Wisdom Software

---ꕥ🕮ꕥ---

Báo Cáo Dự Án
Hybrid-Digital Twin Platform

THÀNH VIÊN THỰC HIỆN:

1. Thái Đặng Phương Nam
2. Nguyễn Lê Thanh Phong
3. Huỳnh Thanh Sơn

TP. HỒ CHÍ MINH, tháng 01 năm 2026

MỤC LỤC

RESEARCHING & DEPLOYING BASYX AAS WEB UI

Reporter: Thai Dang Phuong Nam **Date:** January 07, 2026 – January 13, 2026 **Topic:** BaSyx AAS Web UI (Viewer/Editor) and Backend Connection.

6 PARTS for Detail.

1. Goal and Scope

- **Goal:** To set up the BaSyx AAS Web UI, understand how the UI works, and test the function of uploading .aasx files.
- **Scope:**
  + Focus on **Web UI** and **connecting to Backend**.
  + Use a simple Backend (Docker) for quick testing.
  + *Note:* Other parts like Data Bridge, MongoDB, and full Java Server SDK will be researched by other team members.

2. Implementation Steps

2.1. Web UI (Frontend)

I used the official source code from Eclipse BaSyx.

- **Repository:** https://github.com/eclipse-basyx/basyx-aas-web-ui
- **How to run:**
  1. Clone the repository.
  2. Run command: npm install
  3. Run command: npm run dev
- **Result:** The interface runs successfully at http://localhost:3000.

2.2. Backend (API Provider)

First, I checked the basyx-java-server-sdk. However, the structure is quite complex to set up manually.

Solution: I decided to use **Docker Compose** with the pre-built aas-environment image. It is faster and easier for testing.

My docker-compose.yml file:

```
services:
  aas-env:
    image: eclipsebasyx/aas-environment:2.0.0-milestone-08
    container_name: basyx-backend-service-ver1-0
    ports:
      - "8081:8081"
    environment:
      # Fix CORS issue for UI (port 3000)
      - Basyx_Cors_Allowed-Origins=*
      - Basyx_Cors_Allowed-Methods=GET,POST,PUT,DELETE,OPTIONS
```

- **Backend API:** http://localhost:8081
- **Swagger UI:** http://localhost:8081/swagger-ui/index.html (I used this to check if the API is working).

3. Configuration (Infrastructure)

The BaSyx Web UI does not use a hard-coded server. We must configure the "Infrastructure".

- **Problem:** The default setting points to ports 9081-9084 (full stack with Registry), but my Docker runs on port 8081.
- **Solution:** I changed the configuration in the UI (or basyx-infra.yml) to point to http://localhost:8081.
- **Note:** I did not use Registry or Discovery services to avoid connection errors.

4. UI Features & User Flow

Based on my testing and the BaSyx Wiki, the interface has **3 main columns**:

4.1. Left Column: AAS List

- **Function:** Shows all Asset Administration Shells (AAS) on the server.
- **Upload:** I tested uploading the ExampleV3.aasx file. The system worked and showed "2 Shells".
- **Search:** We can search for an AAS by name.

4.2. Middle Column: Submodel Tree

- **Function:** When I click on an AAS, this column shows the structure.
- **Content:**
    + **AAS:** The main asset.
    + **Submodels:** Groups of data (e.g., Technical Data, Documentation).
    + **Elements:** Specific data points (Properties, Collections, Files).

4.3. Right Column: Details & Visualization

- **Function:** Shows the value of the selected element.
- **Element Details:** Shows ID, value type, and semantic ID.
- **Visualization:** It displays charts or tables if the data is suitable.
- **JSON View:** This tab shows the raw JSON data (very useful for developers).

5. Issues and Fixes

During the test, I found two main issues:

1. **404 Not Found Error:**
    + *Reason:* The UI tried to call Registry/Description services, but I only ran the Repository service.
    + *Fix:* Correct the Infrastructure config to match the running Docker container.
2. **CORS Error:**
    + *Reason:* The browser blocked the connection between port 3000 (UI) and port 8081 (Backend).
    + *Fix:* I added Basyx_Cors_Allowed-Origins=* to the Docker Compose file.

6. Conclusion & Proposal

Conclusion:

- I successfully set up the local environment.
- I understand the flow: Upload AASX -> Backend parses data -> UI displays data.
- The 3-column layout is clear and easy to use.

Proposal:

- **For Demo:** We should use the aas-environment Docker image because it is simple and fast.
- **For Production:** The backend team should research the full Java SDK and MongoDB to save data permanently.

2. Establishing a Persistent Storage Mechanism with MongoDB

2.1. Problem Statement and Solution Selection

In the default architecture of Eclipse BaSyx, Asset Administration Shells (AAS) and Submodels are typically stored in the volatile memory (In-memory) of the Docker container. This creates a significant risk of data loss regarding configuration and historical states whenever the system is restarted or encounters a failure.

To mitigate this issue, the system integrates **MongoDB** as the persistence layer. MongoDB was selected based on the following criteria:

- **Document-Oriented NoSQL Structure:** It aligns perfectly with the hierarchical JSON format of the AAS.
- **High Performance:** It provides high throughput for read/write operations, satisfying the continuous state update requirements of a Digital Twin.
- **Scalability:** It offers easy horizontal scaling as the number of monitored devices increases.

2.2. Technical Configuration

The BaSyx Java Server (Version 2) is configured to connect to MongoDB Atlas via environment variables defined in the `docker-compose.yml` file.

Docker Compose Configuration:

YAML

```
basyx-environment:
  image: eclipsebasyx/aas-environment:2.0.0-milestone-03
  environment:
```

```
    # Activate the MongoDB profile instead of In-memory
    - SPRING_PROFILES_ACTIVE=mongoDbStorage
    # Connection string to MongoDB Atlas (Credentials encrypted)
    -
SPRING_DATA_MONGODB_URI=mongodb+srv://<username>:<password>@cluster0
.mongodb.net/?retryWrites=true&w=majority
    - SPRING_DATA_MONGODB_DATABASE=DigitalTwinDB
  ports:
    - "8081:8081"
  volumes:
    - ./basyx_data:/application/data
```

2.3. Data Organization Results

Upon deployment, AAS data is automatically mapped into specific Collections within MongoDB:

- **`assetAdministrationShells`**: Stores asset identification and metadata.
- **`submodels`**: Stores the submodel structure and the actual values of properties.
- **`conceptDescriptions`**: Stores semantic definitions (Semantic IDs).

(Figure 2.1: MongoDB Atlas Interface displaying automatically generated collections)

3. Real-Time Data Integration with BaSyx DataBridge

3.1. Role and Functionality of DataBridge

The **BaSyx DataBridge** serves as a critical middleware component that facilitates interoperability between the physical layer (OT) and the virtual layer (IT). It bridges the

communication gap between edge devices utilizing IoT protocols (such as MQTT, OPC UA) and the Digital Twin system which operates primarily on HTTP/REST interfaces.

The data processing workflow follows the **ETL (Extract - Transform - Load)** model:

1. **Extract:** The system subscribes to and ingests raw telemetry data from the MQTT Broker.
2. **Transform:** It parses the payload, extracts specific data points, and transforms them into the standardized JSON format required by the AAS.
3. **Load:** The transformed data is propagated to the AAS Server via HTTP PATCH requests to update the corresponding Submodel elements.

3.2. Monitoring Scenario

For the purpose of this project, the system is configured to monitor the performance metrics of a workstation (PC). The data flow is defined as follows:

- **Data Source:** A Python script simulating a sensor node, generating CPU and RAM usage metrics.
- **Communication Protocol:** MQTT (Message Queuing Telemetry Transport).
- **Topic:** `device/pc001/telemetry`.
- **Target Destination:** An AAS with the `idShort` of `PC_Monitor` and a Submodel named `OperationalData`.

3.3. Routing Configuration

The logic for data mapping is defined in the `routes.json` configuration file. This file dictates how incoming messages are parsed and where they are directed within the Digital Twin.

Configuration File: `routes.json`

JSON

```
[
  {
    "datasource": {
      "type": "mqtt",
      "server": "tcp://mqtt-broker:1883",
      "topic": "device/pc001/telemetry"
    },
    "transformers": [
      {
        "type": "jsonata",
        "query": "$.cpu_usage"
        // Extracts the 'cpu_usage' value from the raw JSON payload
      }
    ],
    "datasinks": [
      {
        "type": "http",
        "url": "http://basyx-environment:8081/submodels/[BASE64_SUBMODEL_ID]/submodel-elements/CPUUsage/value"
        // API endpoint to update the CPUUsage property
      }
    ]
  },
  {
    "datasource": {
```

```
    "type": "mqtt",
    "server": "tcp://mqtt-broker:1883",
    "topic": "device/pc001/telemetry"
  },
  "transformers": [
    {
      "type": "jsonata",
      "query": "$.ram_usage"
    }
  ],
  "datasinks": [
    {
      "type": "http",
      "url": "http://basyx-environment:8081/submodels/[BASE64_SUBMODEL_ID]/submodel-elements/RAMUsage/value"
    }
  ]
 }
]
```

4. DT Data Model Building System

The DT Data Model Building System plays a core role in managing the lifecycle and maintaining data consistency between physical assets and their digital twins. This component is designed and implemented based on the **BaSyx AAS Environment**, strictly following the **IEC 63278** standard for the Asset Administration Shell (AAS). The primary goal is to provide a middle layer that enables secure, reliable information exchange and promotes interoperability within the Industry 4.0 ecosystem.

4.1. Functional Analysis and Architecture Alignment

Based on the **ISO 23247-2** reference model, the system is divided into digital resource management services. These correspond to the **Digital Twin entity (23247-DTE)** and the **Device Communication entity (23247-DCE)**. The main functions are mapped through Application Programming Interfaces (APIs) based on the BaSyx OpenAPI specification:

- **AAS Repository Service**: Centrally manages AAS entities. This service supports full CRUD (Create, Read, Update, Delete) operations for Shells, provides asset identification information (**AssetInformation**), and manages references to sub-models (**Submodels**).
- **Submodel Repository Service**: Responsible for managing the detailed data structure of the Digital Twin. Through parameters such as **idShortPath**, the system allows deep access and operations on individual sub-model elements (**SubmodelElement**), including technical properties, file attachments, and operational tasks.
- **Concept Description Service**: Ensures **semantic interoperability** across the entire system. This service manages standardized definitions to explain the meaning of data properties according to industrial dictionaries (such as ECLASS or IEC 61360). This helps third-party systems understand and process data accurately.

4.2. APIs and Interoperability Strategy

Interoperability is a top priority for this platform. The system achieves this through the following mechanisms:

- **RESTful Architecture and Standard Data Format**: All services are implemented on the **HTTP/REST** platform, using **JSON** as the primary format for data exchange. This ensures flexible integration with modern IT systems.
- **Serialization**: The system provides a **/serialization** API to package all or part of a Digital Twin model (including AAS, Submodels, and Concept Descriptions) into a single file. This supports data migration between different AAS servers in distributed production environments.
- **Model Initialization via Upload**: The **/upload** function allows users to upload configuration files in **XML**, **JSON**, or **AASX** formats. This mechanism enables a quick transition from technical design documents to an active **Reactive Digital Twin**, reducing manual errors and deployment time.

4.3. Technical Implementation and QoS Management

To meet non-functional requirements such as reliability and performance in industrial environments, the system applies the following technical management standards:

- **Result Schema**: To ensure **traceability**, all API responses follow a standard **Result** structure. This includes a list of **Messages** containing an error **code**, **text** content, a **timestamp**, and a **correlationId**. Message levels are clearly categorized (INFO, WARNING, ERROR, EXCEPTION) to assist in system diagnosis.
- **Paging and Query Optimization**: For large data retrieval requests (such as listing all Shells), the system implements a paging mechanism based on **cursor** and **limit**. Additionally, parameters such as **level** (deep/core) and **extent** (with/withoutBlobValue) allow users to customize the depth and volume of response data, optimizing network bandwidth and client application performance.
- **Deployment Environment**: In this implementation phase, the **BaSyx AAS Environment** component is deployed and configured at the server address:

http://172.21.211.227:8081. This server acts as the **Single Point of Access** for all interactions related to Digital Twin data models within the overall platform architecture.

4.4. Detailed API and Interaction Specifications

Based on the **BaSyx AAS Environment** architecture, the system provides not only basic storage services but also complex interaction mechanisms. This ensures maximum flexibility for various industrial applications.

4.4.1. Hierarchical Repository Services

The system is organized into separate Repository layers, allowing lifecycle management from a general to a detailed level:

- **AAS Repository API**: Manages asset identification "shells". Besides standard CRUD operations, it supports managing thumbnails and **submodel-refs**, allowing technical properties to be dynamically linked to a single Shell.
- **Submodel Repository API**: Manages the logical structure of data. A key feature is hierarchical access via the **idShortPath** (separated by dots), which allows precise retrieval of nested elements within complex models.
- **Concept Description Repository API**: Acts as a "semantic dictionary". It allows looking up standardized definitions for each property to ensure data has consistent technical meaning across different systems.

4.4.2. Value-Only Representation

One of the most advanced features is the support for **Value-Only** representation via endpoints ending in **/$value**.

- **Purpose**: Minimizes data transfer by returning only the actual values of properties without extra metadata.
- **Application**: This mechanism is highly effective for real-time status updates from IoT devices or sensors with limited bandwidth.

4.4.3. Polymorphic Submodel Elements

The system supports a wide range of **Submodel Elements** to meet all physical asset modeling needs:

- **Static and Dynamic Data**: Includes **Property** (single value), **MultiLanguageProperty** (multi-language descriptions), and **Range** (value intervals).
- **Binary Data and Files**: Supports **Blob** (direct binary data) and **File** (external file links), managed through a specialized **/attachment** interface.
- **Complex Structures**: Allows grouping elements using **SubmodelElementCollection** or **SubmodelElementList** to create high-detail data models.

4.4.4. Operation Invocation

Beyond static data storage, the system allows "activating" asset functions through the **invoke** interface:

- **Request Structure**: Operations are defined with a list of **inputArguments**, **outputArguments**, and **inoutputArguments**.
- **Execution Mechanism**: Supports both synchronous and asynchronous modes depending on task complexity and device response time. This is controlled via the **async** parameter in the query.

Summary of Key Functional Endpoints and Technical Roles

API Group	**Method**	**Main Endpoint**	**Role in the Digital Twin System**
Shell Management	PUT	/shells/{aasIdentifier}/asset-information	Updates identification and core asset

			properties.
Value Management	PATCH	/submodels/.../submodel-elements/{idShortPath}/$value	Updates real-time values for sensors or machine status.
Operation	POST	/submodels/.../submodel-elements/{idShortPath}/invoke	Executes remote control commands to physical devices.
File	GET/PUT	/submodels/.../submodel-elements/{idShortPath}/attachment	Manages technical documents and operation manuals for assets.
System	GET	/description	Provides self-describing information about supported Profiles and standards.

To further enhance the technical documentation, this section provides a comprehensive overview of the interface capabilities. The BaSyx AAS Environment offers a robust set of RESTful APIs designed to manage the entire lifecycle of Asset Administration Shells and their submodels. These endpoints allow seamless interaction between the digital and physical worlds, supporting everything from high-level registry lookups to low-level data element updates.

The following table lists all available endpoints in the system, categorized by their functional roles to provide a clear map for developers and system integrators.

4.5. Comprehensive API Endpoint Reference

Functional Group	Method	Endpoint	Description
AAS Repository API	GET	/shells	Retrieves a list of all Asset Administration Shells.
	POST	/shells	Registers a new Asset Administration Shell in the repository.
	GET	/shells/{aasIdentifier}	Returns the details of a specific Asset Administration Shell.
	PUT	/shells/{aasIdentifier}	Updates an existing Asset Administration Shell's data.

	DELETE	/shells/{aasIdentifier}	Removes an Asset Administration Shell from the system.
	GET	/shells/{aasIdentifier}/asset-information	Retrieves information about the physical asset linked to the AAS.
	PUT	/shells/{aasIdentifier}/asset-information	Updates the asset-specific metadata.
	GET	/shells/{aasIdentifier}/asset-information/thumbnail	Retrieves the asset's thumbnail image.
	PUT	/shells/{aasIdentifier}/asset-information/thumbnail	Uploads a new thumbnail for the asset.

	DELETE	/shells/{aasIdentifier}/asset-information/thumbnail	Deletes the existing thumbnail.
	GET	/shells/{aasIdentifier}/submodel-refs	Lists all submodel references associated with the AAS.
	POST	/shells/{aasIdentifier}/submodel-refs	Adds a new submodel reference to the AAS.
	DELETE	/shells/{aasIdentifier}/submodel-refs/{submodelIdentifier}	Removes a specific submodel reference without deleting the submodel itself.
	GET	/submodels	Retrieves all submodels

Submodel Repository API			hosted in the environment.
	POST	/submodels	Creates and stores a new submodel.
	GET	/submodels/{submodelIdentifier}	Returns the full structure and data of a specific submodel.
	PUT	/submodels/{submodelIdentifier}	Overwrites an existing submodel with new data.
	DELETE	/submodels/{submodelIdentifier}	Deletes a submodel from the repository.

	GET	/submodels/{submodelIdentifier}/$metadata	Retrieves only the metadata of a specific submodel.
	GET	/submodels/{submodelIdentifier}/$value	Returns submodel data in a simplified "Value-Only" format.
	PATCH	/submodels/{submodelIdentifier}/$value	Updates submodel values using the Value-Only representation.
	GET	/submodels/{submodelIdentifier}/submodel-elements	Lists all elements within a submodel, including hierarchy.

	POST	/submodels/{submodelIdentifier}/submodel-elements	Adds a new element to the submodel.
	GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}	Retrieves a specific element using its unique path.
	PUT	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}	Updates a specific submodel element.
	POST	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}	Creates a submodel element at a specified hierarchical path.
	DELETE	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}	Deletes a submodel element at a

			specified path.
	GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value	Retrieves a specific element's value only.
	PATCH	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value	Updates the value of a specific submodel element.
	POST	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke	Triggers a function or operation defined in the submodel.
AAS API (File/Attachment)	GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment	Downloads file content from a specific File or Blob element.

	PUT	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment	Uploads or updates the file content for a submodel element.
	DELETE	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment	Removes the file content associated with an element.
Concept Description API	GET	/concept-descriptions	Lists all semantic concept descriptions.
	POST	/concept-descriptions	Registers a new concept description.
	GET	/concept-descriptions/{cdIdentifier}	Returns details of a specific concept.

	PUT	/concept-descriptions/{cdIdentifier}	Updates an existing concept definition.
	DELETE	/concept-descriptions/{cdIdentifier}	Removes a concept description from the system.
Environment API	POST	/upload	Processes and imports XML, JSON, or AASX environment files.
Serialization API	GET	/serialization	Exports AAS and submodel data in a serialized format.

Discovery Interface	GET	/description	Provides information about the network resource's capabilities.

4.6. Detailed Structure of Common Schema Objects

4.6.1. Result Object (System Response)

This is the most frequently utilized object in the API (appearing 228 times), acting as the standard wrapper for all API responses.

- **Main Structure**:
  - **messages**: A list (array) of Message objects that provide feedback on the request status.
- **Message Object Details**:
  - **timestamp**: The exact time the event or error occurred.
  - **text**: A detailed description of the message.
  - **code**: A specific identifier for the error or status.
  - **messageType**: Categorizes the message as ERROR, WARNING, INFO, EXCEPTION, or UNDEFINED.
  - **correlationId**: An ID used to trace the process flow across a distributed system.

4.6.2. Asset Administration Shell - AAS

The AAS represents the Digital Twin entity at the highest administrative level.

- **Main Structure**:

+ **id**: The unique global identifier for the AAS.
+ **idShort**: A brief, logical name used within the system hierarchy.
+ **assetInformation**: Contains metadata about the physical asset, such as the globalAssetId and assetKind.
+ **submodels**: A list of Reference objects pointing to associated functional submodels.
+ **administration**: Administrative data including version and revision details.
+ **displayName / description**: Multi-language strings for user-friendly identification and details.

4.6.3. Submodel

A functional block containing specific technical or operational data for an asset.

– **Main Structure**:
  + **id / idShort**: The unique identifier and logical name of the submodel.
  + **semanticId**: A reference to a standardized semantic definition (e.g., IEC 61360), ensuring other systems can interpret the content.
  + **submodelElements**: An array of specific data elements like Properties, Operations, or Files.
  + **kind**: Specifies if the submodel is an INSTANCE or a TEMPLATE.
  + **qualifiers**: Additional conditions or constraints applied to the data within the submodel.

4.6.4. SubmodelElement

The "atomic" data units that form the content of a Submodel.

– **Common Structure**:
  + **idShort**: The identifier of the element within the scope of its submodel.
  + **semanticId**: Defines the technical meaning of the specific element.

+ **value**: The actual data stored in the element (for Properties, Files, or Blobs).
+ **category**: Categorizes the element, such as a PARAMETER or VARIABLE.

4.6.5. Reference and Key

The linking mechanism used to connect different components within the Digital Twin ecosystem.

- **Reference Structure**:
  + **type**: Specifies if it is a MODEL_REFERENCE (internal) or an EXTERNAL_REFERENCE.
  + **keys**: A list of Key objects used to resolve the path to the target object.
- **Key Structure**:
  + **type**: The type of the referable element (e.g., SUBMODEL, PROPERTY, AAS).
  + **value**: The identifier value of the key.

Table 3: Summary of Key Schema Attributes and Technical Roles

Schema	**Critical Attribute**	**Data Type**	**Technical Purpose**
AAS	submodels	Array of Reference	Connects the Shell to functional data sets.
Submodel	submodelElements	Array of Elements	Stores technical specifications and operational states.

Result	messages	Array of Message	Provides status feedback for client applications.
Reference	keys	Array of Key	Defines the retrieval path to other digital objects.
Operation	inputVariables	Array of Variables	Defines input parameters for executable functions.