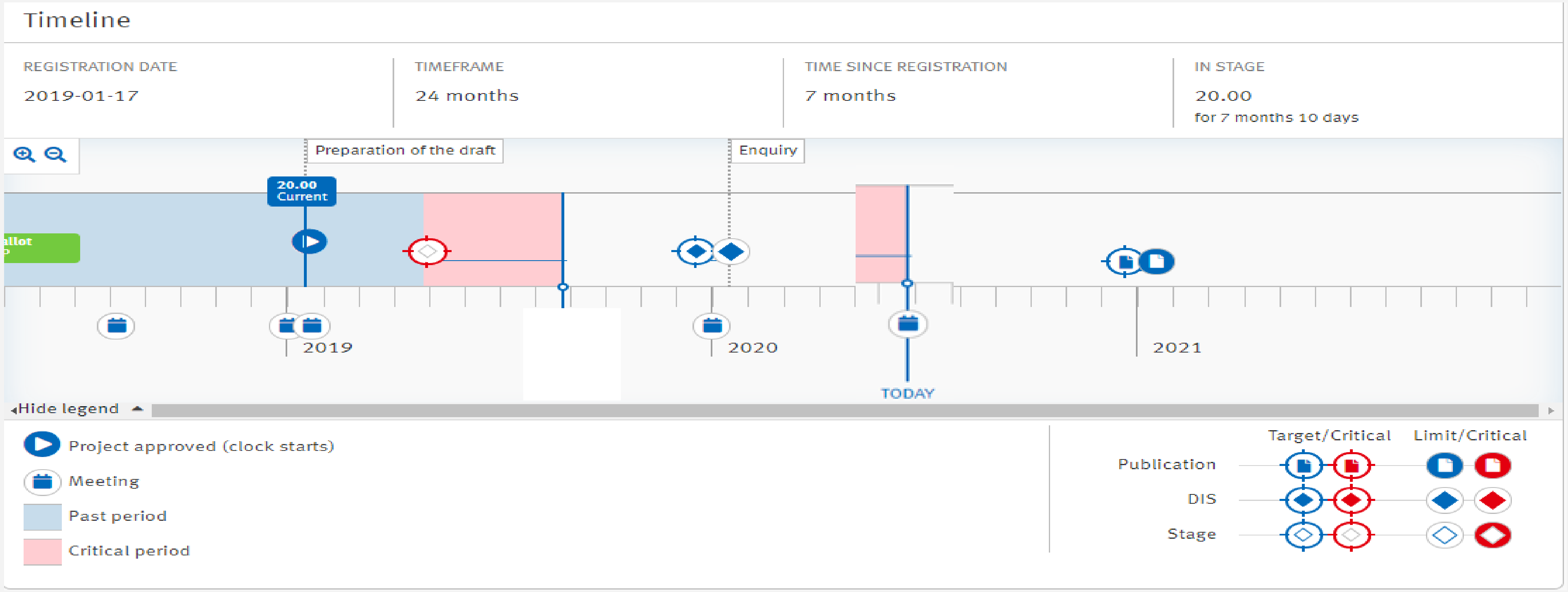
Kwan-Hee Yoo(Chungbuk National University, Korea)

# Status of this issue

| Purpose | Title | ISO NUMBER & Stage |
| --- | --- | --- |
| Propose concept and architecture for representing a live actor and entity in MAR | **Information technology — Computer graphics, image processing and environmental representation — Live Actor and Entity Representation in Mixed and Augmented Reality** | ISO/IEC JTC1 IS 18040 |
| Propose nodes of data structures for implementing LAE system in MAR | **Information technology — Computer graphics, image processing and environmental representation — Information Model for Live Actor and Entity in Mixed and Augmented Reality** | ISO IEC NP 23490 |
| | | |

# On ISO/IEC 23490

ISO/IEC 23490 Information technology — Computer graphics, image processing and environmental representation — Information Model for Live Actor and Entity in Mixed and Augmented Reality




Cancelled

# Future Direction

Information model for LAE
is tightly related to information model for MAR

Propose new NP/CD as soon as possible

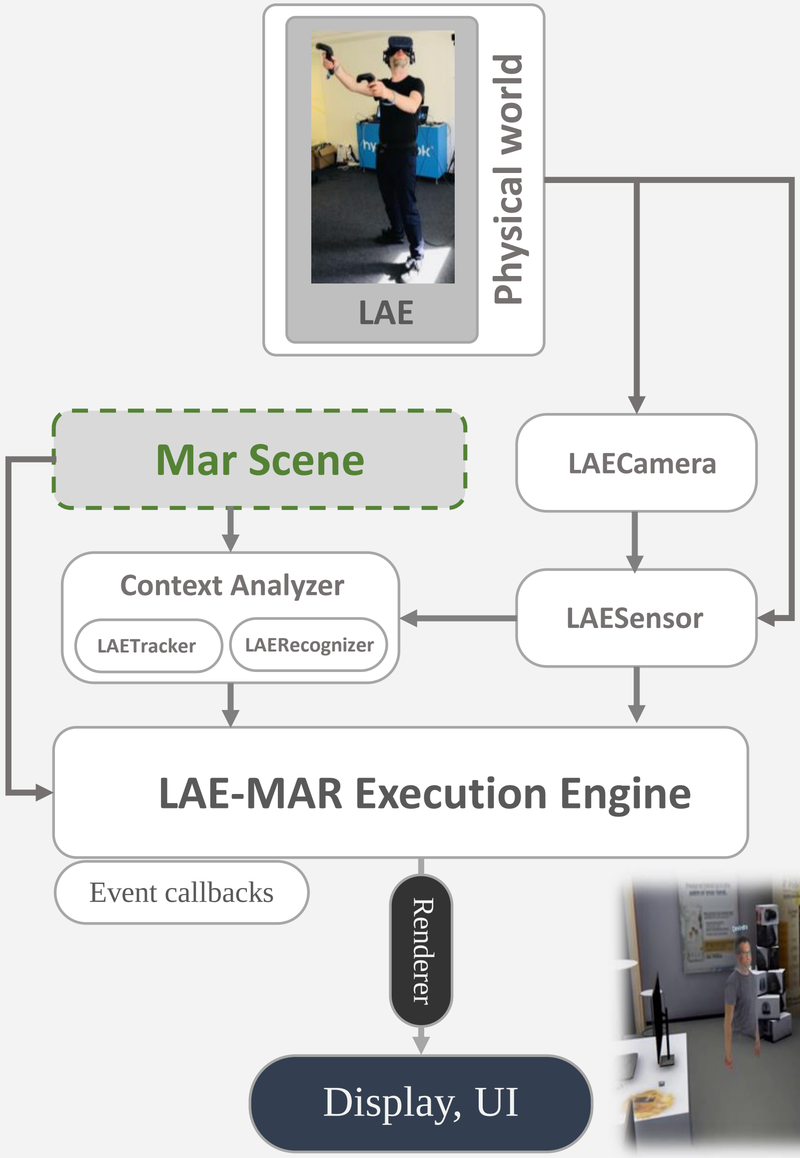
ISO/IEC 3721-3 Information technology — Computer graphics, image processing and environmental representation — Information Model for Mixed and Augmented Reality(MAR) Contents Part 3: Live actor and entity

# Scope

This document has an objective to extend the previous research project and existing standard for improving the LAE information models on Mixed and Augmented Reality scene/contents description. This extension is enhancing the capabilities of LAE-MAR system more reliable and putting system in advance stage of development.

- ✓ Improving LAE-MAR system working more effectively
- ✓ Allowing deep learning techniques to involve in system process
- ✓ Using Virtual Reality (VR) technology to extend the immersive experience to user
- ✓ Interaction ability between LAE and MAR models
- ✓ Standardization of using LAE-MAR system with defined structures of nodes for LAR-MAR

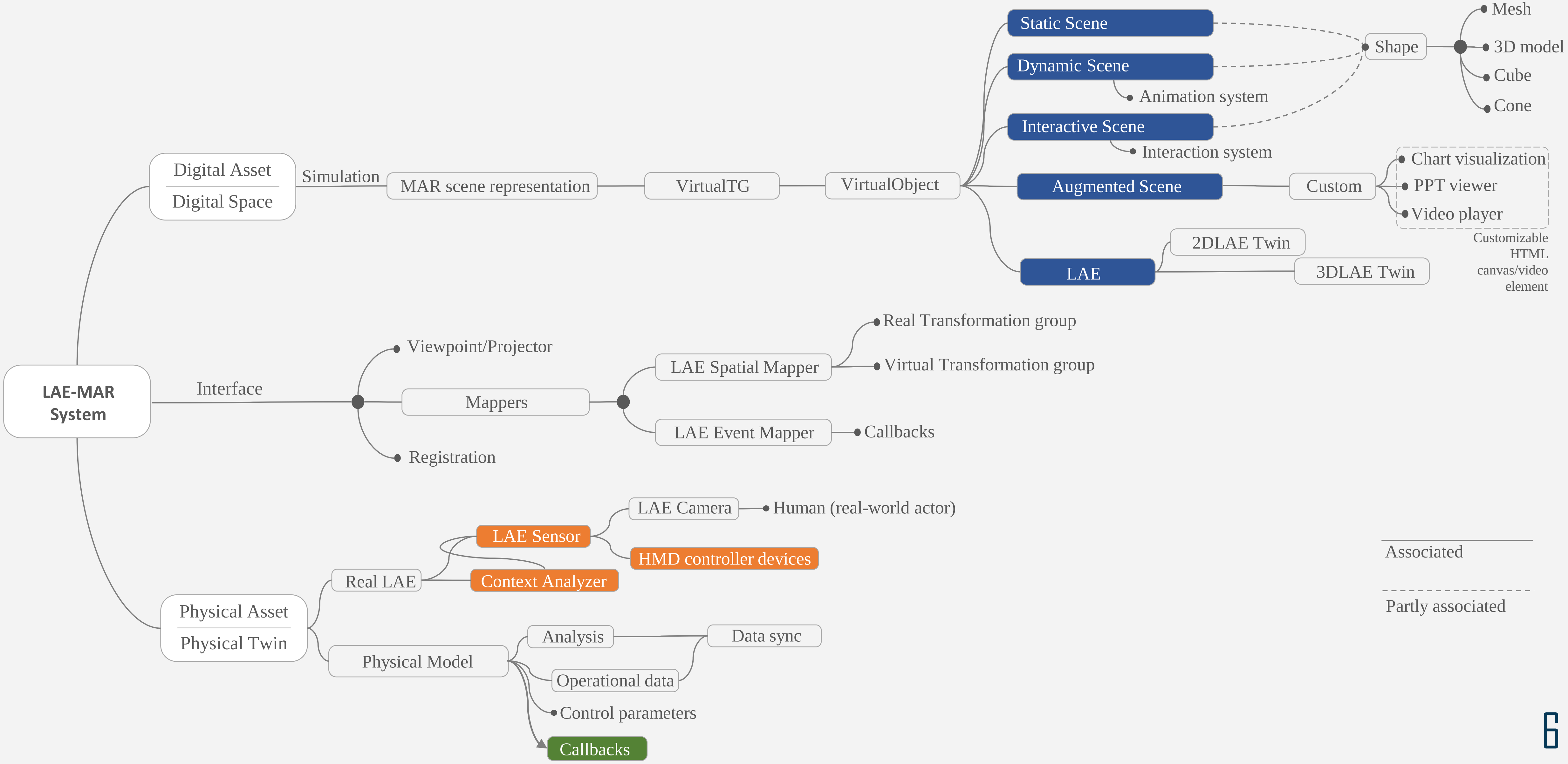
# Introduction




- **Live Actor and Entity Representation in Mixed and Augmented Reality (LAE-MAR)** is a system that is designed to handle a comprehensive representation of a live actor and entity (LAE) in a physical world by observing the information through various sensors into mixed and augmented reality (MAR). Especially, the system includes the architecture of embedding a **live actor and entity** into the virtual space.
- **LAE-MAR** is a representation of a living physical or real objects, such as a human being, animal, or bird, in the Mixed and Augmented Reality (MAR) content or system
- Performing **Live actor and entity** (LAE) in a virtual environment as natural as real-world activities
- The virtual actor can be reconstructed through machine learning techniques as a **2D or 3D model**. It can interact with embedded entities, which also attached with the event-callbacks
- Using the deep learning techniques to **analyze** the sensing data/information and to **simulate** the virtual scene
- Giving an immersive experience to the user by using **virtual reality** (VR) technology, which its perception is to experiencing physically present in a non-physical world

# DTw Visualization Integrated with LAE-MAR System

## The 3 parts of DTw visualization of Maturity Model based on LAE-MAR



# DTw Visualization Integrated with LAE-MAR System

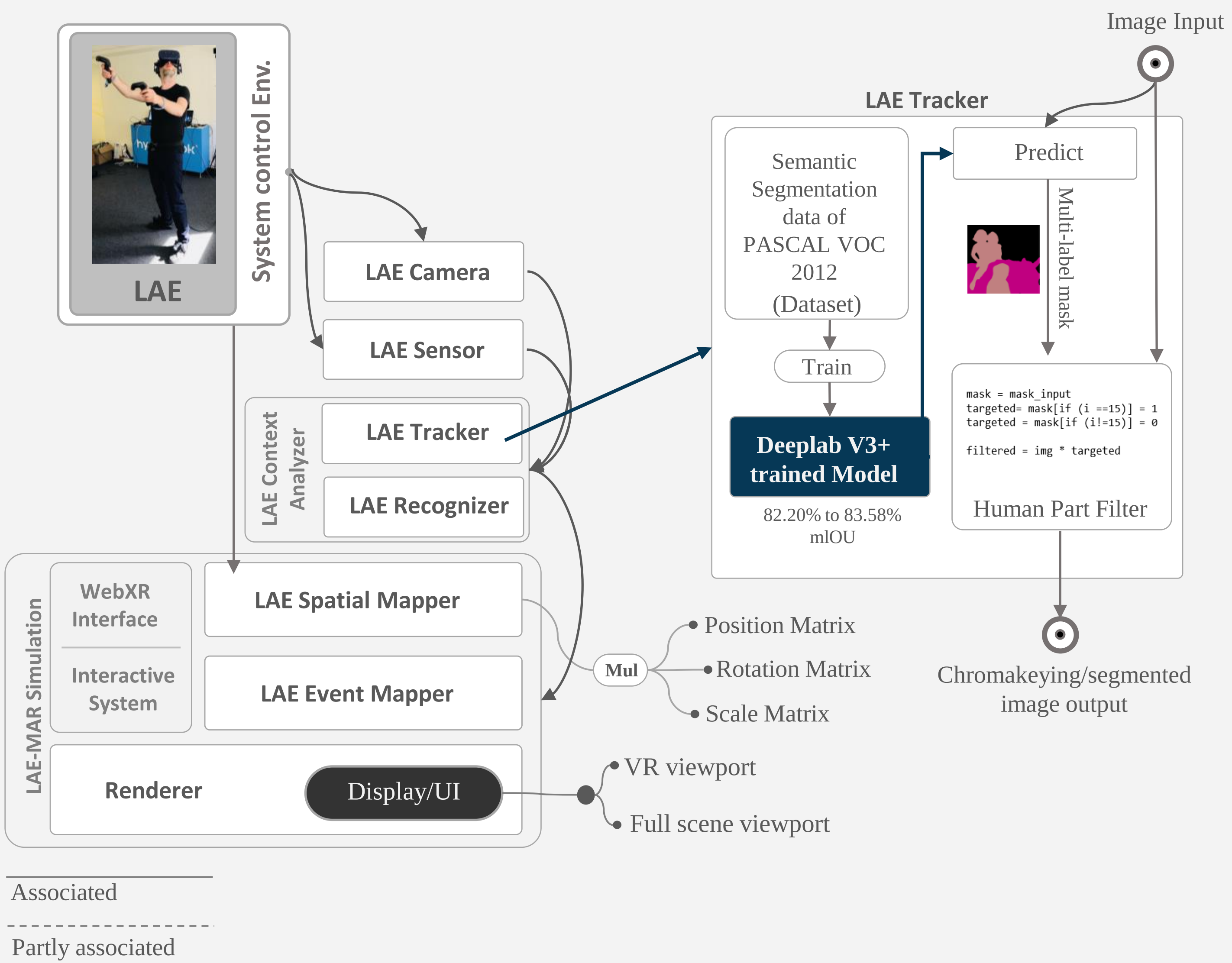
## The principle of LAE-MAR maturity models for DTw in LAE-MAR

| Name | Functionalities | LAE Role | System-design level | Example | Details |
|---|---|---|---|---|---|
| Static Scene | • Persistent, static, and initial data connection<br>• No models of behaviors and dynamics but process control logics applied | • Seeing/Moving around | Developer | • Virtual objects for environment design<br>• Independent models | • Support 3D model file (e.g. GLTF)<br>• Mesh<br>• Texture model |
| Dynamic Scene | • An object used to play animation on a loaded model<br>• timeAt: is useful when jumping to an exact time of animation<br>• deltaTime: is to slow down or fasten the animation | • Seeing/Moving around<br>• Observing working process | Developer | • Animated operational equipment<br>• Dynamic manufacturing engine<br>• A model relatively updated by sensor data | 3D model file (e.g. GLTF)<br>• Required pre-defined animation clips<br>Use tools to create<br>• Blender<br>• Maya<br>• Unity3D and etc. |
| Interactive Scene | • Synchronized and interactive operations among Digital Twins, but through human intervention for action<br>• Allowed user to interact with<br>• Fire the callback function on demands | • Seeing/Moving around<br>• Interactable (event handling) | Developer | • Interactable cube, cone<br>• Interactable 3d model<br>• Model roles can be repositioning and click-button event | • Support 3D model file (e.g. GLTF)<br>• Interaction for object relocation<br>• The interactive object does not depend on spatial mapper in the runtime. |
| Augmented Scene | • Usually, it is used as a panel to visualize the state of run-time data or data configuration<br>• An object model that is customizable in order to serve the operational requirement | • Seeing/Moving around<br>• Observing data visualization<br>• Interactable ( event handling) | Developer and System | • Built-in component (HTML canvas, HTML video)<br>• Ex. image viewer, chart graph, ppt viewer, video player, etc. | • This can handle models with their own functionalities. It mostly requires a developer to create/design the panel |

# LAE and LAE-MAR Maturity Models

# LAE2D in LAE-MAR

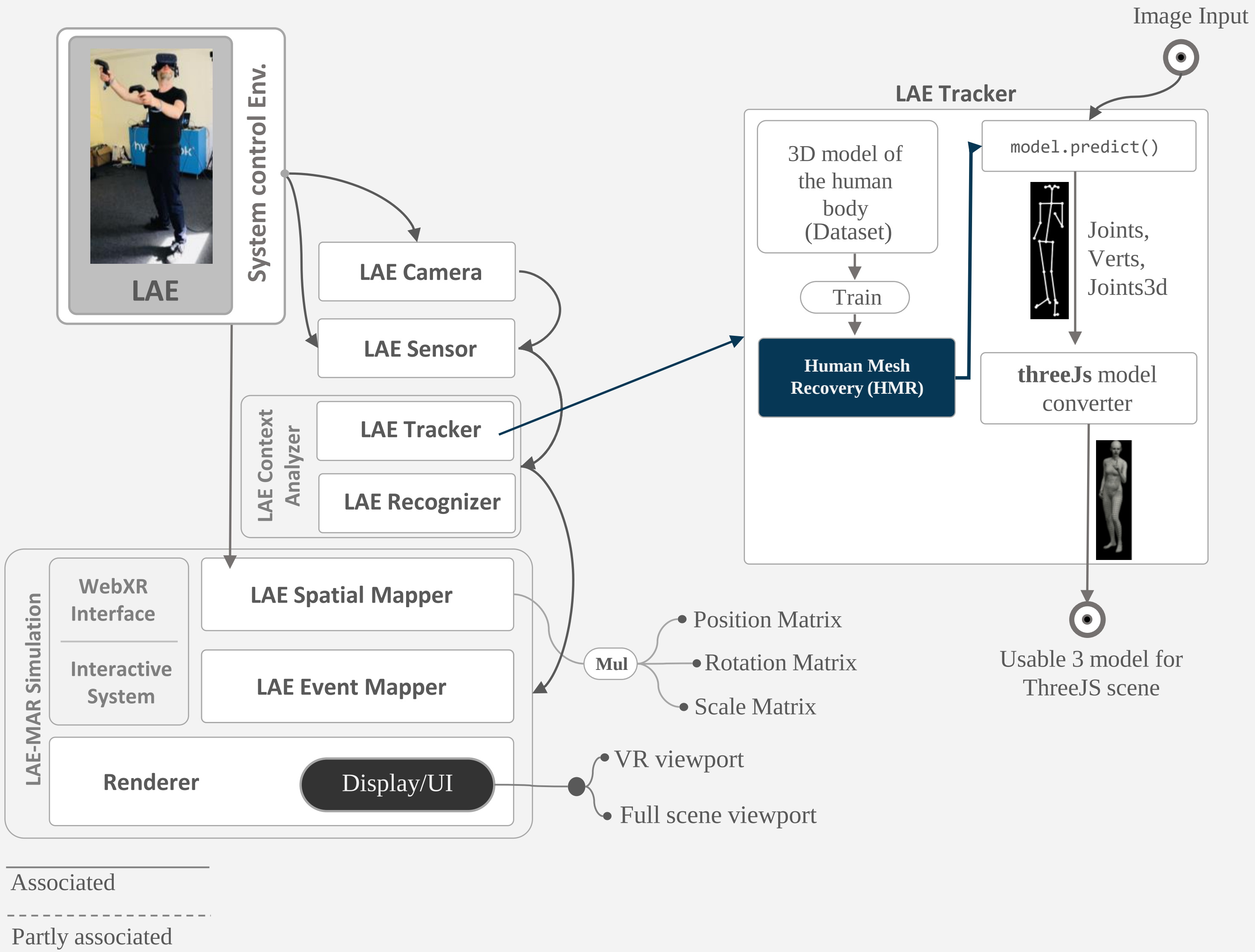
**Level 2** | **VTG::Virtual Object::2D LAE**




- ❖ A Live actor in LAE-MAR is a core component to represent the user as a virtual object. The 2D LAE is implemented as a human body digital twin in 2D form.
  - Deeplab is applied in the tracker module for image segmentation
  - The system also embed the WebVR for allowing the user to experience the digital world with an HMD device
  - Implementing virtual reality is a key to allow the user interaction in the system

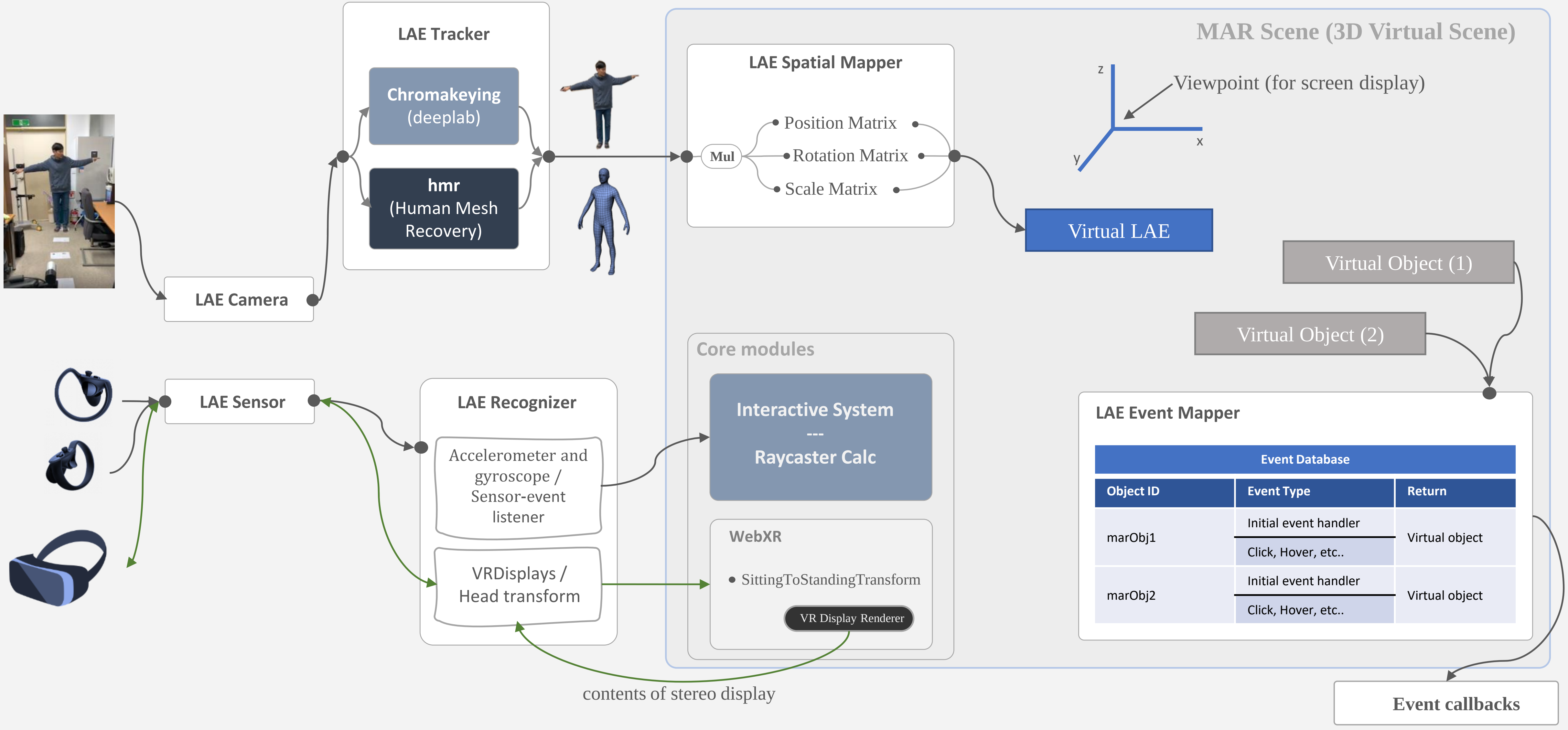
# LAE3D in LAE-MAR

Level 2 | VTG::Virtual Object::LAE3D




- LAE-MAR system provides various possibilities for representing the physical human as a digital live actor. Instead of a 2D live actor twin, we can construct a 3D live actor twin in real-time.
  - With Human Mesh Recovery (HMR), the system can predict the 3D body poses from 2D image input. The skeleton is mapped along the predicted poses to represent a physical human as a 3D form in the MAR scene.
  - In order to support the 3D model structure in ThreeJS, the HMR output must be translated accordingly to the ThreeJS object

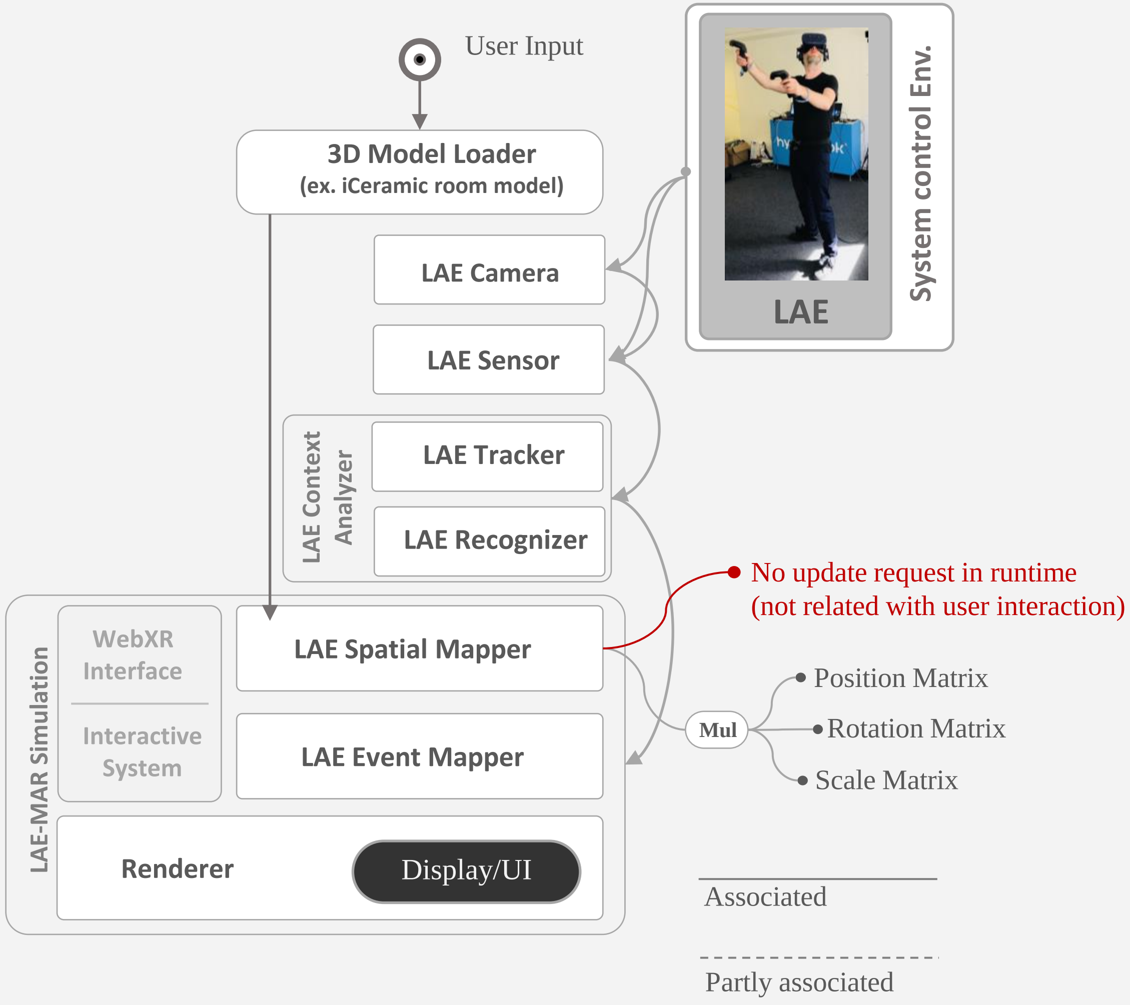
# LAE Spatial and Event Control Functions




| Object ID | Event Type | Return |
| --- | --- | --- |
| marObj1 | Initial event handler | Virtual object |
| | Click, Hover, etc.. | |
| marObj2 | Initial event handler | Virtual object |
| | Click, Hover, etc.. | |

# Static Scene Maturity Models in LAE-MAR

## VTG::Static Scene

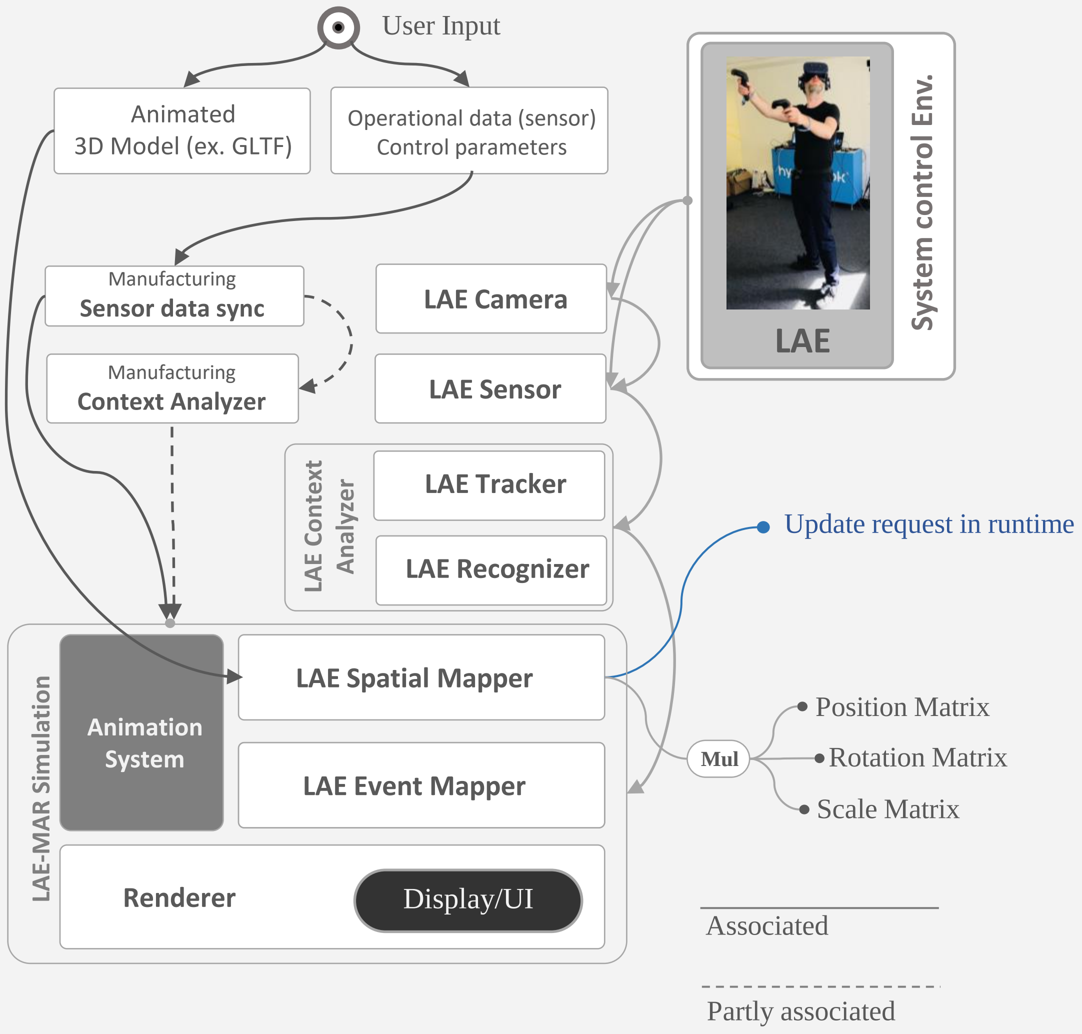



❖ This static model is purposely designed as a virtual object. Thus, this model respects the following roles:

- Load 3D model as an input
- Persistent, static, and initial data connection (Position, Scale, Rotation)
- In runtime, the loaded model never request for change/update
- After spatial mapping, the model is added to the LAE-MAR scene

# Dynamic Scene Maturity Models in LAE-MAR

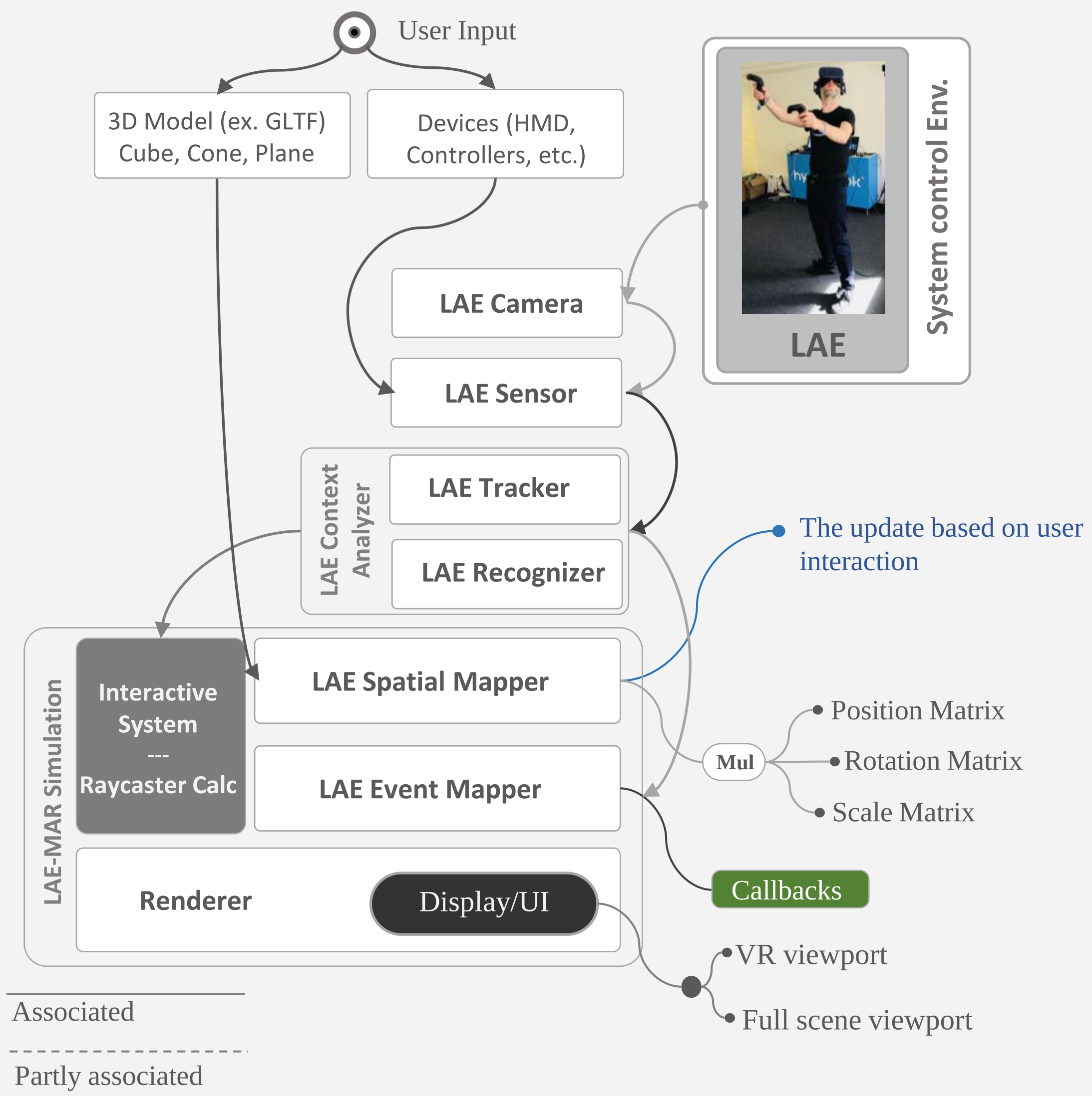
## VTG::Virtual Object::Dynamic Scene




- ❖ Main functionality to simulate the state of physical operating equipment to MAR Scene model as realistic as possible. The weight and time scales are used for simultaneous animations on the object.
  - Requires animated 3D objects (ex. GLTF)
  - The animation is updated based on control parameters or sensor data, which depends on logic definitions
  - Animation system takes part in controlling series of keyframes like play, pause, loop, or atTime

# Interactive Scene Maturity Models in LAE-MAR

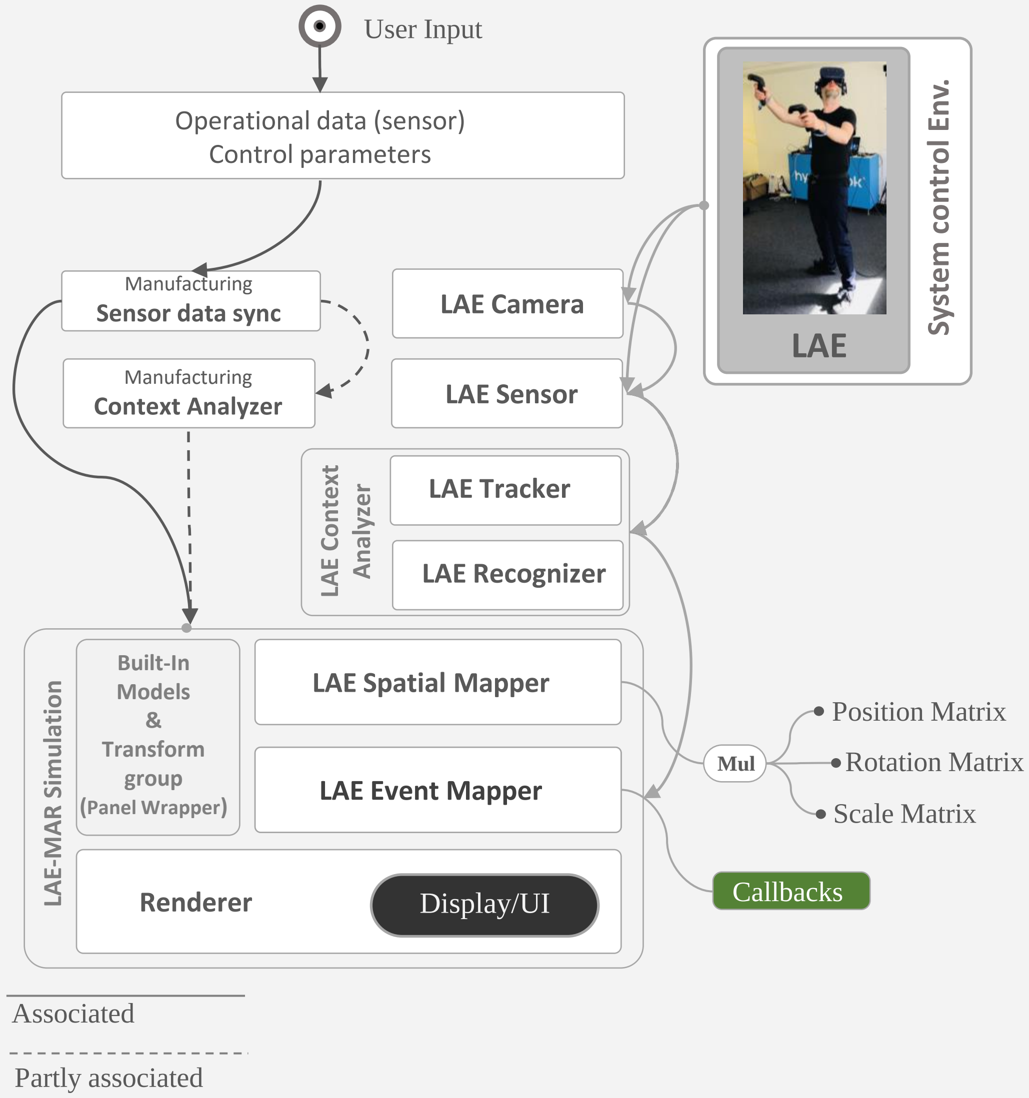
## VTG::Virtual Object::Interactive Scene




- ❖ Multiple objects in a MAR Scene are federated each other and perform mutual interactions for their cross-dependent operations. However, this interactive model merely receives the action from user interaction.
  - It requires LAE to perform actions (moving object, click, etc.)
  - Mainly focused on Event handling by the event mapper itself
  - Helpful in creating such a setting/configuration panel.

# Augmented Scene Maturity Models in LAE-MAR

## VTG::Virtual Object::Augmented Scene




❖ A particular object wrapper model can be customizable according to utilities or types of visualization:

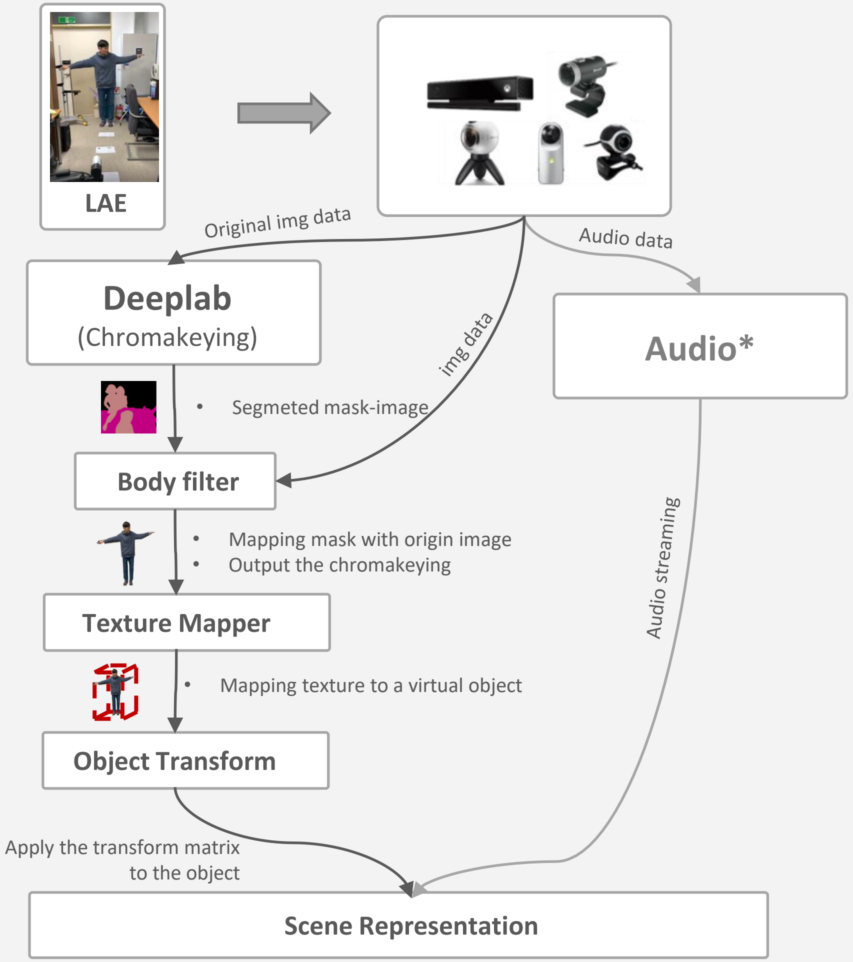
- Developer/System level for creating a built-in virtual, augmented object as a panel floating in space
- In runtime, augmented model listens to the user interaction for repositioning in spatial-mapper module
- If the augmented model contains an interactive object, it may listen to the event handler in the event-mapper module

# Implementation & Results

# LAE-MAR Node Definition

## LAE2DModel

- On the physical side, **LAE2DModel** composes of required sensors, camera, tracking technique, and recognizing technique to output as a streaming texture on a plane, which mapping with a virtual object designed to represent as a **live actor** in 3D space

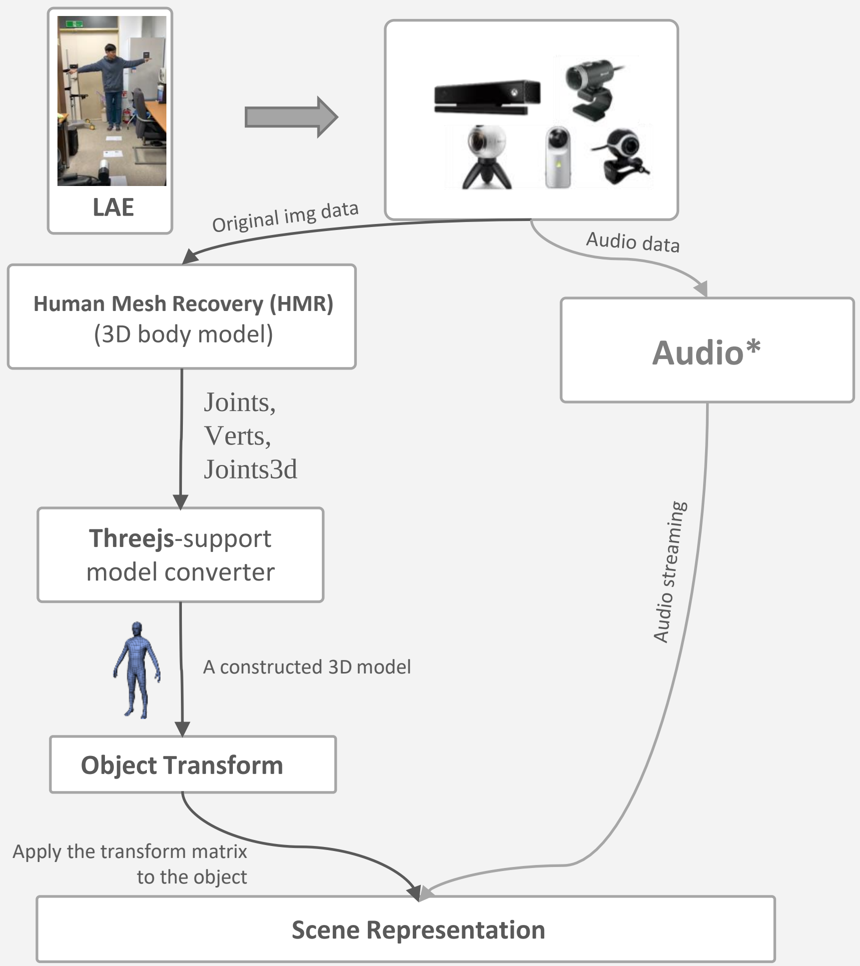



| LAEModel :: 2DLAEModel | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAE2DModel() | LAE2DModel | Protected | Constructor function |
| id | String | Public | Identifier |
| hidden | Boolean | Public | Hidden in a virtual scene |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| 3Dobject | 3Dobject | Public | A virtual object used for a virtual scene |
| type | String | Public | The default type is 2DLAE (2d-live-actor) |
| originalImg | Image | Private | A variable for the original image from the camera |
| maskImg | Image | Private | Mask image generated from Deeplab model |
| chromakeyingImg | Image | Private | Final output after filtering the live actor body |
| rotation | Vector3 | Public | A matrix for rotation in a scene |
| position | Vector3 | Public | A matrix for a position in a scene |
| scale | Vector3 | Public | A matrix for scale in a scene |
| processDeeplab() | Void | Private | Function to execute the model for image segmentation |
| processAudio() | Void | Private | Process the audio if it exists |
| bodyFilter() | Image | Private | Filter the body from original image with segmented mask |
| mappingTexture() | Void | Private | A function to map a virtual object with texture |
| setData() | Void | Public | Set the sequences of image/audio as the input |
| getImgData() | Void | Public | Access function for the segmented image |
| getAudioData() | Audio | Public | Access function for the audio |

# LAE-MAR Node Definition

## LAE3DModel

- On the physical side, **LAE3DModel** composes of required sensors, camera, tracking technique, and recognizing technique to output as a constructed 3D model, which mapping with a virtual object designed to represent as a **3D live actor**

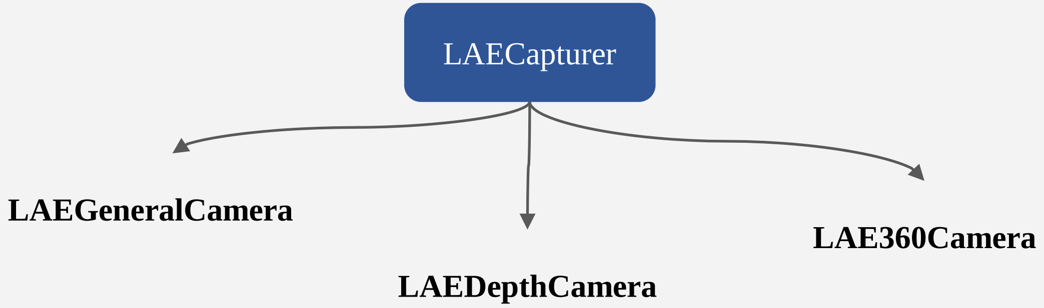


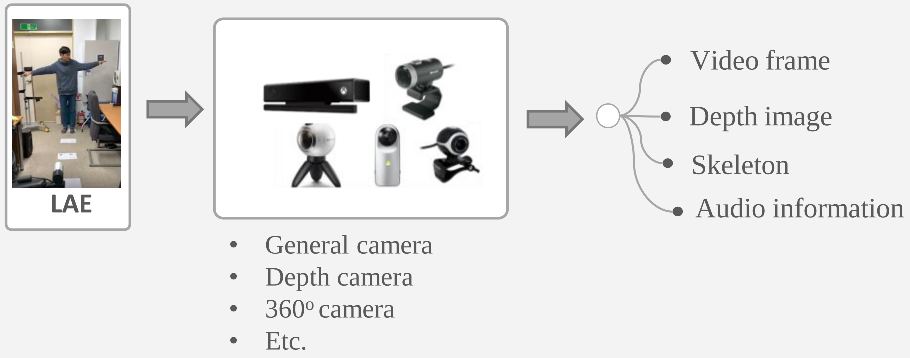

| LAEModel :: 3DLAEModel | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAE3DModel() | LAE3DModel | Protected | Constructor function |
| id | String | Public | Identifier |
| hidden | Boolean | Public | Hidden in a virtual scene |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| 3Dobject | 3Dobject | Public | A virtual object used for virtual scene |
| type | String | Public | Default type is 3DLAE (3d-live-actor) |
| joints | Array<vector2> | Private | A variable to store 2D joints of a body from an image |
| verts | Array<vector3> | Private | The vertices information for a predicted body model |
| joints3D | Array<vector3> | Private | 3D joints of a body for skeleton behaviors |
| normals | Array<vector3> | Private | Compute the normal for a 3D model to reflex with light |
| faces | Array<vector3> | Private | The faces of vertices |
| rotation | Vector3 | Public | A matrix for rotation in a scene |
| position | Vector3 | Public | A matrix for a position in a scene |
| scale | Vector3 | Public | A matrix for scale in a scene |
| processHMR() | Void | Private | Function to execute the model for constructing a 3D model from 2D image |
| processAudio() | Void | Private | Access the audio if it exists |
| threejsModelConverter() | 3DModel* | Private | Translate the joints3d and vertices to a 3D model |
| setData() | Void | Public | Set the sequences of 3d model data/audio as the input |
| getModelData() | 3DModel* | Public | Access function for the constructed model |
| getAudioData() | Audio* | Public | Access function for the audio |

# LAE-MAR Node Definition

## LAECapturer

❖ **LAECapturer** is responsible for accessing the connected camera and dealing with the image properties. The other modules can use this capturer data for different purposes.



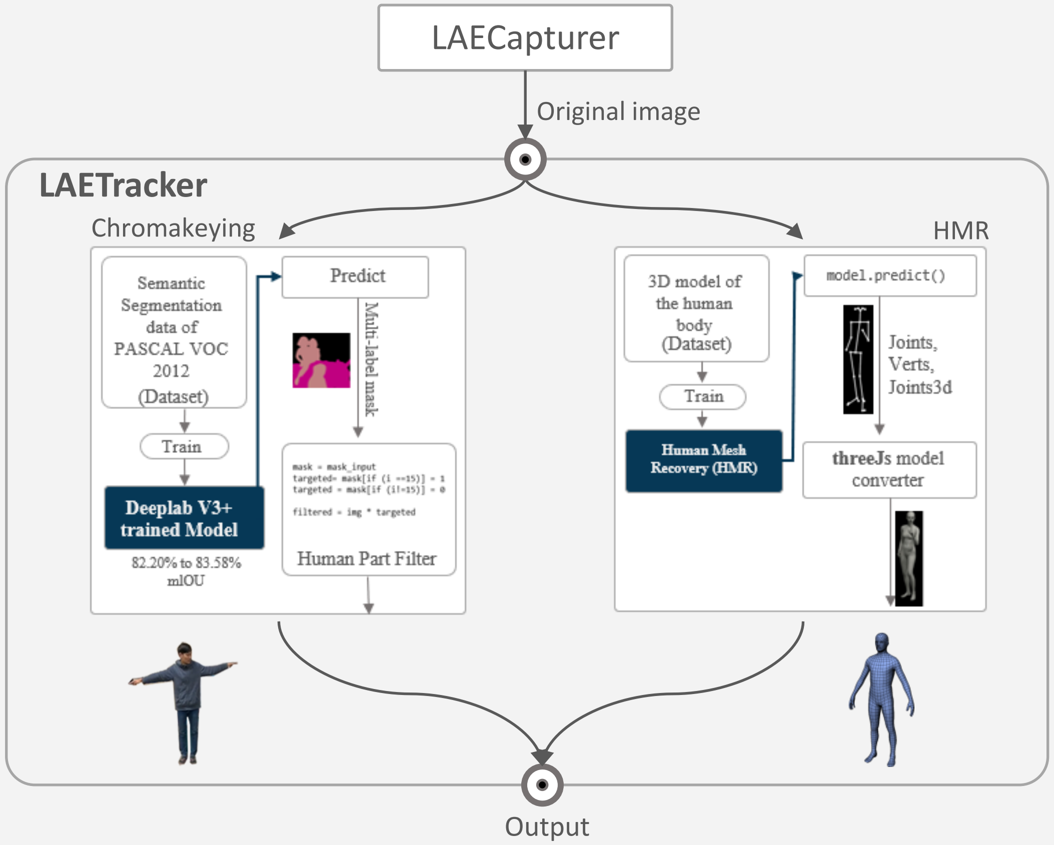




| LAECapturer | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAECapturer() | LAECapturer | Protected | Constructor function |
| id | String | Public | Identifier |
| enable | Boolean | Public | Enabling the process of capturing |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Define the type of camera (depth camera, general camera) |
| cameraId | number | Public | Set an id of a camera to be used |
| resolution | (number, number) | Public | Obtain *Width* and *Height* |
| mode | string | Public | Define image mode. E.g., RGB or black-white |
| imgData | Image | Public | Stores the sequentially captured images from a camera |
| setRawData() | Void | Public | Read the data directly from the camera |
| getData() | Any | Public | Access function for the captured data |

# LAE-MAR Node Definition

## LAETracker

- **LAETracker** functions to read the image, *laeCapturer*’s output, and outputs the data based on the type of the tracking method. In addition, the output can be of various types depending on the tracking technique—for instance, the tracker using for 2D LAE or using for 3D LAE.



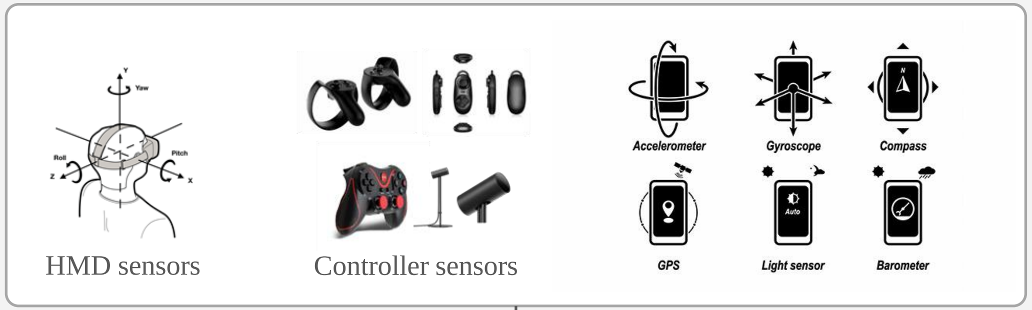

| LAETracker | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAETracker() | LAETracker | Protected | Constructor function |
| id | String | Public | Identifier |
| enable | Boolean | Public | Enabling the process of tracking |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Define the type of tracker (chromakeying or HMR) |
| rawData | Any | Private | Input data from sensor/capturer |
| deeplabModelSrc | Object | Private | Path of pre-train Deeplab model to be used |
| originalImg | Image | Private | A variable for the original image from the camera |
| maskImg | Image | Private | Mask image generated from Deeplab model |
| chromakeyingImg | Image | Private | Final output after filtering the live actor body |
| hmrModelSrc | Object | Private | Path of pre-train HMR model to be used |
| joints | Array<vector2> | Private | A variable to store 2D joints of a body from an image |
| verts | Array<vector3> | Private | The vertices information for a predicted body model |
| joints3D | Array<vector3> | Private | 3D joints of a body for skeleton behaviors |
| normals | Array<vector3> | Private | Compute the normal for a 3D model to reflex with light |
| deeplabPredict() | Object | private | A function to run the pretrained Deeplab model and do perdition |
| hmrPredict() | Image | private | A function to run the pre-trained HMR model and do perdition |
| setRawData() | Void | Public | Set the sensed data/captured data as the input |
| getResultData() | Any | Public | Access function for the tracked data |

# LAE-MAR Node Definition

## LAESensor

❖ In LAE context, **LAESensor** connects directly to the sensor or device and prepares the data for the *recognizer* module, which handles event listeners.

**Sensor Devices**




Input

LAE Sensor

Output

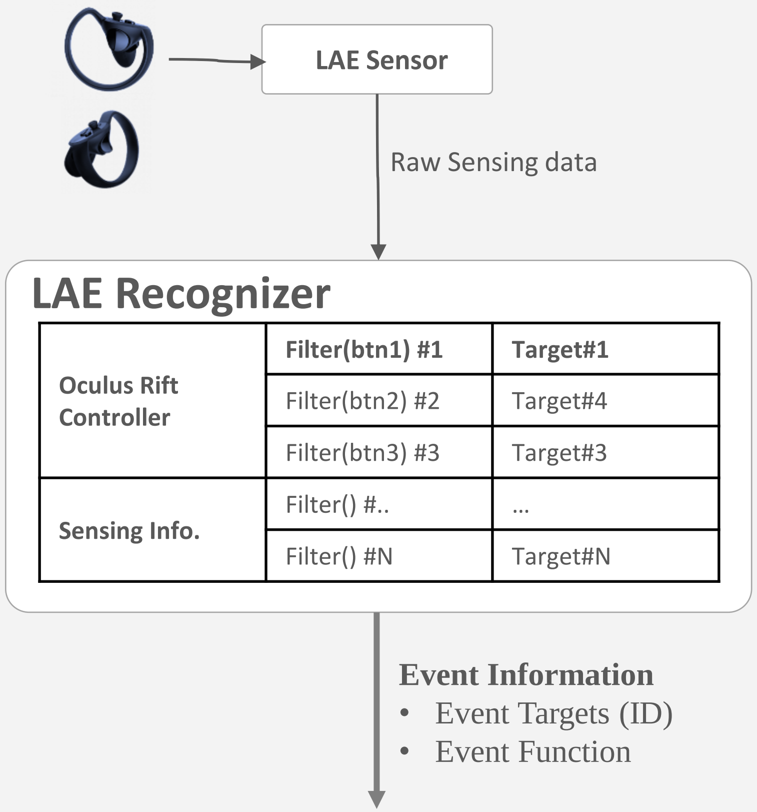
- Latitude, longitude
- Moving direction
- Camera
- Touch sensing
- Acceleration
- Velocity
- Motion
- Distance depth
- Gyroscope

| LAESensor | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAESensor() | LAESensor | Protected | Constructor function |
| id | String | Public | Identifier |
| enable | Boolean | Public | Enabling the process of capturing |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Specify the type of the device |
| rawData | Any | Private | Stores the sensing data |
| setRawData() | Void | Public | Read the data directly from the camera |
| getData() | Any | Public | Access function for the sensing data |

# LAE-MAR Node Definition

## LAERecognizer

- **LAERecognizer** tries to understand the targets from the input data and converts it to a piece of understandable information that can be used for LAE. In this case, LAE using this module to recognize the data of a device sensor, which is translated to an event listener

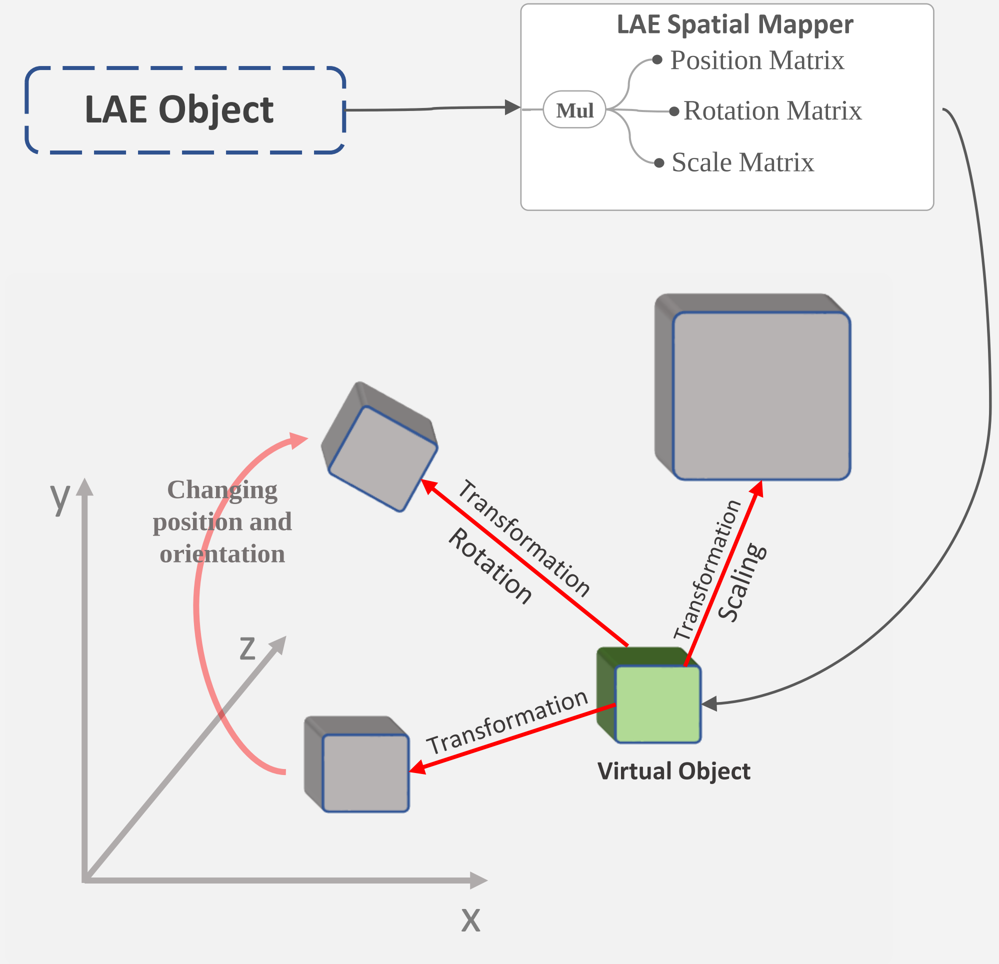



| LAERecognizer | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAERecognizer() | LAERecognizer | Protected | Constructor function |
| id | String | Public | Identifier |
| enable | Boolean | Public | Enabling the process of capturing |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Specify the type of event listener |
| rawData | Any | Private | Sensing data for being recognized |
| target | Any | Public | Definition of target |
| filter | Void | Private | Function filtering or post-processing the sensed and recognized data |
| targetHandler() | Void | Public | Telling that the target is activated |
| setRawData() | Void | Public | Read the data directly from the sensor module |
| getData() | Any | Public | Access function for the recognized data |

# LAE-MAR Node Definition

LAESpatialMapper

❖ In order to bring a real object into a virtual scene, there must be a functional node, **LAESpatialMapper**, to map a composed model with a virtual object in the MAR scene

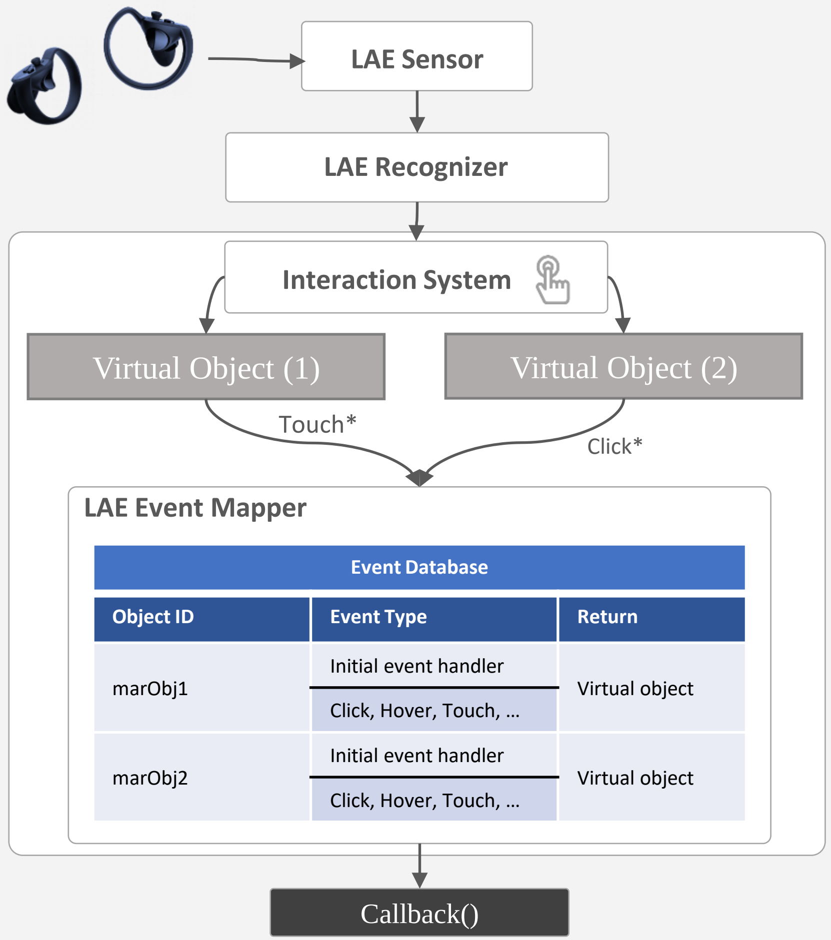



| LAESpatialMapper | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAESpatialMapper() | LAESpatialMapper | Protected | Constructor function |
| id | String | Public | Identifier |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| mappingLAEObj | Object | Public | A real LAE object to be mapped with MAR object |
| mappingMARObj | Object | Public | A virtual MAR object in a scene |
| position | Vector3 | Public | (X, Y, Z) is a vertex for positioning |
| scale | Vector3 | Public | (width, height, depth) is for seizing the virtual object in space |
| rotation | Vector3 | Public | (X, Y, Z) is a rotation based on the direction |
| mappingSpatialInfo() | Model | Private | Mapping the spatial information with a virtual object |
| setData() | void | Public | Read the data directly from the sensor module |
| getData() | Any | Public | Access function for the spatial information data |

# LAE-MAR Node Definition

## LAEEventMapper

❖ The events are all defined in the database, which describes the action of each object. Thus, the **LAEEventMapper** is just mapping a ***callback()*** function to an object that may return the model itself



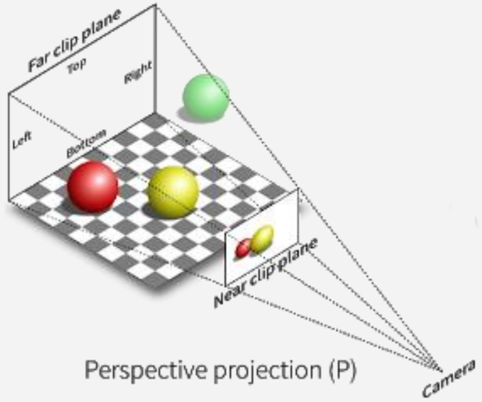

| LAEEventMapper | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAEEventMapper() | LAEEventMapper | Protected | Constructor function |
| id | String | Public | Identifier |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Specify the type of event listener |
| targetObjects | [IneractiveObject] | Public | An object that listens to the event trigger |
| getIntersection() | Void | Private | A raycaster function for recognizing the intersection |
| eventHandler() | Callback | Public | Handle the event depending on the action and target object; return the target |
| initialHandler() | Callback | Public | Handle the event at the initial stage; return the target |
| getData() | Any | Public | Access function for the data |

# LAE-MAR Node Definition

## LAEProjectionDisplay

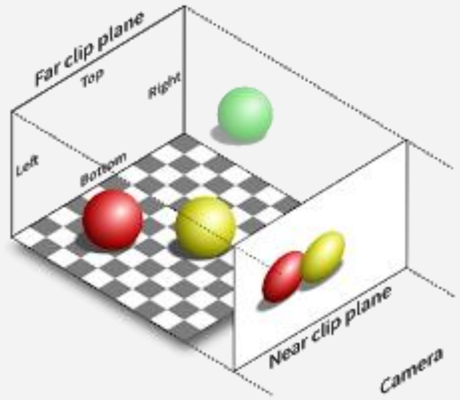
❖ **LAEProjectionDisplay** is used to describe where the camera should be put or looked at. There are properties applicable to project the scene, where it is considered as a view of interest.

- **Perspective**




Perspective projection (P)

- **Orthographic**



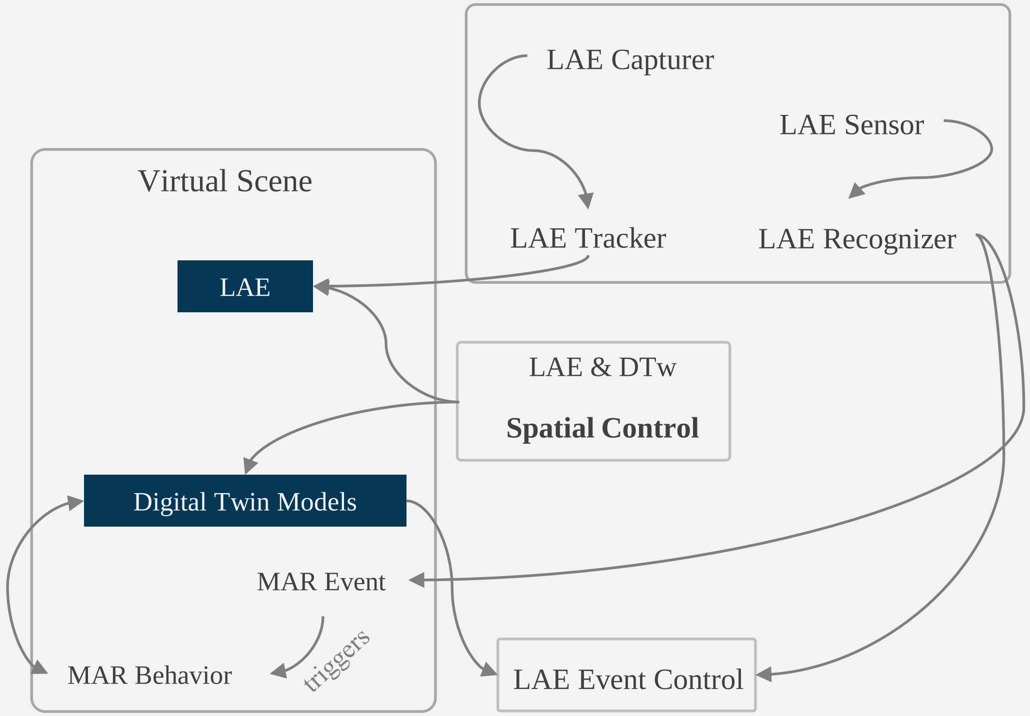

Orthographic projection (O)

| LAEProjectionDisplay | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAEProjectionDisplay() | LAEProjectionDisplay | Protected | Constructor function |
| id | String | Public | Identifier |
| entity | HTMLNode | Public | The entity that stores the HTML node information |
| type | String | Public | Type of camera (Perspective, Orthographic) |
| control | String | Public | Screen control type (Orbit, Map, etc.) |
| left | Number | Public | Left margin with a scale (0-1) |
| right | Number | Public | Right margin with a scale (0-1) |
| width | Number | Public | Width screen with a scale (0-1) |
| height | Number | Public | Height screen with a scale (0-1) |
| position | Vector3 | Public | Define a standing position of a projector |
| fov | Number | Public | Field of view, which is a maximum area for camera to image |
| nearDistance | Number | Public | The nearest distance to be captured |
| farDistance | Number | Public | The farthest of distance to be captured |
| lookAt | Vector3 | Public | Camera to look at |
| getData() | Any | Public | Access function for the projection display data |

# LAE-MAR Node Definition

## LAESceneRepresentation

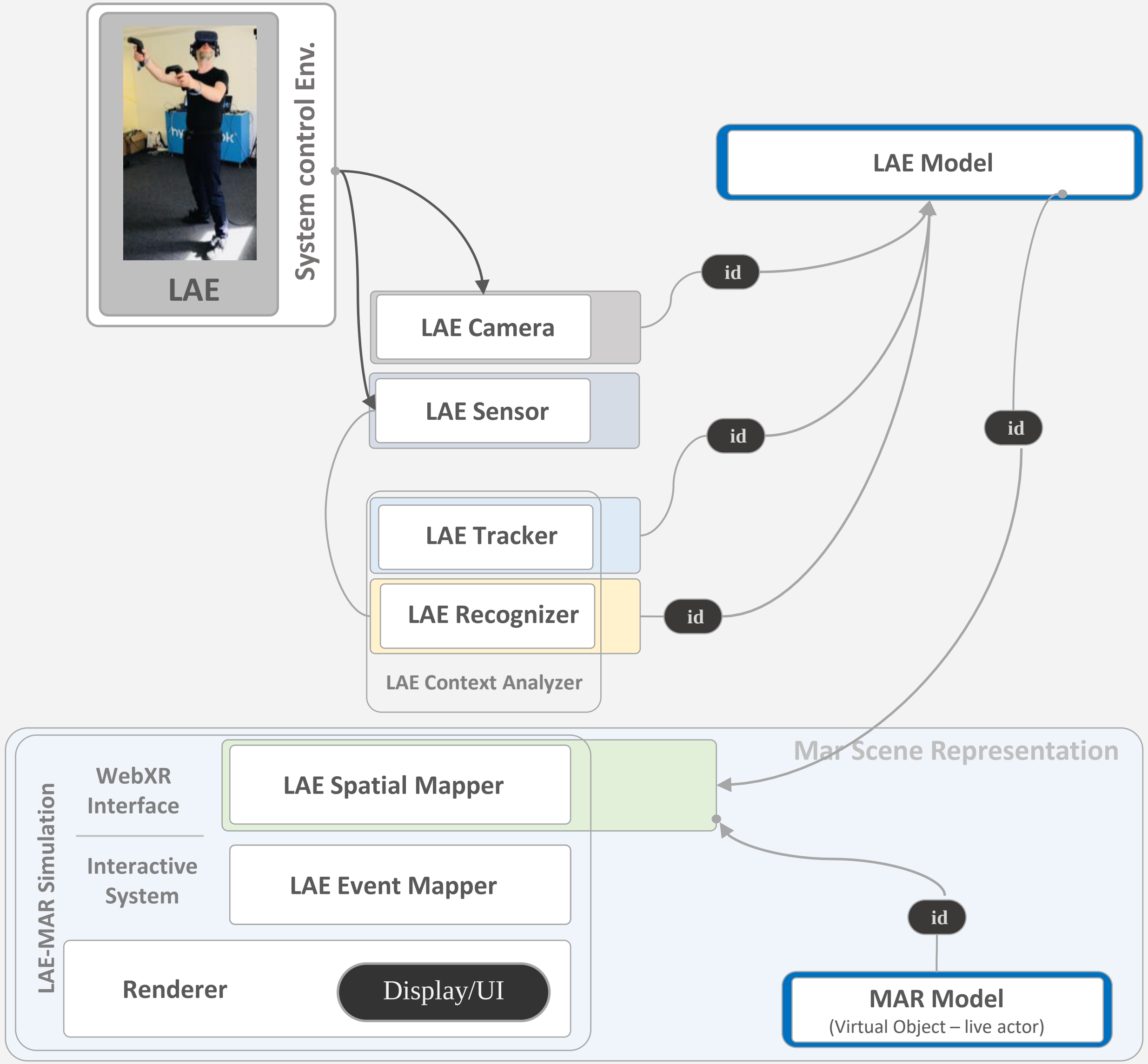
❖ **LAESceneRepresentation** plays a critical part in constructing the entire scene virtually. In addition, it is designed to cover the LAE Node and MARNode and simulate them into virtual objects under the control of the LAE spatial mapper and Event mapper.




| LAESceneRepresentation | | | |
|---|---|---|---|
| **Attr/Method** | **Type** | **Accessibility** | **Description** |
| LAESceneRepresentation() | LAESceneRepresentation | Protected | Constructor function |
| id | String | Public | Identifier |
| entity | HTMLNode | Public | The entity that stores the html node information |
| autoUpdate | Boolean | Private | Default is true. The renderer checks every frame if the scene and its objects need matrix updates |
| background | Object | Public | It can be set to a color or texture |
| children | [VirtualObject] | Private | Store the children node, which also represents the physical and virtual objects |
| addChild() | Void | Public | Add a child to this scene node |
| removeChild() | Void | Public | Remove a child from this scene node |
| removeAllChild() | Void | Public | Remove all children node |
| getChildren() | [VirtualObject] | Public | Obtain all containing node, children |
| getData() | Any | Public | Access function for the scene representation data |

# Node Relation

## LAE Node Relation

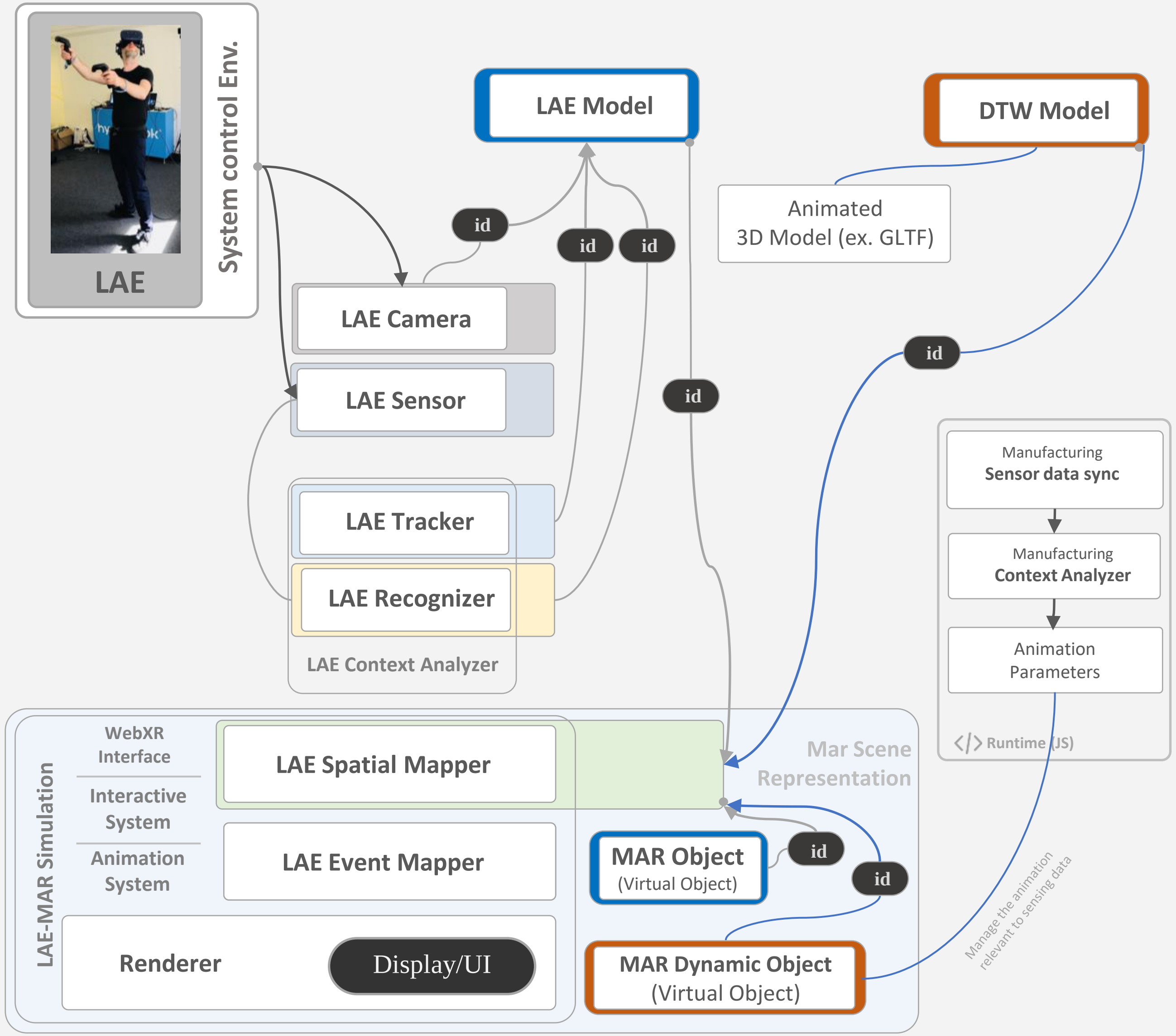



```
<LAE2DModel id="laemodel1" laeCaputurer="laecapture1"
       laeTracker="laetracker1" laeRecognizer="laereconizer1" />

<LAECapturer id="laecapture1" type="general-camera"
       cameraId="0" resolution="512x512" mode="rgb" />

<LAETracker id="laetracker1" type="chromakeying" />

<LAERecognizer id="laereconizer1" type="oculus-controller" />

<LAESpatialMapper id="laespatialmapper6" model="laemodel1"
       marObject="object1" position="1 1 1" scale="0.5 0.5 0"
       rotation="0.01 0.6 0.01"/>

<MARObject id="object1" type="live-actor"/>
```

Mar Scene Representation

# Node Relation

## LAE and DTW Node Relation (Ex. Dynamic DTW node)



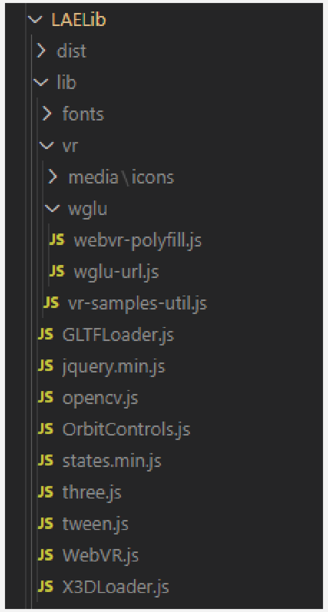

```
<LAE2DModel id="laemodel1“ laeCaputurer="laecapture1"
       laeTracker="laetracker1” laeRecognizer="laereconizer1“ />

<DTWModel id="dtwmodel2" modelFormat="gltf"
       src="/templates/3Dmodels/tapecasting.glb"/>

<LAECapturer id="laecapture1" type="general-camera"
       cameraId="0" resolution="512x512" mode="rgb“ />

<LAETracker id="laetracker1" type="chromakeying“ />

<LAERecognizer id="laereconizer1" type="oculus-controller“ />

<LAESpatialMapper id="laespatialmapper1" model="laemodel1"
       marObject="object1" position="1 1 1" scale="0.5 0.5 0"
       rotation="0.01 0.6 0.01“/>

<LAESpatialMapper id="laespatialmapper2" model="laemodel1"
       marObject="dtwmodel2" position="1 1 1" scale=“1 1 0"
       rotation=“0 0 0“/>

<MARObject id="object1" type="live-actor“/>

<MARDynamicObject id="sceneobject1“ type="3d-model“ deltaTime="0.016"
       timeAt="0.4“ />
```

Mar Scene Representation

# Implementation Results

Use case: Getting start with scene creation

## System-support libraries




Internal libraries for DTw-LAE-MAR

## Create your first scene

- Download the LAELib
- Import the LAE library to JavaScript tag

```
<script src="templates/LAELib/dist/bundle.js"></script>
```

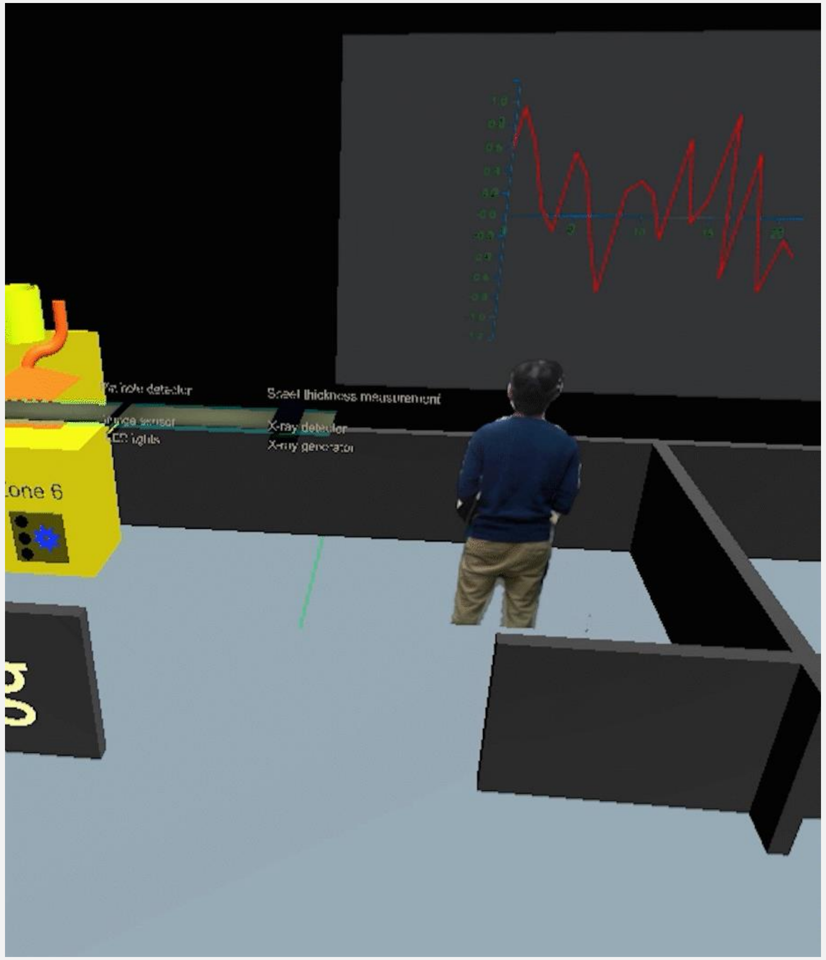
- Create a simple scene in <body> html tag with 3d object

```
<!-- MAR-LAE node -->
<MAR-LAE id="lae" isShownFPS="true" isShownConfig="false">
    <LAEProjectionDisplay>
        <LAEProjector id="projector"
                      control="orbit"
                      type="perspective-projector"
                      left="0" bottom="0" width="1" height="1"
                      position="0 4 5"
                      fov="75"
                      nearDistance="0.1"
                      farDistance="100000"> </LAEProjector>
    </LAEProjectionDisplay>
    <MARSceneRepresentation id="mar-scene">
      <!-- MAR Object -->
      <MARStaticObject id="staticbject1" type="3d-model"></MARStaticObject>
    </MARSceneRepresentation>
    <!-- Model -->
    <DTWModel id="laemodel1" src="/env-model.gltf"></DTWModel>
    <!-- Mappers -->
    <LAESpatialMapper id="laespatialmapper1"
                      model="laemodel1"
                      marObject="staticbject1"
                      position="1 1 1"
                      scale="0.5 0.5 0"
                      rotation="0.01 0.6 0.01">
    </LAESpatialMapper>
</MAR-LAE>
```

# Implementation Results

## Virtual Object::2D LAE

```
<!-- MAR-LAE node -->
<MAR-LAE id="lae" isShownFPS="true" isShownConfig="false">
   <MARSceneRepresentation id="mar-scene">
     <!-- MAR object for lae -->
     <MARObject id="object1" type="2d-live-actor"></MARObject>
  </MARSceneRepresentation>
  <!-- LAE Model -->
  <LAE2DModel id="laemodel1"
          laeCaputurer="laecapture1"
          laeTracker="laetracker1"
          laeRecognizer="laerecognizer1"
  ></LAE2DModel>
  <LAECapturer id="laecapture1"
              type="general-camera"
              cameraId="0"
              resolution="512x512"
              mode="rgb">
  </LAECapturer>
  <LAETracker id="laetracker1" type="chromakeying"> </LAETracker>
  <LAERecognizer id="laerecognizer1"
                type="oculus-controller">
  </LAERecorgnizer>
  <!-- Mappers -->
  <LAESpatialMapper id="laespatialmapper1"
                  model="laemodel1"
                  marObject="object1"
                  position="1 1 1"
                  scale="0.5 0.5 0"
                  rotation="0.01 0.6 0.01">
  </LAESpatialMapper>
</MAR-LAE>
<script>
  var laeMapper = document.getElementById("laespatialmapper1");
  laeMapper.setAttribute("scale", "1 1 1");
</script>
```

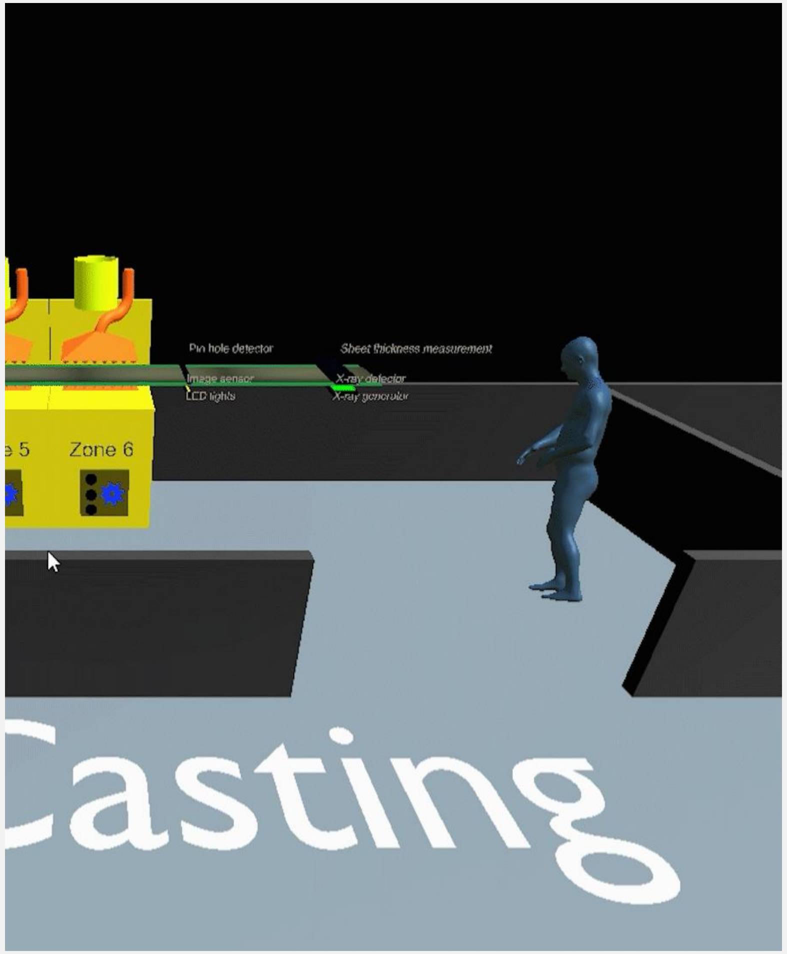


2D LAE DEMO

# Implementation Results

## Virtual Object::3D LAE

```
<!-- MAR-LAE node -->
<MAR-LAE id="lae" isShownFPS="true" isShownConfig="false">
  <MARSceneRepresentation id="mar-scene">
      <!-- MAR object for lae -->
      <MARObject id="object1" type="3d-live-actor"></MARObject>
  </MARSceneRepresentation>
  <!-- LAE Model -->
  <LAE3DModel id="laemodel1"
            laeCaputurer="laecapture1"
            laeTracker="laetracker1"
            laeRecognizer="laerecognizer1">
  </LAE3DModel>
  <LAECapturer id="laecapture1"
                type="general-camera"
                cameraId="0"
                resolution="512x512"
                mode="rgb">
  </LAECapturer>
  <LAETracker id="laetracker1" type="hmr"> </LAETracker>
  <LAERecognizer id="laerecognizer1"
                 type="oculus-controller">
  </LAERecorgnizer>
  <!-- Mappers -->
  <LAESpatialMapper id="laespatialmapper1"
                    model="laemodel1"
                    marObject="object1"
                    position="1 1 1"
                    scale="0.5 0.5 0"
                    rotation="0.01 0.6 0.01">
  </LAESpatialMapper>
</MAR-LAE>
<script>
  var laeMapper = document.getElementById("laespatialmapper1");
  laeMapper.setAttribute("position", "1 1 2");
</script>
```

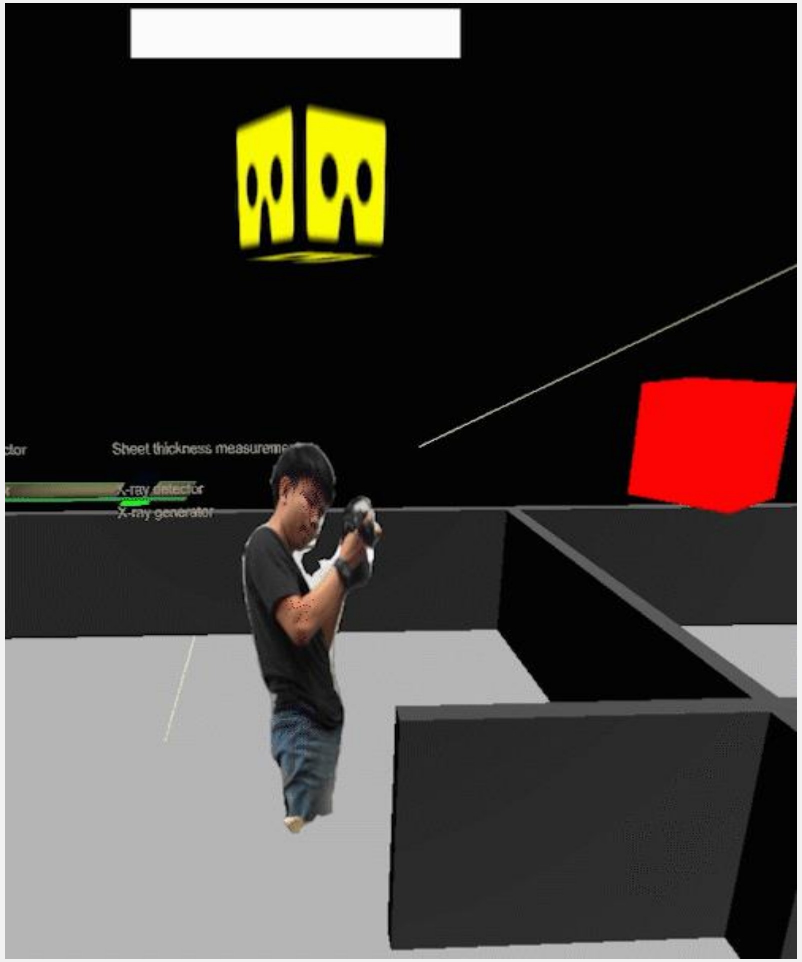



3D LAE DEMO

# Implementation Results

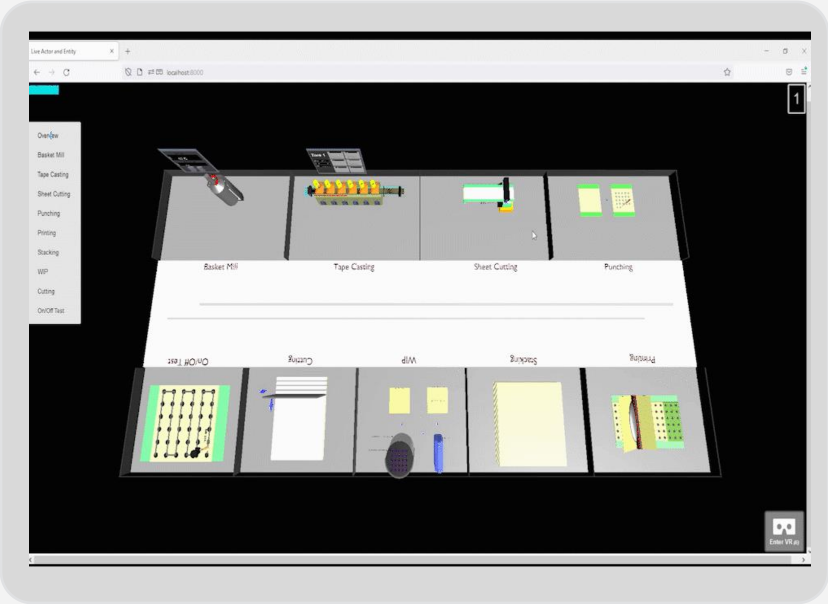
## Virtual Object::Interactive Scene

```
<MAR-LAE id="lae" isShownFPS="true" isShownConfig="false">
  <MARSceneRepresentation id="mar-scene">
      <MARObject id="object1" type="3d-live-actor"></MARObject>
      <MARInteractiveObject id="object2" type="cube"></MARInteractiveObject>
  </MARSceneRepresentation>
  <LAE3DModel id="laemodel1"
              laeCaputurer="laecapture1"
              laeTracker="laetracker1"
              laeRecognizer="laerecognizer1">
  </LAE3DModel>
  <DTWModel id="laemodel2"></DTWModel>
  <LAECapturer id="laecapture1"
        type="general-camera"
        cameraId="0"
        resolution="512x512"
        mode="rgb">
  </LAECapturer>
  <LAETracker id="laetracker1" type="chromakeying"> </LAETracker>
  <LAERecognizer id="laerecognizer1"
                 type="oculus-controller">
  </LAERecognizer>
  <!-- Mappers -->
  <LAESpatialMapper id="laespatialmapper1"
                    model="laemodel1"
                    marObject="object1"
                    position="1 1 1"
                    scale="0.5 0.5 0"
                    rotation="0.01 0.6 0.01">
  </LAESpatialMapper>
  <LAESpatialMapper id="laespatialmapper2"
                    model="laemodel2"
                    marObject="object2"
                    position="1 2 0"
                    scale="1 1 1"
                    rotation="1 1 1">
  </LAESpatialMapper>
  <LAEEventMapper id="laeeventmapper1" marObject="object2" type="click"> </LAEEventMapper>
</MAR-LAE>
<script>
  function onClickEvent(evt, a){
    console.log("I'm fired")
  }
  $('#laeeventmapper1').on( "onclick", onClickEvent);
</script>
```

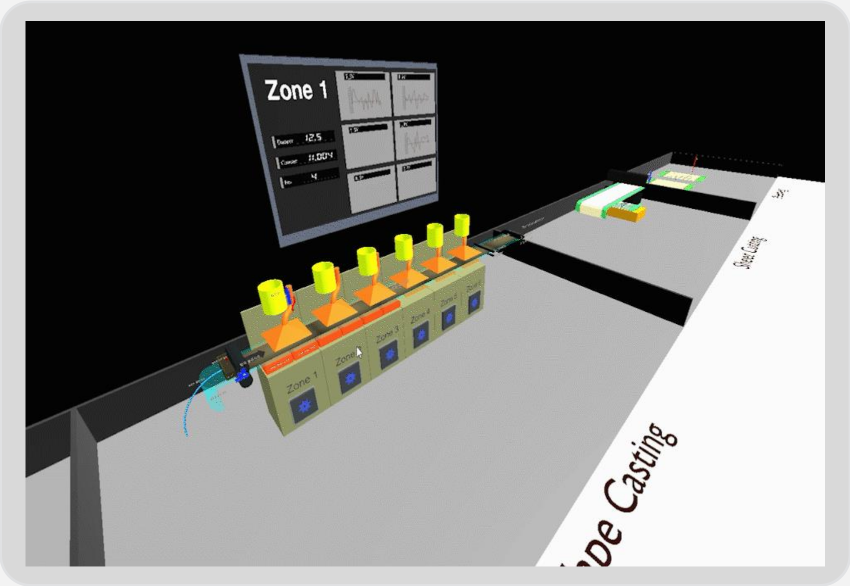



Interactive Scene Demo

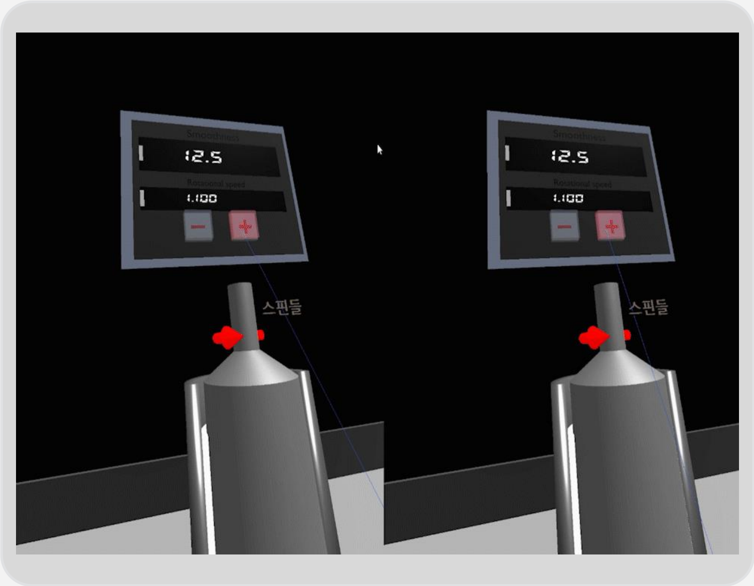
# Implementation Results




(A) Navigation through the manufacturing environment




(B) Focusing on a machine block




(C) The stereo display in VR device (HMD)

LAE and MAR Scene representation in virtual scene

In the Figure, (A) illustrates the overview of machine blocks and the navigation of routing to a specific block, which facilitates the ease of seeing details and controlling the system. As well, (B) describes that in a machine block, the process of manufacturing DTw models with system control panels. The LAE representation can be formed in space as a live actor to play around the scene with the ability of interaction through controller devices. In (C), the system provides two renders for screen display and VR display that the user in real-world can manage and watch the entire system process by using HMD.