OpenTwins Digital Twin Mockup for BIPV

Revisions

- Version 1.0 - First draft – Architecture and data model - Nicolò Molinari - 26/05/2025
- Version 1.1 – Dashboard description and typo correction – Nicolò Molinari – 06/08/2025
- Version 1.2 – Upgrade OpenTwins procedure, infrastructure schema update – Nicolò Molinari – 08/09/2025
- Version 1.3 – Latest progress on Unity panel and Grafana dashboard – Nicolò Molinari – 22/09/2025

Goal: Evaluate the adoption of OpenTwins for the development of a BIPV digital twin

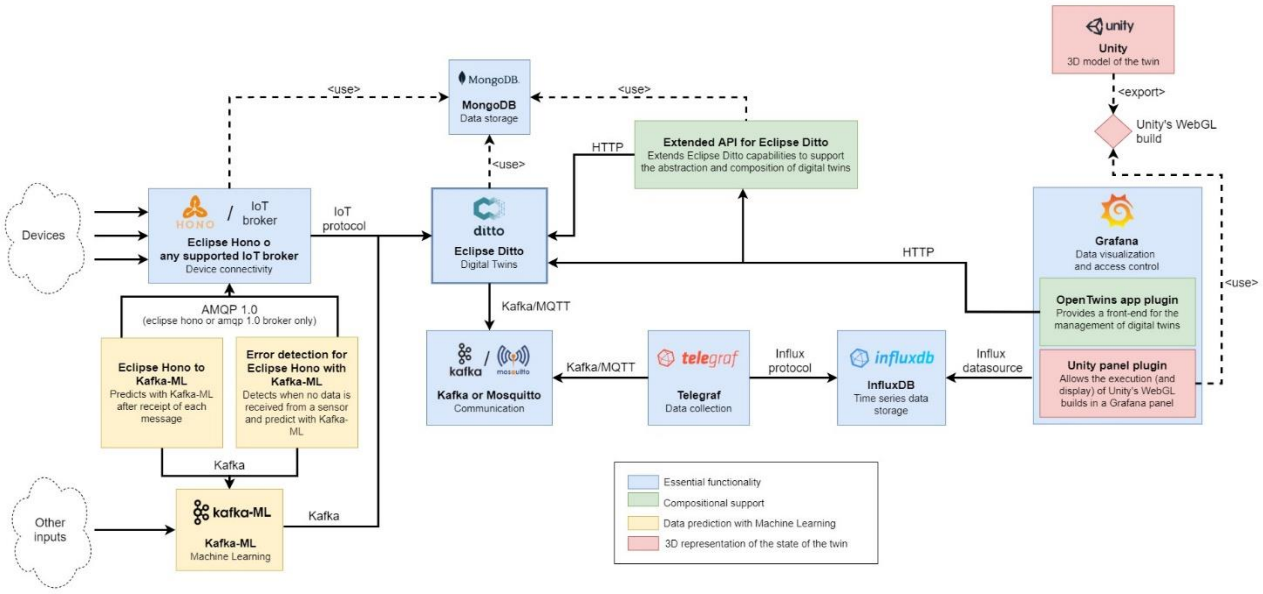
OpenTwins

OpenTwins is an open-source project developed by the University of Malaga. It is not a complete and fully supported product. Every feature needs to be checked and validated, and the documentation does not cover all the components.

Ref: https://ertis-research.github.io/opentwins/docs/quickstart

Architecture

OpenTwins works as a composition of components. The basic idea is to collect data directly from IoT devices (e.g., sensors) with an IoT broker (Eclipse Mosquitto) together with Eclipse Hono.



The collected data need to be mapped to elements that compose the digital twin. Eclipse Ditto oversees this work. Finally, the data are stored in InfluxDB through Telegraf collector.

For what concerns the presentation layer, the adopted tool is Grafana with its native integration with InfluxDB. 3D rendering is enabled by an OpenTwins custom plugin that grants the possibility to visualize Unity objects in Grafana.

The digital twin creation and management are done through a second OpenTwins plugin. In this way, it is possible to design the Ditto elements that compose the digital twin directly from Grafana.

The steps to create a digital twin are the following:

- Define a hierarchical structure of elements with their attributes from the OpenTwins Ditto plugin
- Write and run a script that receives data from sensors, typically through MQTT, and creates and sends the Ditto messages that match the structure defined on the plugin
- Once the data are sent to Ditto, OpenTwins automatically populates the digital twin and stores all the time series in InfluxDB
- Stored data can be presented and analyzed on Grafana using its out-of-the-box capabilities
- The 3D graphical representation should be first developed in Unity and then imported into Grafana using the OpenTwins Unity plugin
- Given the fact that Unity is a game engine, there is a wide range of possibilities in terms of interaction and simulation with the 3D object
- The OpenTwins Unity plugin enables the possibility to send data from Grafana to Unity (it should be possible, for example, to act on the 3D object dynamically based on the value fetched from the sensors and accessed through Grafana in InfluxDB)

Deployment

OpenTwins is designed to be deployed on a Kubernetes cluster and installed via Helm (manual installation is also possible). The recommended container manager is Docker. There is also the possibility to install OpenTwins locally using Minikube. For the BIPV project, Minikube, used just for trial purposes, showed a malfunction with Mosquitto. Once OpenTwins is deployed with Helm, some settings should be done on Grafana (see OpenTwins documentation).

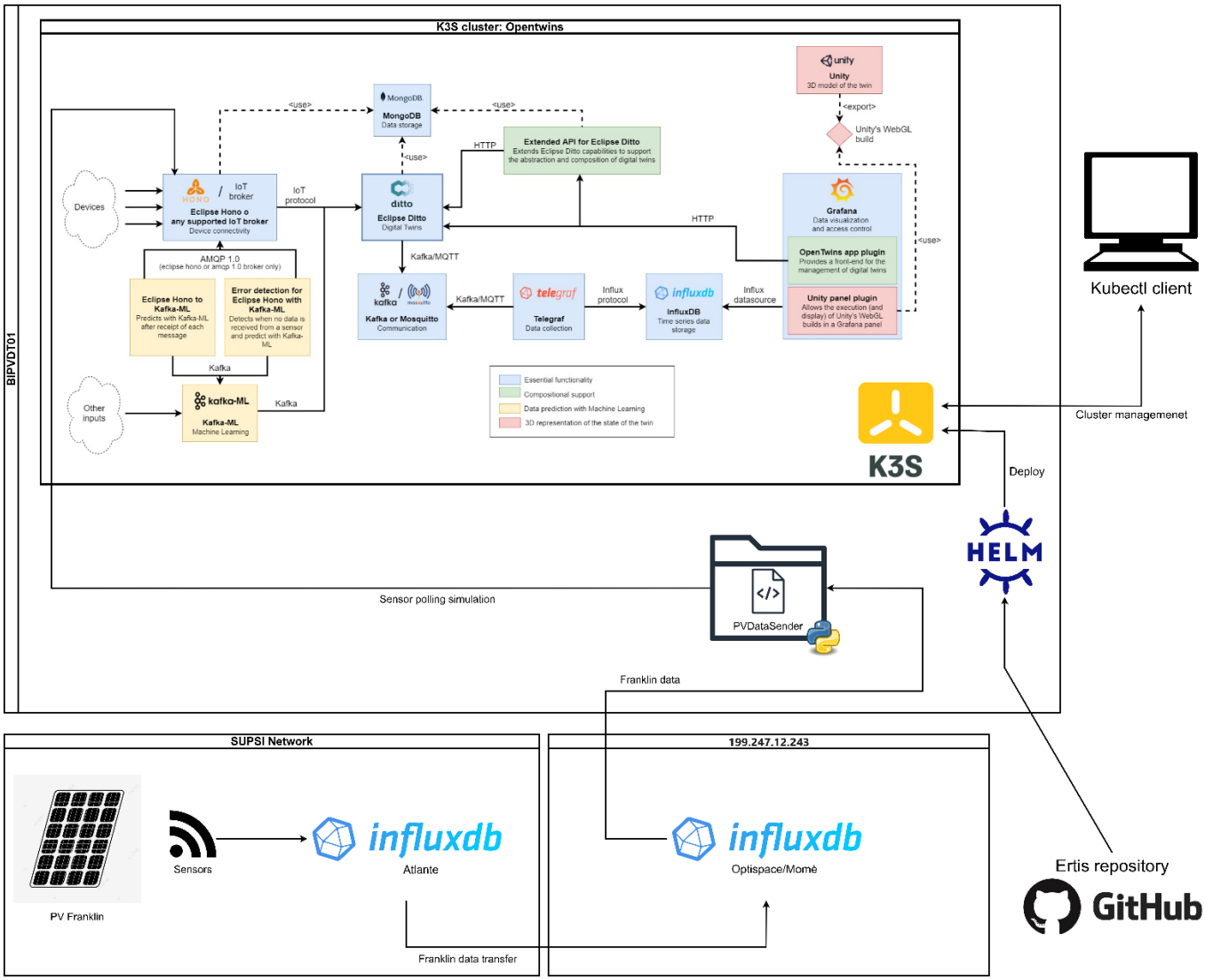
Franklin mockup DT in OpenTwins

In order to test OpenTwins with the target of implementing a BIPV digital twin, OpenTwins had been deployed on a single-node K3S cluster. K3S is a lightweight distribution of Kubernetes. The cluster had been installed on a CentOS virtual machine (4 vCPU, 8 GB memory, 80 GB disk).

The data used for the test were taken from SUPSI Franklin mockup. Those data are already collected on an InfluxDB instance used for other purposes. For this reason, in order to simulate sensors polling, a Python script (PVDataSender) was scheduled to fetch Franklin data from the external InfluxDB and send them to OpenTwins simulating a sensors polling process. The script oversees querying InfluxDB, prepares the Ditto messages with the obtained data and sends them to OpenTwins via MQTT. The script is scheduled, using crontab, to run every 60 minutes on the VM where OpenTwins is deployed. A sample for each measure is sent at each execution.

The InfluxDB instance from which data are fetched, exposed to internet, is not the direct collector of Franklin sensors' data. Indeed, there is another InfluxDB instance (named Atlante) which is the first stage for those data. Atlante is not directly exposed to internet. It can be accessed only through SUPSI network. Thus, Franklin data are automatically copied to the public InfluxDB that represents the OpenTwins data source.

The described infrastructure is summarized in the schema below.



Virtual machine references

Public hostname: 37-156-47-96.dc4-a.pub2.etik-cloud.com

Public IP: http://37.156.47.96/

Provider: Infomaniak

Linux user: cloud-user (sudo can performed with this user)

OpenTwins services references

Grafana: http://37.156.47.96:30718/

InfluxDB: http://37.156.47.96:30716/

Ditto: http://37.156.47.96:30525/

Ditto API: http://37.156.47.96:30526

Mosquitto: 37.156.47.96, port: 30511, topic on which Franklin data are sent: telemetry/

Credentials can be shared separately. Infomaniak VM can be accessed via SSH using a private key. The virtual machine management can be done using the Infomaniak Openstack console.

Troubleshooting

In case of problems, a Docker and K3s restart can be useful. From the cluster node:

sudo -i

systemctl restart docker

systemctl stop k3s

systemctl start k3s

If one of the OpenTwins service get stuck, kill the corresponding pod. From an external machine (with kubectl installed and configured to reach the k3s cluster).

Get pod status: *kubectl get pod*

Delete pod: *kubectl delete pod <pod name>*

Example: *kubectl delete pod opentwins-mosquitto-74f848fc8-wkbp9*

OpenTwins upgrade

When a new version of OpenTwins is released on GitHub (https://github.com/ertis-research/Helm-charts) the following steps need to be performed to deploy the new version from the cluster node.

- Take a snapshot of the VM
- *sudo -i*
- check if the repository is still configured: *helm repo list*
- https://ertis-research.github.io/Helm-charts/ should appear in the list
- If the repo is not present: *helm repo add ertis https://ertis-research.github.io/Helm-charts/*
- Check the current version of OpenTwins (check chart and app version): *helm ls*
- Upgrade OpenTwins: *helm upgrade opentwins ertis/OpenTwins --dependency-update*
- Check if OpenTwins had been upgraded (check chart and app version): *helm ls*
- Check everything is still running (from the kubectl machine): *kubectl get pod*

Data model

The digital twin creation in OpenTwins is done with types and things. Using the Grafana plugin, it is possible to create and manage them. Automatically, Ditto types and things are created accordingly, and well-formatted Ditto messages can populate the digital twin with automatic InfluxDB data storage. Ditto types define the characteristics of an element (static attributes, dynamic measurements and relationships with other types). Things are instances of types.

Franklin had been modeled as follows.

Types

- PV system:
  - Global PV entity
  - no measurements
- PV module
  - Child of PV system
  - No measurements
- MPPT
  - Child of PV system
  - Static attribute: position
  - Measurements:
    - Vm
    - Im
- Pyranometer
  - Child of PV system
  - Static attribute: position
  - Measurements:
    - Irradiance
- Semi PV module
  - Child of PV module
  - No measurements
- PV cell
  - Child of Semi PV module
  - Measurements:
    - Tbd
    - Tbom
    - Tair

Things

- 1 PV system
- 4 PV modules (L2, L3, L4, L5)
- 3 MPPTs (MPPT246, MPPT286, MPPT287)
- 3 Pyranometers (Pyr1, Pyr2, Pyr3)

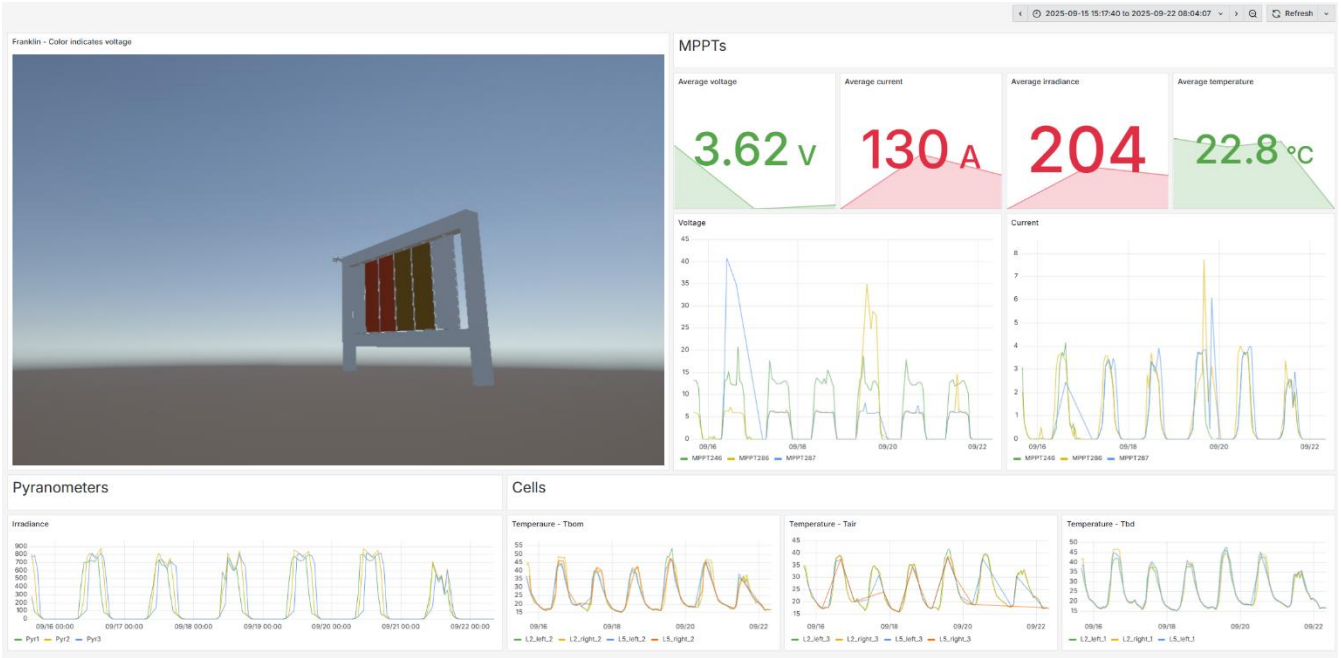
- 8 PV semi modules (L2_left, L2_right, L3_left, …, L5_right)
- 48 PV cells (L2_left_1, L2_left_2, …, L5_right_6), numeric indexing starts spatially from the top to the bottom of the module

Each of the measurements corresponds to a single data point fetched from the sensor and sent in an MQTT message to Ditto. The same data model is automatically created and populated by OpenTwins in its InfluxDB instance.

Dashboard

The Grafana dashboard on which both sensors' data and 3D twin of the PV system are visualized is under development. Time series are currently displayed in charts and other Grafana visualization panels such as KPIs. In addition, the OpenTwins plugin for Unity is employed to render the 3D model of Franklin. The render is navigable through mouse commands and data can be sent from Grafana to Unity to control the object dynamically based on live data. At the moment, the render is configured to color the various PV modules based on the average voltage value in the time period selected on Grafana.

Below the dashboard at the current development status.



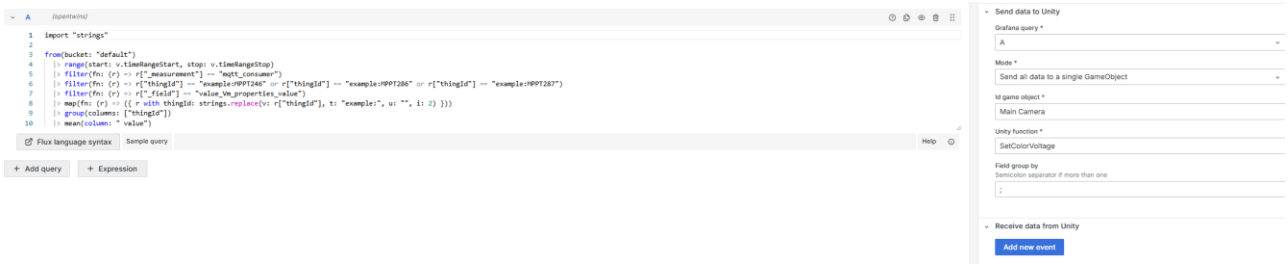
Unity

In order to render the object in Grafana, a Unity project should be created. In our case, a Revit model was available. Revit offers a plugin (free in trial version) to create an export that can be imported into Unity (https://apps.autodesk.com/RVT/en/Detail/HelpDoc?appId=3759955758891315427&appLang=en&os=Win64). Using the same strategy, it could be possible to do the same starting from an IFC file. The export process returns an object file that reproduces the structure of the Revit project, thus each element defined in Revit is represented as an independent game object in

Unity. Once the export is imported into Unity, it is necessary to write all the scripts needed to manage the object. In our case, the following features have been implemented until now:

- Rotation: left button (keep the button pressed and move)
- Translation: right button (keep the button pressed and move)
- Zoom in/out: mouse wheel up/down
- Keyboard input capture preventing (needed to make the Unity Grafana panel works, see: https://ertis-research.github.io/opentwins/docs/guides/unity/base-config)
- Color module/semi-module based on voltage value.

The OpenTwins Grafana plugin can call a method defined in Unity for a game object and pass to it a string parameter containing the JSON with the results of one of the queries defined in the panel (the desired query can be selected in the plugin configuration). The plugin requires the following parameter configuration:



Since the data are sent to a single game object, while, for example, voltage values are present for multiple modules, each represented by a different game object, there is a need to have a method that collects all the data sent from Grafana, and after parsing them, calls the methods of all the involved game objects. In our case, a script associated with the Unity main camera is called from the Grafana plugin. The script parses the JSON that contains the voltage values of the various modules and calls a second script, defined for all PV modules, that, given the voltage value, changes the color of the game object. In this way, based on the live values collected and the time filter set on the dashboard, the 3D model is colored accordingly.

Once the Unity project is ready, the export should be done through a WebGL build, details at: https://ertis-research.github.io/opentwins/docs/guides/unity/base-config

Versioning

The GitHub repository BIPV OpenTwins (https://github.com/molinarinicolo/BIPVOpentwins) tracks the changes of the following component:

- Python scripts to handle data from Atlante InfluxDB to OpenTwins (“src” folder)
- Entire Unity project, comprehensive of all the C# scripts (libraries are not tracked, until now the usage is limited to the standard set)
- Grafana dashboard (the configurations of the dashboards are exported in JSON format and can be reimported in Grafana).
- This documentation