# Orchestration Of Model Computing Assets For The Development Of Digital Twins

https://doi.org/10.31713/MCIT.2023.072

Oleksandr Stepanets
Department of Automation of Energy Processes
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine

Anastasia Zakharchenko
Department of Automation of Energy Processes
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
zakharchenko.anastasia@lll.kpi.ua

*Abstract*—**With the development of industries and manufacturing requirements, the use of digital twin technology is promising for increasing efficiency and improving product quality. An important component of digital twin implementation is the provision of tools for modeling and simulation processes. In this study, a solution based on the use of Python-based models and the Controller software platform, which provides communication between models and external elements of digital twins and other IIoT solutions, was proposed as such a toolkit. It defines the main approaches for creating model, scenario, and function libraries, along with the concept of implementing dynamic scenarios based on JSON requests from the orchestrator. In addition, the algorithm, the structure of the Controller, and its main functionality are considered. The resulting solution takes advantage of the Python and Node.js languages in performing the relevant tasks and contains several solutions that reduce traffic and workload on communication systems, increase system reliability, scale the system, etc.**

**Keywords—Digital Twins; Python; Node.js; Modeling; Simulation; IIoT.**

## I. INTRODUCTION

Industrial automation systems are becoming increasingly complex for various industries. The requirements for energy efficiency, environmental friendliness and reliability, and product quality are rising [1]. The use of modern technologies, such as artificial intelligence, data-driven technologies, digital twins, and IIoT, is aimed at ensuring progress. Also, it promotes the development of the modeling and simulation field, not only in terms of studying the object's behavior but also as a full-fledged tool necessary to analyze the system's operation in near real-time. Modeling is an integral part of what-if analysis, decision support systems, energy consumption planning, virtual sensors, etc., which are part of modern intelligent user support systems and could be integrated directly into control systems.

The use of cloud technologies is one of the main concepts of IIoT and the development of digital twins, allowing the storage, processing, and transmission of huge amounts of data generated by sensors and services [2, 3]. The use of cloud computing and analytics makes it possible to obtain detailed information about the state of an object in real-time, using the computing power required at the moment. Implementation of modeling capabilities based on cloud applications allows the use of their functionality as a service with the ability to deploy the required number of modules.

Based on these conditions, this article investigates the organization of Python-based mathematical model simulation processes using the available tools of the Node.js platform as a cloud service.

## II. GENERAL CONCEPT

Developing highly detailed models of technological processes or their elements and creating their computer code representation raises the issue of implementing research results into industry, incorporating them into analytics, and dynamically comparing the performance of certain equipment with the expected results. The integration of computer models IIoT-based solutions and the design of digital twins has led to the challenge of developing a special software platform that manages and coordinates modelling, analytics, etc.

For this purpose, IIoT and digital twin solutions will be considered as a set of special modules or services that are managed and synchronized by a special orchestrator [3]. Thus, in this paper, we focus on the encapsulated, independent part of the "orchestrator-computing module" of a distributed system. Such separation allows for improving the system's reliability, given that in case of errors or loss of communication on one of the modules, it does not cause the collapse of the entire system.

There are a number of requirements for the model Controller platform's development, which are based on the specific conditions of our task and skills, as well as questions about future project growth. These include:

- support for computer models created in Python;
- security of data transmission;
- project scalability;
- the ability to exchange data between platforms and services;
- the ability to configure the platform according to the user's objects and tasks;

- universality of solutions and the ability to additionally use them to implement analytical codes;
- support for real-time data transfer;
- control of calculations, logging, alarming and reporting of possible errors.

Accordingly, certain functions are also distributed between them (Fig. 1).

## III. PYTHON-BASED MODELS

Different model types can be used to describe the behavior of objects, including static and dynamic mathematical models. At the same time, one object can have several models that use different approaches, accuracy, scale, etc. [4]. Thus, the platform can have a library of object models, their elements, certain internal processes, etc. Their combination enables modeling more complex processes and creates a number of possible scenarios. The models working is supported by a library of additional functions, including data conversion, transmitting to other software applications, averaging, related Python libraries and modules, post-processing etc. (Fig. 2). In particular, the related NumPy and SciPy modules can be included.

The model library is a set of classes that contain the mathematical description of an object or its element and functions that allow one to calculate static and dynamic models depending on the request. When a particular query is executed, the orchestrator launches the corresponding modeling script, which may include a certain arrangement of elements from the class library and function library. As a first approximation, such scenarios were written statically as separate Python scripts.

A special type of scenario is real-time work based on the Mixed Discrete-Continuous Simulation approach [5] and employing a fixed-time algorithm. In general, the algorithm is described in Fig. 3.

Although this algorithm and the static description of scenarios have shown a stable operation, to implement modularity and integration of Digital Twin functions, automatic scenario layout, and improve system flexibility, we observe the need to implement dynamic scenario layout by the orchestrator's request. In this case, the orchestrator forms the structure of the scenario in the form of a JSON [3] with the specified initial conditions of the system based on the information about the models. During initialization, the Scenario class accepts a set of model classes, their arguments, and a description of its structure as a variable-length argument list (**kwargs), creating a corresponding set of instances and using sequential calls to their functions.

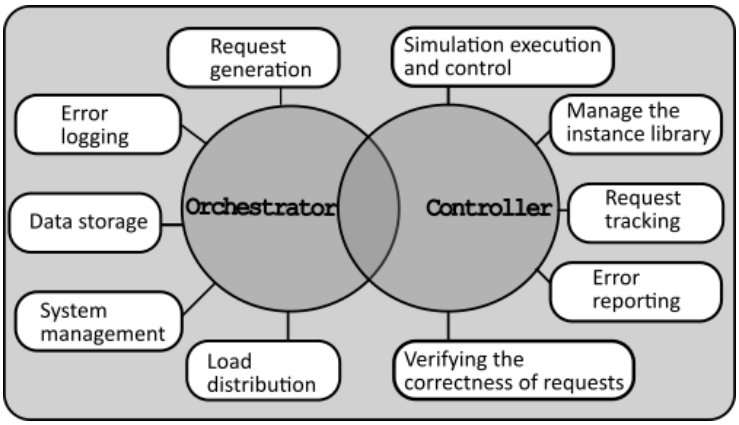


Figure 1. Functionality of Orchestrator and Controller sides

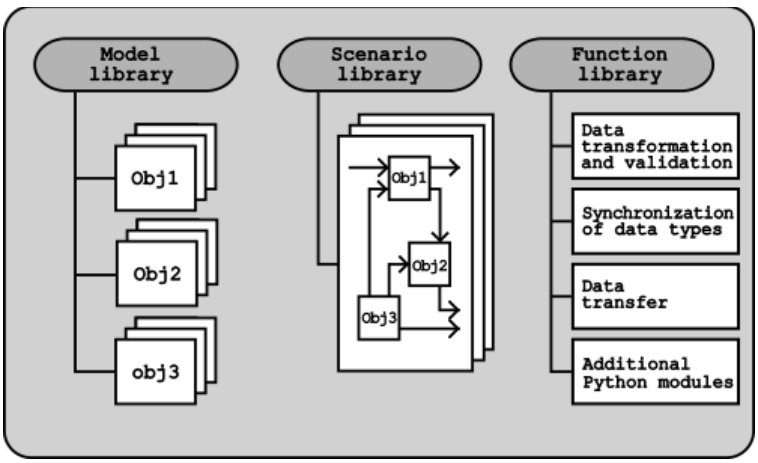


Figure 2. Basic structure of Python-based models realization

```
•{module and model library import}
•{initialization of state variables}
•{model initialization}
while True:
   •{current sampling time
     calculation}
   •{update model status}
   •{new data processing and storage}
   •{result transmission}
   •{waiting for the next iteration
     using the time.sleep() function}
```

Figure 3. Algorithm of real-time process calculation

## IV. NODE.JS-BASED PLATFORM

In developing software to support modeling processes, Node-Red was initially considered to be a tool for managing data flows [6]. However, due to the need to use different environments and to avoid conflicts between specific environment nodes related, for example, to their updates, as well as the necessity to install additional software, organize access to the file system, etc. it was decided to use node.js as the main platform for developing a universal platform.

In the proposed solution the MQTT broker is used to communicate between the Controller and the Orchestrator and to transmit data further to the target destination. Each Controller has its own unique identifier and monitors a specific topic for the presence of commands addressed to it. After receiving a message, the Controller evaluates the correctness of the received command in accordance with the functionality allowed by the platform deployment, the list of accounting objects in the instance library, the required models and analytical functions in the library, and the availability of the necessary arguments for them. If there are conflicts between requests and the Controller's capabilities, it returns an error with a corresponding message that is logged by the orchestrator. Otherwise, the corresponding script is launched as a child process, the results of which are returned as a ready-to-send JSON message to the Data Sender and sent via the MQTT Client (Fig. 4). The use of centralized data transfer allows you to separate the calculation process and reduce the number of clients for the MQTT broker compared to direct access to communication tools by Python script. This enables additional data verification before sending and the ability to save work results in case of a connection loss.

The Controller manages its instance library, which contains the main design characteristics of equipment and processes that do not change over time and are necessary for modeling. The purpose of this library is to reduce traffic between elements of a branched system and to keep track of objects that are within the area of responsibility of a particular Controller.

One object instance could be duplicated in the libraries of several separate Controllers, which will allow the Orchestrator to distribute the computing workload more evenly and improve the reliability of the distributed system by using redundancies in case of a loss of functionality or communication of one of the platforms.

## V. FUTURE STEPS

The proposed system has a number of conceptual solutions that form the basis and, in our opinion, are prospective for further research. The following studies focus on optimizing the performance of computational codes and continuing work on dynamic scenario composition to develop a single general approach to describe heterogeneous object models. Also, we are working to expand the library of models and analytical functions for conducting experimental studies on real objects.

In addition to the direct improvement of the system itself, it is planned to develop solutions for integrating the work results into control systems to obtain practical benefits and evaluate them in the field.

## VI. CONCLUSION

In this paper, we considered approaches to the realization of the modeling and simulation of processes as a separate module that can be used as a service for various IIoT solutions and the implementation of digital twin functions. The proposed solutions have a modular nature, which enables them to distribute traffic, provide high response speeds, and parallelize information flows. They allow for high reliability of the developed system in case of off-design errors, loss of communication or system performance due to redundant computing capabilities on other platforms, data storage until the ability to transmit data is restored, etc.

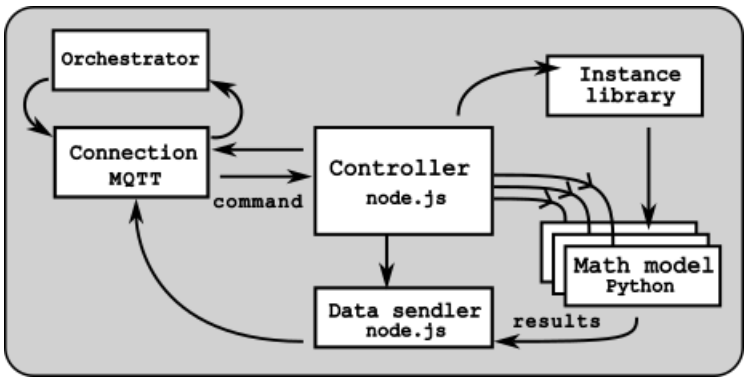


Figure 4. . Basic structure of Node.js-based Controller

The platform takes advantage of the Python and Node.js languages to perform relevant tasks and contains a number of solutions that reduce traffic and the load on communication systems, such as organizing centralized communication with an MQTT broker and creating its own library of object instances.

The use of a distributed system for modeling objects has high prospects. Therefore, we are planning to continue research in the area of optimizing computational loads, developing a model library, and integrating the results into control systems.

## REFERENCES

[1] Z. Gao, "Special Issue on Modelling, Monitoring, Control and Optimization for Complex Industrial Processes," Processes, 2023. 10.3390/pr11010207.

[2] K. Agalianos, S. Ponis, E. Aretoulaki, G. Plakas, O. Efthymiou, "Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics," Procedia Manufacturing, vol. 51(2), 2020, 1636–1641. 10.1016/j.promfg.2020.10.228.

[3] G. De Giacomo, D. Ghedallia, D. Firmani, F. Leotta, et al, "IoT-based Digital Twins Orchestration via Automated Planning for Smart Manufacturing," Workshop on Generalization in Planning (GenPlan), 2021.

[4] U. Dahmen and J. Rossmann, "Experimentable Digital Twins for a Modeling and Simulation-based Engineering Approach," 2018 IEEE International Systems Engineering Symposium (ISSE), Rome, Italy, 2018, pp. 1–8, doi: 10.1109/SysEng.2018.8544383.

[5] N. Karanjkar and S.M. Joshi, "A Python-Based Mixed Discrete-Continuous Simulation Framework for Digital Twins," In: International Conference on Simulation and Modeling Methodologies, Technologies and Applications. Cham: Springer International Publishing, 2021, pp. 204–223.

[6] C. Steinmetz, G.N. Schroeder, A. Binotto, S. Panikkar et al, "Digital Twins modeling and simulation with Node-RED and Carla," IFAC-PapersOnLine, vol. 55(19), 2022, pp. 97–102. https://doi.org/10.1016/j.ifacol.2022.09.190.