# Sensor Data Viewer

**Elif Akbas**[†], **Beyzanur Kaya**[†]

**Abstract.** Buildings in Europe account for 40 percent of all energy use; therefore, it is crucial to monitor building energy usage levels, conditions originating from the building, and external factors like user behaviors and climate to generate accurate forecasts. Digital twins, gbXML models, and energy models are a few techniques to forecast potential energy usage,but the gap between real-life data results has a significant gap, therefore gathering and evaluating actual data is crucial for energy consumption prediction. Building Management Systems (BMS) and other smart building technologies, such sensor and Internet of Things (IoT) devices, are becoming widely used in new buildings and generate and gather a large quantity of data. Building information(BIM) modeling can be used to utilize this data, implement it into buildings, and produce meaningful implementation. Compiling these multiple data kinds from various sources and developing a common understanding is also compatible with BIM, which is a representation of the building, in order to comprehend the ways to combining these diverse data types is the purpose of the paper. To gain a basic understanding of the relationship between time series data and IFC models, temperature sensor values were projected into the IFC Model of the Umar Unit of the Nest Building.

## 1. Introduction

The forecast of energy usage in buildings is crucial in order to optimize their energy efficiency, with the goal of attaining energy conservation and improving environmental effect considering buildings account for 40 percent of all energy use in Europe [1]. To achieve better forecasting, monitoring data, and connecting the consumption with internal-building related, and external-user, climate related factors should be processed. Collecting data through sensors and IoT devices, maintenance of this data, forming useful and understandable output can be helpful to create more accurate predictions.

Building information modeling (BIM) addresses the continuous transfer of digital information throughout the entire life-cycle of a building complex, from the design and construction phases to operation [2]. BIM models are also machine readable information format contains geometry and further about the building, therefore convenient context to constitute a linking system, or a pattern for diverse data. Linking diverse data through BIM Models, presents data in a easily graspable way, and defining efficient usage of new technologies in the building environment makes interoperability and simplification of data very valuable [3].

Therefore first, creating a connection of sensor values and BIM model after visualizing the data over a building to broaden the target audience of the possible applications of the BMS was studied with the data from Umar Unit of NEST building.

### 1.1. Nest Building

Next Evolution in Sustainable Building Technologies(NEST) (Figure 2) is a research and development center located in Dübendorf, Switzerland, of two Swiss Research institutes, Swiss Federal Laboratories for Materials

[†] Chair of Design Computation - CAAD (Computer Aided Architectural Design), RWTH, Aachen, Germany.
Email: elif.akbash@rwth-aachen.de, beyzanur.kaya@rwth-aachen.de.

Science and Technology(EMPA) and Swiss Federal Institute of Aquatic Science and Technology (Eawag) [4]. Nest has a constant core and three free platforms allow, different units to be implemented in the building, creating the possibility to replace units as their purpose is completed, and opening up space for further research [4]. The research center creates an environment with advanced and innovative building technologies to test, better, and exhibit the applications in real-life circumstances[5].

Quarable Sensor Time Series Data monitored in web server, Sensor Metadata, and BIM Models (Revit and IFC) alongside gbXML model and Digital Twin for UMAR unit are the references for the case study, also knowledge graph broadcasting on GraphDB servers.

*Figure 1.* Umar at the Empa NEST © Zooey Braun, Stuttgart

### 1.2. Problem definition

NEST Building (Figure - 2) collects and stores various types of data such as BIM, sensor data, schematics, and user interfaces, interoperability and interconnectivity of different types is required to constitute meaningful results and usage of the innovative systems and information obtained from it. Creating the connection between sensor, GUI and Interfaces, and deriving precise information of the building in real time are the objectives of the study.

### 1.3. Research approach

”Room number” served as the foundation for all element-related input, and the BIM model, Time Series Data, and energy estimations all share a common room number value. As a result, the building’s room number values are identical by the Revit Model, ”IfcSpace” elements, and Time Series Data values, which all respond to spaces as a common aspect. Due to the comprehensiveness of their geometry, IfcSpace elements acted as a bridge between, as well as an element that could project information.

## 2. State of the art

To monitor and manage the energy consumption, sensors inputs and BIM, BEM configuration come into prominance. The integration of sensor data enables to facility management and reduces the gap between

predictions and reality. Therefore, to associate the sensors with the relative components creates more predictable model. The Nest building has been equipped with sensors network and has been a case study of BIM, BEM and BMS Project. The experimental studies in the existing building reveal emission-aware flexible prosumer system with Model Predictive Control (MPC) strategy. [6] The complex structure of sensor data is inconvenience to be interpretable by users; hence, visualization of monitored and analysed data simplify the data collections. In the case study; Integrating Building Information Modelling and Sensor Observations visualize the BIM model and sensors data by combining them in turtle format. Mainly, the BIM model is comprehended as 3D models which could be used for wide purpose with semantic model of BMS and improve to design phases with the analysis. [7]. In the case study, using BOT converter for IFC model leads to create semantic relationship in between IFC model and sensors which has visualized in 2D model to simplify the data

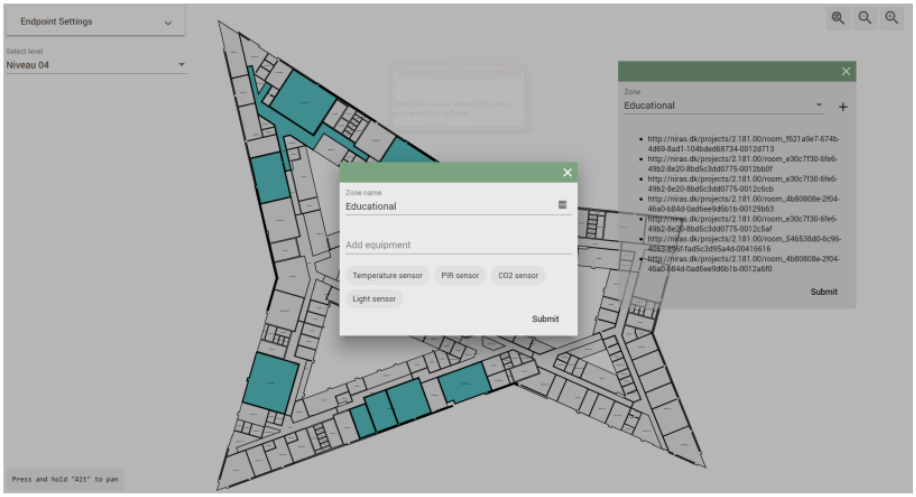


*Figure 2.* Application for assigning equipment templates to architectural spaces (Rasmussen et al., 2018a)

Even though research may serve several tools to visualize data thanks to the available technologies, it is hard to find an approach which merge inputs from different sources and formats, display it comprehensibly, keep it analysable properties at the same time. [7] To use architectural software for solid geometry restricts the interoperability of BIM applications. Although Revit does not provide appropriate RDF format, data converter ensure integration BIM with Linked Building Data to visualize it in graphic based interface in web server. Not only visualization, but also architectural model in RDF format allows querying. Therefore, BMS element does not exist in IFC model, it has converted to RDF format with IFCtoLBD to used in Graphdb which enable to query with sparQL and visualize knowledge graph. [3] The Nest Unit Umar Case study aims to integrate BIM and LBD with the scalable approach to represent the system semantically and display in user friendly interface with keeping semantic properties.

## 3. Methodological approach

This paper and prototype focuses on processing the data from:

### 3.1. Sensor Metadata

Sensor metadata has been stored in a web server including location, room number type data unit, signal type, numericID of sensor embedded in Nest building. Unit Umar which is the research scope contains various sensors in different purpose. Moreover, there is not correlation between IFC model of Unit Umar and sensor network.

To link the IFC model and sensors, temperature sensor has been determined as the scope of the project. Temperature sensor in Unit Umar is elected and its numericID, room number are obtained to get sensor timeseries.

```
for data in systemDatas:
    location_buildingDistrict = str(data['location_buildingDistrict'])
    datapoint_Unit = str(data['datapoint_Unit'])
    if location_buildingDistrict == 'Unit UMAR':
        if datapoint_Unit == ' C ':
            numericId = str(data['numericid'])
```

### 3.2. Sensor Timeseries

Nest building time series data collected in a web server is questionable with the sensor numericID.

```
for data in systemDatas:
    location_buildingDistrict = str(data['location_buildingDistrict'])
    datapoint_Unit = str(data['datapoint_Unit'])
    if location_buildingDistrict == 'Unit UMAR':
        if datapoint_Unit == ' C ':
            numericId = str(data['numericid'])
```

After Since the room is including various sensors, several values occurs for the same room. Therefore; averaging the value in the same room to visualization gives more accurate result.

```
def Average(lst):
    return sum(lst) / len(lst)
```

### 3.3. IFC Model and Sensor Metadata-Timeseries

Umar IFC model stores 3D model, except sensor entities which is a part of Building Managament System and obstruct the link IFC model and sensor metadata. IFC Space representing the room in the IFC model is linked to the room number, either involves property sets standing for sensors such as IfcThermodynamicTemperatureMeasure under SpaceTemperature. Within the scope of the project, linking temperature timeseries to the ifc space has been determined as a main objective.

.

3.3.1.Export Settings

As the starting point of the project, export setting in Revit should be adjusted to prevent errors caused because of different formats and versions.

Settings have been followed to export the model with IfcSpace;

1. Each room of the building is also marked with room tool on revit model

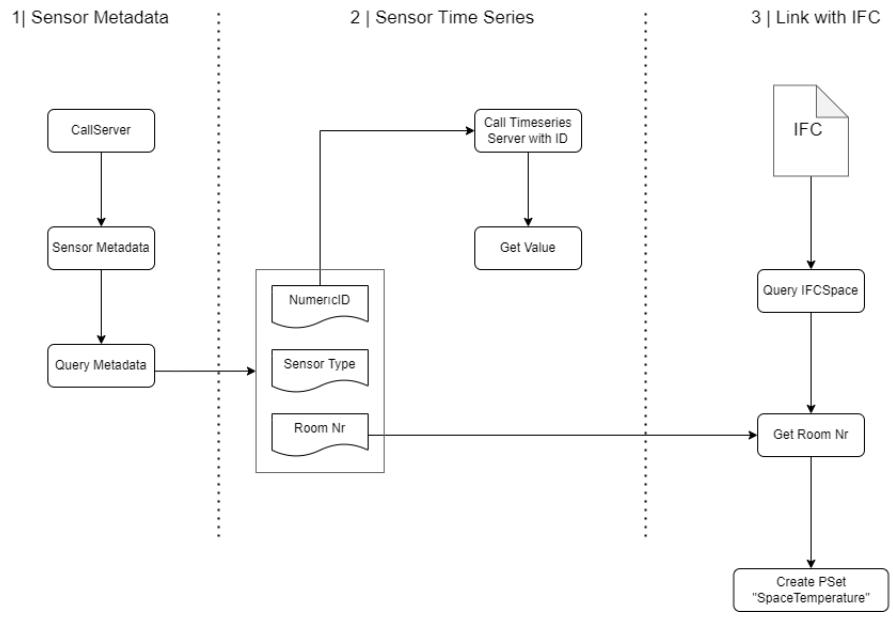


*Figure 3.* Sensor Metadata — Timeseries — Ifc Model

2. All the elements are in the active view before exporting

3. On the export options following Revit Elements match the certain Ifc Types

   (a) Space - IfcSpace

   (b) Rooms - IfcSpace

   (c) Room Tags - IfcPropertySingleValue

4. On the export setup, following setting were used

   (a) IFC version : IFC Version 4 reference view

   (b) Space boundaries : 1st Level

   (c) Export only elements visible in view, export rooms in 3D views : checked

   (d) Export IFC common property sets : checked

   (e) Use 2D room boundaries for room volume : checked

   (f) Include IFCSITE elevation in the site local placement origin

.

3.3.2.Linking IfcSpace to Sensor Network

1. To link sensor metadata which stored in web server and IFC file, room numbers are pointed out as a common ground. Matching IfcSpace through room number with the sensor metadata stored as JSON enabled to interoperability between different formats.

```
if apiRoomNumber == ifcSpaceRoomNumber:
    print('Room API ' + apiRoomNumber)
    print('Room IFC ' + ifcSpaceRoomNumber)
    val = sensorValue[0]['value']
    value_result_list.append(val)
```

1. To create PSet representing Building Management System, IfcThermodynamicTemperatureMeasure property set is created under the selected IfcSpace matching with sensor metadata.

```
property_values = [
    ifcFile.createIfcPropertySingleValue("SpaceThermalRequirements", "SpaceTemperature",
    space, ifcFile.create_entity("IfcThermodynamicTemperatureMeasure"))
]
property_set = ifcFile.createIfcPropertySet(space.GlobalId, owner_history,
"SpaceThermalRequirements", None, property_values)

ifcFile.createIfcRelDefinesByProperties(space.GlobalId, owner_history, None, None,
[space], property_set)
```

### 3.4. Viewer

Xeokit SDK is a toolkit to program the BIM viewer, it contains plugins and various functions in Javascript language, and runs in the browser via HTML files, some examples are shown on the Github page. In addition to the SDK, Xeolabs provides a BIM Viewer application unrelated to our project, the xeokit-metadata tool creates the model hierarchy and the xeokit-conver tool converts the IFC file to the native XKT Xeokit format [8].

The viewer supports a number of file formats, including IFC, XKT, CityJSON, LAZ, LAS, gITF, STL, glb, and OBJ. However, as we wanted to link to several data types with our BIM model, we focused on IFC and XKT formats. The application runs on the browser, so creating a correct HTML page is critical, any line of code not written correctly will directly break the functionality of the viewer. On the header section of the HTML code, styling the viewer which will be added to viewer to extend functionality can be styled.

### 3.5. Plugins

To create the app, all the plugins should be imported, Nav Cube Plugin, Tree View Plugin (Figure 3.5), Viewer, and a Loader which will be explained in more detail alongside all the elements. Nav Cube Plugin allows us to control the model by the center of the model, and rotating NavCube allows orbiting the camera accordingly [9].

```
  <script type="module">
    import {
      Viewer,
      XKTLoaderPlugin,
      NavCubePlugin,
      TreeViewPlugin,
    } from "https://cdn.jsdelivr.net/npm/@xeokit/xeokit-sdk/dist/xeokit-sdk.es.min.js";
```

TreeViewPlugin is an HTML tree to navigate IFC elements according to hierarchy, Hierarchy is aligned with IFC structure, IfcSite contains IfcBuilding which contains IfcStorey... Each element, even a group of elements, has

a checkbox to change its Visibility control, even the visibility of elements can be cycled through view or load modules if it needs to be automated [10]. The three hierarchy modes are Containment, Types, and Floors.

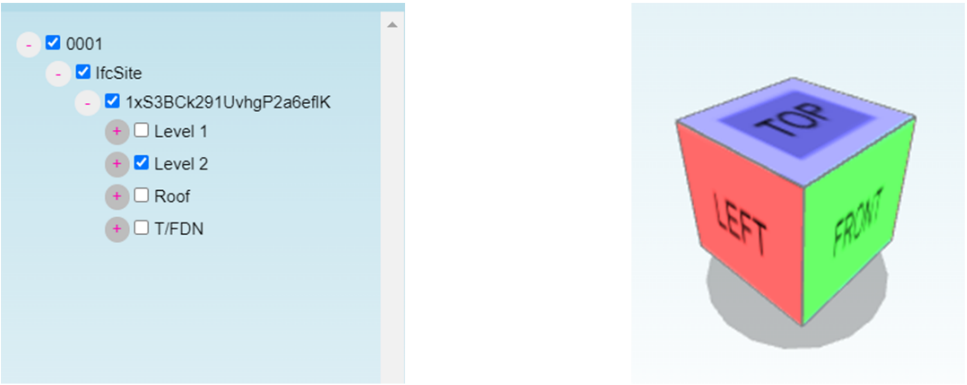


*Figure 4.* Tree View Plugin and Navigation Cube Plugin

Besides these additional tools that we added to increase the functionality and control of the model, the main plugins are Viewer and LoaderPlugins, which are the modules for getting and reading the file and creating the geometry. After defining the viewer, a canvas element must be defined and the camera must be positioned. For positioning, the camera module must be set for camera.eye, camera.look and camera.up values [11].

```
const viewer = new Viewer({
  canvasId: "myCanvas",
  transparent: true,
});

viewer.camera.eye = [-3.933, 2.855, 27.018];
viewer.camera.look = [4.4, 3.724, 8.899];
viewer.camera.up = [-0.018, 0.999, 0.039];
```

TreeViewPlugin is an HTML tree to navigate through the IFC elements according to the hierarchy, the hierarchy is aligned with the IFC tree, IfcSite contains IfcBuilding which contains IfcStorey ... Each element even group of elements has a checkbox to control the visibility , also the Visibility of the elements can be navigated through viewer or Loader Modules if it wants to be automated. The modes of the tree hierarchy are ”containment”, ”types” and ”storeys” [12].

Besides to these additional tools that we have added to increase the functionality and control over the model, the main plugins are ”Viewer” and LoaderPlugins, the modules to retrieve and read the file and create the geometry. After defining the viewer, you need to define a canvas element and position the camera as well. To position, the camera module must be defined for the camera.eye, camera.look and camera.up values.

### 3.6. Model Loaders

After configuring the viewer, the model is upload, which is available by setting the file path through the Loader plugins, WebIfcLoaderPlugin for IFC models and XKTLoaderPlugin for XKT models. Loaders let you set default objects to control visibility, opacity, color, priority, and pickability for different types of IFC elements. A loader must be defined with ID, file source, excluded types to prevent certain IFC element types from being loaded, include types to load only certain IFC element types, and edges to limit the visibility of edges [13]. Different storeys should be isolated in order to visualize temperature values in the viewer, which can also be managed with ViewsPlugin. However, because we only have one building storey for UMAR Unit, using x-ray options over BIM Model to isolate small elements from other storeys was sufficient. Also, for the Duplex model trial sessions, the x-ray module, which is a viewer subfunction, was chosen. However, in order to reduce the model's complexity and level of detail, the visibility of the "IfcCovering" and "IfcFurniture" elements are also enabled.

```
    const myObjectDefaults = {
        IfcCovering: {
        visible: false,
      },
        IfcSpace: {
        visible: true,
      },
      ...
    };
```

### 3.7. IFC and Model Viewer

The model (Figure 5) used for this research contains the Umar Unit of the Nest Building, since the units aimed to be modular, the model only contains the unit and can be replaceable with the next unit for the following research. The model was developed in Autodesk Revit but also modified with the BlenderBim add-on. The model contains space, storey, and building elements alongside detailed information on the material of each component. However, visualizing temperature values through IfcSpaces only requires basic geometry and proper IfcSpace elements, the room elements have significant importance. IfcSpaceElements are masked by loaders by default. Additionally, you may want to consider creating IfcProjection elements with the same names (room numbers from the Revit model) and room geometry they replicate for easier adaptation to the software.

```
  <script type="module">
    import {
      Viewer,
      XKTLoaderPlugin,
      NavCubePlugin,
      TreeViewPlugin,
    } from "https://cdn.jsdelivr.net/npm/@xeokit/xeokit-sdk/dist/xeokit-sdk.es.min.js";
```

One of the plugins to open the model is WebIfcLoaderPlugin, which allows to open the IFC model without converting it to XKT format. However, it is slower and only suitable for small to medium-sized models, which can cause problems with more complex building geometries. Although this view is compatible with the Umar unit source model, the mentioned options do not work well enough with WebIFCLoaderPlugin. Since excluding and manipulating models with the xeokit-sdk tools was not entirely efficient, IfcStyledItem was used to colorize

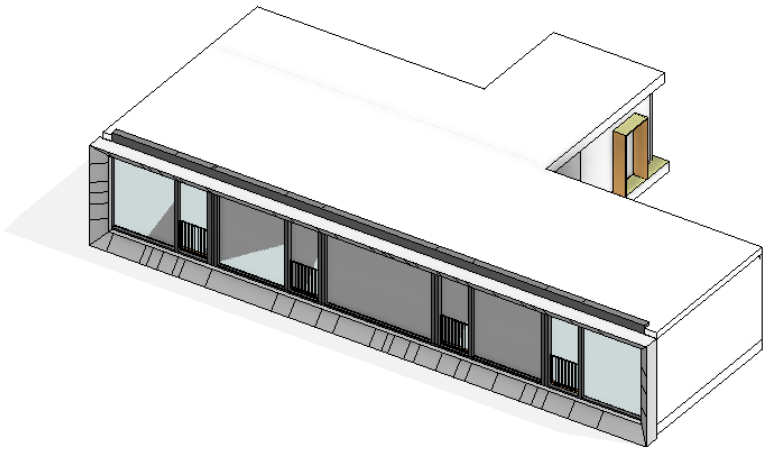

*Figure 5.* BIM Model Developed in Revit

IfcSpaces in the same loop, creating ifcThermodynamicTemperatureMeasurement property sets. The approach for projecting temperature values via IfCSpace elements was to create a color gradient in blue-white and white-red and match different thermal values to build a dictionary, then match these values with sensor values to assign the correct RGB color values for the relevant IfcSpace elements. The approach for projecting temperature values via IfCSpace elements was to create a color gradient in blue-white and white-red and match different thermal values to build a dictionary, then match these values with sensor values to assign the correct RGB color values for the relevant IfcSpace elements. This approach was not preferred due to the static value. However, given the compatibility issues with the WebIfcLoader and IFC models, creating a prepared static model was an efficient way for the viewer with an IFC model.

```
colorsblue = (list(blue.range_to(Color("white"), 50)))
...
temperature = [*range(-50, 51, 1)]
temperature_color_match = {k: v for k, v in zip(temperature, rgb_colors)}
...
        color = temperature_color_match[int(round(average))]
        print(color)
        surfaceStyleRendering = ifcfile.createIfcSurfaceStyleRendering()
        surfaceStyleRendering.SurfaceColour = color
        surfaceStyle = ifcfile.createIfcSurfaceStyle(
            color.Name, "BOTH", (surfaceStyleRendering,))
        presStyleAssign = ifcfile.createIfcPresentationStyleAssignment(
            (surfaceStyle,))
        item = space.Representation.Representations[0].Items[0]
        ifcfile.createIfcStyledItem(item, (presStyleAssign,), color.Name)
```

### 3.8. XKT Model

The second method of opening a file involved using the XKTLoaderPlugin. While this format improves the functionality of the viewer, it also requires multiple steps to convert to the correct version because different versions handle some conversions differently.

```
const sceneModel = xktLoader.load({
  id: "myModel",
  src: "../assets/models/xkt/v8/ifc/Duplex.ifc.xkt",
  //excludeTypes: ["IfcBuildingElementsProxy"],
  objectDefaults: myObjectDefaults,
  edges: true,
});
```

XKT is a binary format that contains metadata as well as the GUIDs from IFC Models and is used to load and view models quickly on a browser page. These GUIDs are used to identify different elements or hierarchical elements. The IFC model can be converted to an XKT model using the xeokit-convert command line tool, resulting in a Version 10 XKT model; however, Version 10 has limitations when converting IfcSpace. To keep the IfcSpace Elements model, it must be converted to a version 8 model, which is a process that involves several steps using various open source command line tools [14], and unfortunately, reaching required versions of some tools is not possible.

Experimenting on the version 8 XKT model of the duplex model, exporting rooms from Revit to IfcProjectionElement, or making a copy of the space with IfcOpenShell library through geom function into an IfcProjectionElement can help on avoiding the complex conversion procedure into version 8 model. The project can then be completed using IfcProjectionElement instead of IfcSpace Elements. Converting with xeokit-convert is a straightforward process that can be run over the xeolabs docker image, albeit it does not meet all criteria.

After defining the XKT model to the loader plugin, metadata function can be used to collect the required data. Element GUIDs from IFC Models, serve as ID in metaModel. First, a metaModel should be given which was already described during the Loader plugin definition, afterwards, an ID of an element needs to be used to define metaObject [15]. Besides, the picking function can select the element and get the id instead of a static description. After defining a floor for metaObjects using “getObjectIDsInSubtreeByType “ helps to collect IfcSpace elements in a container. IfcSpace elements colorized with “setObjectsColorized” function under scene.

### 3.9. Conclusion

Despite the sensor network is seen as tool for creating digital twin, deficiencies of relationship between sensor network and IFC model causes differentiation between in real time value and forecast. Due to incoordination of semantic relationship and sensor, IoT devices, forecast is inadequate for energy management which is essential for optimizing energy consumption. To know building properties and relationship of components with each other are valuable to predict energy consumption scenarios and minimize it. Nevertheless, handling with different formats is inconvenience to ensure interoperability and unintelligible for users, which reduce eligibility of it. Therefore, linking BIM and sensor network and creating user friendly, interpretable interface via visualization are the main objectives of the sensor viewer project.

# Bibliography


[1] Hai-xiang Zhao and Frédéric Magoulès. "A review on the prediction of building energy consumption". In: *Renewable and Sustainable Energy Reviews* 16.6 (2012), pp. 3586–3592.

[2] André Borrmann et al. "Building information modeling: Why? what? how?" In: *Building information modeling*. Springer, 2018, pp. 1–24.

[3] Lasitha Chamari, Ekaterina Petrova, and Pieter Pauwels. *A web-based approach to BMS, BIM and IoT integration: a case study*. May 2022. DOI: 10.34641/clima.2022.228.

[4] *About Nest*. URL: https://www.empa.ch/web/nest/aboutnest.

[5] Felix Heisel and Sabine Rau-Oberhuber. "Calculation and evaluation of circularity indicators for the built environment using the case studies of UMAR and Madaster". In: *Journal of Cleaner Production* 243 (2020), p. 118482.

[6] Hanmin Cai and Philipp Heer. "Experimental implementation of an emission-aware prosumer with online flexibility quantification and provision". In: *ArXiv* abs/2110.12831 (2021).

[7] Mads Holten Rasmussen et al. "Demo: Integrating Building Information Modeling and Sensor Observations using Semantic Web". In: Oct. 2018.

[8] Xeolabs. *Xeokit Github Page*. URL: https://github.com/xeokit.

[9] *Documentation NavCubePlugin*. https://xeokit.github.io/xeokit-sdk/docs/class/src/plugins/NavCubePlugin/NavCubePlugin.js~NavCubePlugin.html. [Accessed 28-Aug-2022].

[10] *Documentation - Tree View Plugin*. https://xeokit.github.io/xeokit-sdk/docs/class/src/plugins/TreeViewPlugin/TreeViewPlugin.js~TreeViewPlugin.html. [Accessed 28-Aug-2022].

[11] *Documentation - Viewer Plugin*. https://xeokit.github.io/xeokit-sdk/docs/class/src/viewer/Viewer.js~Viewer.html. [Accessed 28-Aug-2022].

[12] Xeokit. *Documentation - Tree View Plugin*. https://xeokit.github.io/xeokit-sdk/docs/class/src/plugins/TreeViewPlugin/TreeViewPlugin.js~TreeViewPlugin.html. [Accessed 28-Aug-2022].

[13] Xeokit. *Documentation - XKT Loader Plugin*. https://xeokit.github.io/xeokit-sdk/docs/class/src/plugins/XKTLoaderPlugin/XKTLoaderPlugin.js~XKTLoaderPlugin.html. [Accessed 28-Aug-2022].

[14] Xeokit. *Converting IFC Models to XKT using Open Source Tools - A Simpler Pipeline*. https://www.notion.so/Converting-IFC-Models-to-XKT-using-Open-Source-Tools-A-Simpler-Pipeline-02d45ba457eb4f808f63bcacb71a4fb3. [Accessed 28-Aug-2022].

[15] Xeokit. *Scene Representation and Metadata*. https://www.notion.so/Scene-Representation-and-Metadata-936fa040f4e9417bb83cc04136ca0b8d. [Accessed 28-Aug-2022].