D4.16– i4Q Digital Twin v2

WP4 – BUILD: Manufacturing Data Analytics for Manufacturing Quality Assurance

i4Q has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958205.

Document Information

GRANT AGREEMENT NUMBER	958205	ACRONYM	i4Q	
FULL TITLE	Industrial Data Services for Quality Control in Smart Manufacturing			
START DATE	01-01-2021	DURATION	36 months	
PROJECT URL	https://www.i4q-project.eu/			
DELIVERABLE	D4.16– i4Q Digital Twin			
WORK PACKAGE	WP4 – BUILD: Manufacturing Data Analytics for Manufacturing Quality Assurance			
DATE OF DELIVERY	CONTRACTUAL	31-Dec-2022	ACTUAL	30-Dec-2022
NATURE	Report	DISSEMINATION LEVEL	Public	
LEAD BENEFICIARY	IKERLAN			
RESPONSIBLE AUTHOR	Hector Martin Aguilar, Alex de la Puente, Angel Rodriguez, Rafael Espadas, Urko Leturiondo (IKERLAN)			
CONTRIBUTIONS FROM	1–CERTH, 2–ENG, 8–BIBA			
TARGET AUDIENCE	1) i4Q Project partners; 2) industrial community; 3) other H2020 funded projects; 4) scientific community			
DELIVERABLE CONTEXT/ DEPENDENCIES	This document presents a technical overview of the Digital Twin solution (i4Q^{DT}).			
EXTERNAL ANNEXES/ SUPPORTING DOCUMENTS	None			
READING NOTES	None			
ABSTRACT	This document is a Technical Specification document about the development of the i4Q Digital Twin (i4Q^{DT}). This document provides a thorough description and analysis of the functionalities, features, and the current implementation status. It provides an in-depth technical overview of the principal functional sub-components (i.e., features) of the Solution.			

Document History

VERSION	ISSUE DATE	STAGE	DESCRIPTION	CONTRIBUTOR
0.1	7-Nov-2022	ToC	ToC created and sent for review	IKERLAN
0.2	25-Nov-2022	1st Draft	First draft sent for internal review	IKERLAN
0.3	02-Dec-2022	Internal Review	Internal Review	IBM, TIAG
0.4	09-Dec-2022	2nd Draft	Addressing reviewers' comments	IKERLAN
1.0	30-Dec-2022	Final Document	Final quality check and issue of final document	CERTH

Disclaimer

Copyright message

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS/ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interfaces
CNC	Computer Numerical Control
CPU	Central Processing Unit
DT	Digital Twin
FMU	Functional Mock-up Unit
ML	Machine Learning
RA	Reference Architecture
REST API	RESTful API
STL	STereoLithography
UI	User Interface

1. Executive summary

This document presents a technical explanation of the **i4Q Digital Twin** (i4Q^{DT}) Solution providing the general description, the technical specifications, and the implementation status. This is an updated deliverable with respect to its previous version D4.8 Digital Twin. The deliverable **D4.16** points to the Source Code of the i4Q^{DT} Solution that is in a private repository of Gitlab: https://gitlab.com/i4q.

The documentation associated with the i4Q^{DT} Solution is deployed on the website http://i4q.upv.es. This website contains the information of all the i4Q Solutions developed in the project "Industrial Data Services for Quality Control in Smart Manufacturing" (i4Q). The direct link to the i4Q^{DT} Solution documentation is http://i4q.upv.es/16_i4Q_DT/index.html.
The documentation is structured according to:

- General description
- Features
- Images
- Authors
- Licensing
- Pricing
- Installation requirements
- Installation Instructions
- Technical specifications of the solution
- User manual

2. Document structure

Section 3: Contains a general description of the **i4Q Digital Twin** solution providing an overview and a list of features. It is addressed to the end users of the i4Q Solution.

Section 4: Contains the technical specifications of the **i4Q Digital Twin** solution, providing an overview and an architecture diagram. It is addressed to software developers.

Section 5: Details the implementation status of the **i4Q Digital Twin** solution, explaining the current status, next steps and summarizing the implementation history.

Section 6: Provides the conclusions.

APPENDIX I: Provides the PDF version of the **i4Q Digital Twin** web documentation, which can be accessed online at: **http://i4q.upv.es/16_i4Q_DT/index.html**

3. General Description

3.1 Overview

i4Q^{DT} (Digital Twin Simulation Services) allows industrial companies to achieve a connected 3D production simulation, with a digital twin for manufacturing enabling virtual validation/visualisation and productivity optimisation using data from different factory levels (small cell to entire factory).

i4Q^{DT} provides a virtual representation and contextualization of all the assets present in a manufacturing line. The virtual representations of the different physical devices are accessible through APIs, creating a framework of consistent interoperability that allows the building of the digital twins reducing the complexity of IoT deployments. Additionally, when virtual sensors are to be obtained, physics-based models are developed. An industrial model exchange standard is used for facilitating the integration of models with monitoring algorithms, protecting the model intellectual property, and making the framework independent from the modelling source.

3.2 Features

- Provides the capability of building models and establishing the relationships between the inputs of the model and the collected data (contextualization).
- Provides the capability of loading, storing and updating individual models representing the different sections and machines of a plant.
- Provides the capability of running simulations of both data-driven models (machine learning python models) and physics-based models (FMU compiled models).
- Provides the capability of storing and visualizing the results from the simulations.

4. Technical Specifications

4.1 Overview

i4Q^{DT} is a deployed as a docker container allowing to launch simulations of a manufacturing asset/plant based on production/machine data and their digital twin (DT) and obtain results that are visualized in different data visualization formats.

Each DT allows calls to perform simulations that can be managed through REST APIs. Internally it can make use of, for example, Functional Mock-up Units (FMU) for physics-based models based on modelling languages like Modelica, and data-based models libraries like TensorFlow, Keras, Sklearn. For the 3D visualization of the results, tools such as Godot or Unity can be used. In general, for the visual representation of the results of i4Q^{DT} tools like Shiny, Tkinter or, Plotly can be used.

This solution provides inputs to other solutions such as the i4Q Prescriptive Analysis Tool and the i4Q Line Reconfiguration Toolkit.

4.2 Architecture Diagram

The processes and services that are being included in the i4Q^{DT} software tool are mapped to two tiers in the i4Q Reference Architecture: the Platform Tier and the Edge Tier, as can be viewed in **Figure 1**.

Considering that it is a solution found at various levels of the architecture:

- **Platform Tier**: The service that is being used in this tier is the “Digital Twin Services”. Here the DT will be offering simulations to other solutions as a service, in order for the results to be exploited by them with any other purpose.
- **Edge Tier**: The i4Q^{DT} solution is comprised of the “Digital Twin” and the “Data Collecting” services, giving the DT the ability to represent with high fidelity the specific component, asset, or plant.

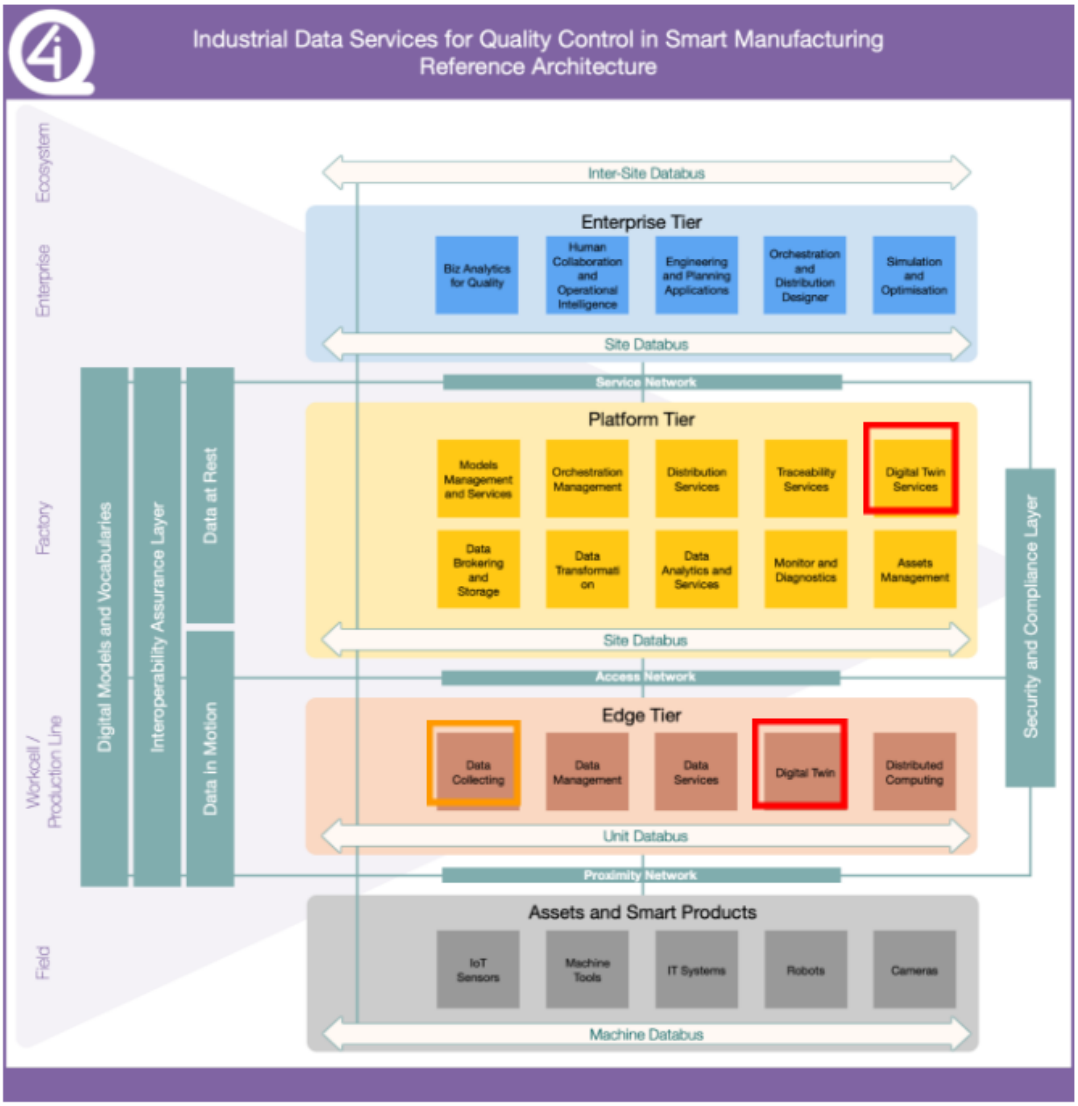

Figure 1. i4Q RA mapping with i4Q^{DT}

5. Implementation Status

5.1 Current implementation

The i4Q^{DT} is comprised of three main software packages: physics-based back end, data-driven back-end and user interface back-end.

In the following sections the current implementation of all these packages within the i4Q^{DT} Solution are described:

5.1.1 Physics-based back-end

The physics-based workflow makes use of Functional Mock-up Units (FMU) that have been compiled from different modelling languages like Modelica. These languages which are component-oriented and based on a set of equations defining the physics behaviour of the system [1].

A Python library (fmiSim) has been developed that contains two main classes: Model and Master. The class Model is used to store the information contained in a FMU file and to give access to its functionality. The class Model is used through two derived classes: ModelCS and ModelME, for fmu CoSimulation FMUs and ModelExchange FMUs respectively. The class Master is an implementation of an orchestrator to carry out CoSimulations. It takes a set of models, a set of connections between their variables and a set of simulation parameters and runs a simulation.

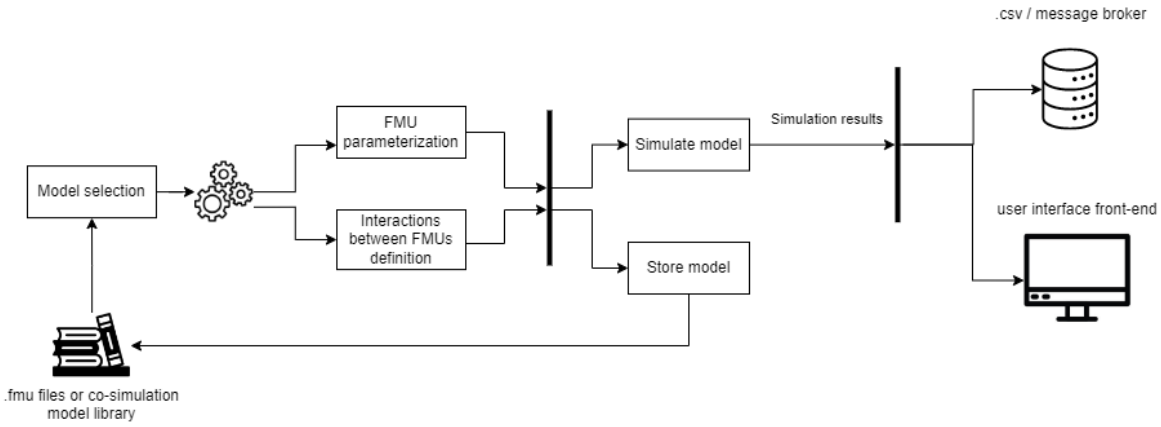

Figure 2. Physics-based i4Q^{DT} workflow diagram

The main functionalities of the physics-based workflow are described in **Figure 2**. The developments carried out until now are the following ones:

- Model class implementation:
  - Definition of the model API: there is a Python base class ModelBase which contains the basic functionality and two derived classes ModelCS and ModelME used for Co-Simulation and Model Exchange, respectively.
  - Read a model from an FMU file: the model information (general information, variables and their attributes, etc.) are read from an xml description file contained on the FMU.

    - The variables of the model and their attributes are stored in a dictionary data structure.
    - Properties implemented for easy access of variables based on causality: inputs, outputs, locals, independents, parameters and calculated parameters. This classification of variables is used extensively in master algorithm implementations.
    - Function to return variables with options for filtering, grouping and selecting variable attributes. The filtering is performed based on an argument function which must contain a Boolean condition for one or multiple attributes. The grouping is performed by the value of the attribute supplied as argument. The attributes of the variables to be returned by the function can be selected according to an argument.
    - Write the model back to an FMU file. The decompressed FMU file is compressed again to an output file using zip compression.
    - Handle FMU files compiled in a platform different from the target platform.
    - Decompression and recompilation of the c code inside the FMU.
    - Integrate gcc commands within python functions.
- Master class implementation:
    - Definition of the Master API. The Master class must be able to orchestrate multiple interconnected models and perform co-simulations of the whole system.
    - Delegation of model related work to the ModelCS class. Model information is retrieved dynamically from the model objects stored in the master and the work is performed by calling functions defined for the model. This simplifies the implementation of the Master class, allows to clearly distinguish the responsibilities of each class, and improves maintainability.
    - Implementation of a graph to characterize the connections between variables. A graph data structure allows to define arbitrary connections between the outputs and inputs of the models. The variables involved are the nodes of the graph and the directed edges connect outputs (source nodes) to inputs (target nodes).
    - Options to add/delete connections and models after the master is created. The connections are added/deleted by modifying the nodes and edges of the graph. The models are stored in a list, so they are easily added and deleted.
    - Simulation using the Jacobi method: the evaluation of the models is performed in parallel.
    - Serialization of the co-simulation to and from file: cosim files. These files can be used to save and load a co-simulation previously defined. It must define the list of models, the connections, the external inputs and the options of the simulation.
    - Return the results in different formats. The results of the simulation can be retrieved as a dictionary of arrays, a structured array or a multidimensional array.
    - Plot functions: connections of variables, connections of models.

- The state of the simulation (current iteration, simulation time, CPU time, etc.) is stored as a dictionary and can be retrieved at any moment during the simulation.
- Regeneration of the input FMUs when the origin and the docker base platforms do not match: these FMUs must be recompiled within the docker container for further use (gcc).

- Physics based back-end Flask-based Server API development:
  - Loading FMUs to the server and reloading the Master:
    - POST method to load a FMU to the server. The user can load a FMU from his/her local machine, and it is saved in an internal folder that he/she can access later for use.
    - PUT method to reload the master. Re-instantiates the Master class to get rid of all the previously loaded FMUs.
  - FMU selection:
    - GET method to show all available FMUs saved in data folder. The data folder is where all FMUs are stored within the application.
    - POST method to load FMUs to Master. Reads the input from the user and adds the selected FMUs to the Master.
  - Simulation - simulation implementation on the server:
    - POST method to receive simulation parameters (start values, model's connections and output variables):
    - Start values: reads user input on the initial values of the desired parameters and defines them on the Master.
    - Model's connections: implementation of connections between the FMUs specified by the user.
    - Output variables: reads the user input on the desired output variables.
  - Integration with i4Q message broker:
    - i4Q^{DT} back-end must be adapted to receive configuration input and simulation output to a Kafka broker
- Physics based test suite development:
  - Heat system model:
    - Create OpenModelica submodels and generate fmus: heatPlant and temperatureController.
    - Validate the results: OpenModelica simulation vs fmiSim simulation; whole model simulation vs split model simulation.
    - Test different external setpoints and check that the temperature controller is capable of following the input signal.
    - Test changes in parameters of the temperature controller.
  - Production plant model:
    - Generate FMUs from the subsystems contained in the plant: generation, server, transport delay, batch creator, batch splitter.
    - Simulate the whole model and the split models using the library.
    - Simulate the plant model from a cosim file.

5.1.2 Data-driven back-end

The data-driven approach designed and developed within the implementation phase of i4Q[DT] correspond to machine learning (ML) methods, comprising data-driven machine learning techniques, which are highly promising since a model learns critical insights directly and automatically from the given datasets.

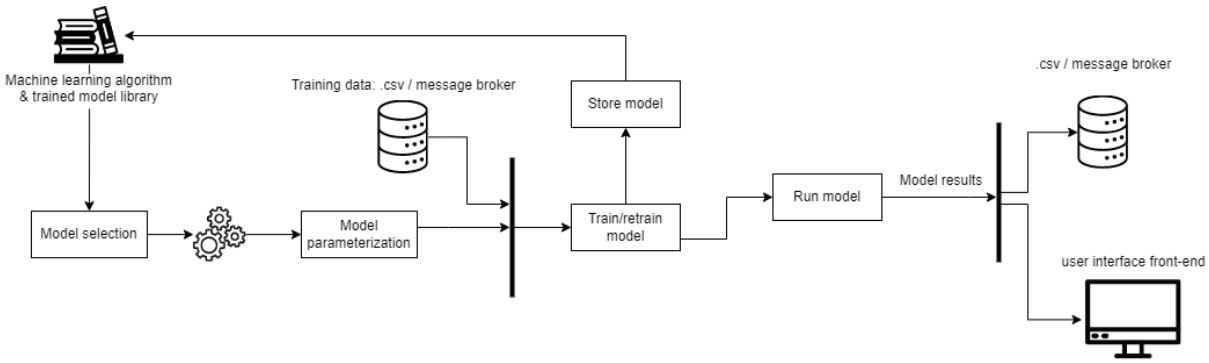

Figure 3. Data-driven i4Q[DT] workflow diagram

The main functionalities of the data-driven workflow are described in **Figure 3**. The developments carried out until now are the following ones:

- Data-Driven Backend Flask-based Server API development
  - ML Model selection
    - GET method to send available ML models to front-end
  - Data loading
    - PUT method to load a dataset in CSV format.
    - GET method to list all the available CSV files in Data folder. The data folder is where all csvs are stored within the application.
  - ML Models
    - Define ML algorithms based on the problem type they're able to solve
      - Neural network
      - K-nearest neighbours
      - Decision Tree
      - Support Vector Machine
      - Extreme Gradient Boosting
    - Define a dictionary with the parameters of each model, with the maximum and minimum values, as well as other hyperparameters to be modified by the user.
  - Model selection and parameter definition
    - GET method to list all the variables in the dataset and select the target and dependent variables.
      - Define more general training parameters:
        - Cross-validation
        - Metrics
    - PUT method to receive the user input of the selected parameters.
  - Training Resource

- POST method to perform the training of the selected algorithm.
- Define a Report to show to the user with the model's parameters and its training information

- Model saving
  - Define methods and procedures to save the trained models and its report.
- New predictions
  - GET method to show all available models saved in model's folder.
  - POST method to make new predictions. This method will read the file (the new data to make predictions on) and the selected model to make predictions with. The results will be sent back to the front-end.

5.1.3 User interface front-end

The front-end developed for i4Q^{DT} allows the user to select between the physics-based and the data-driven approach. Each of these options has a user-friendly interface that enables an easy use of the functionalities of both back-ends. The front-end has been developed using React.

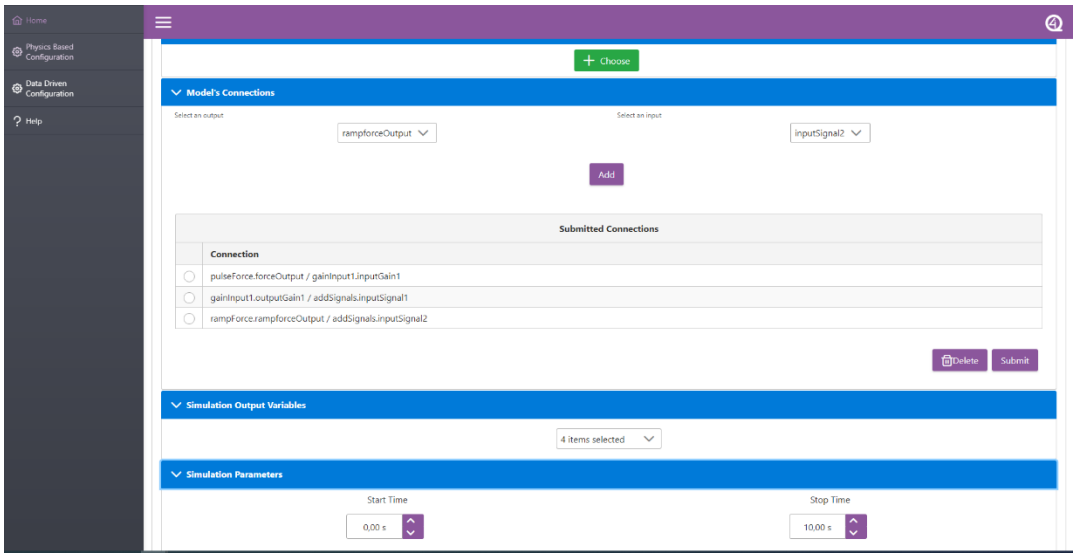

Figure 4. Physics-based i4Q^{DT} front-end screenshot

One of the main interfaces of the front-end is shown in **Figure 4**. The developments carried out until now are the following ones:

- Implementation of the physics-based workflow interface:
  - View Master TAB, where the user can see the defined Master and parametrize the start value:
    - Send FMU files to the API - The user can select a FMU file and send it to the backend. This FMU is added to the list of selectable FMU.
    - Select desired FMUs to be added to the Master. The user can select FMUs from a list.
    - Master reset order, API call - Send a reset signal to the backend in order to reset the master.

- Models Tree Table with the Master Data - Master`s retrieved data is shown as a Tree Table of fmus. The start values can be modified from this component.

- Configuration TAB, where the user can modify the simulation parameters:
  - Send CSV file with an external input signal to the API - The user can select a .csv file and send it to the backend.
  - Models' interconnection, easy way - Text or dropdown lists with inputs and outputs, respectively.
  - Selection of the output variables to be plotted - A list of all the variables is shown. The user can select the desired variables to be plotted.
  - Simulation configuration (start time, stop time, and step size) - The user can define the simulation parameters.
- Results TAB, where the user can see and export the simulation results:
  - Plot the outputs of the simulation - The selected outputs are plotted as a scatter/line chart.
  - Allow the user to download the data, as .csv, if desired.

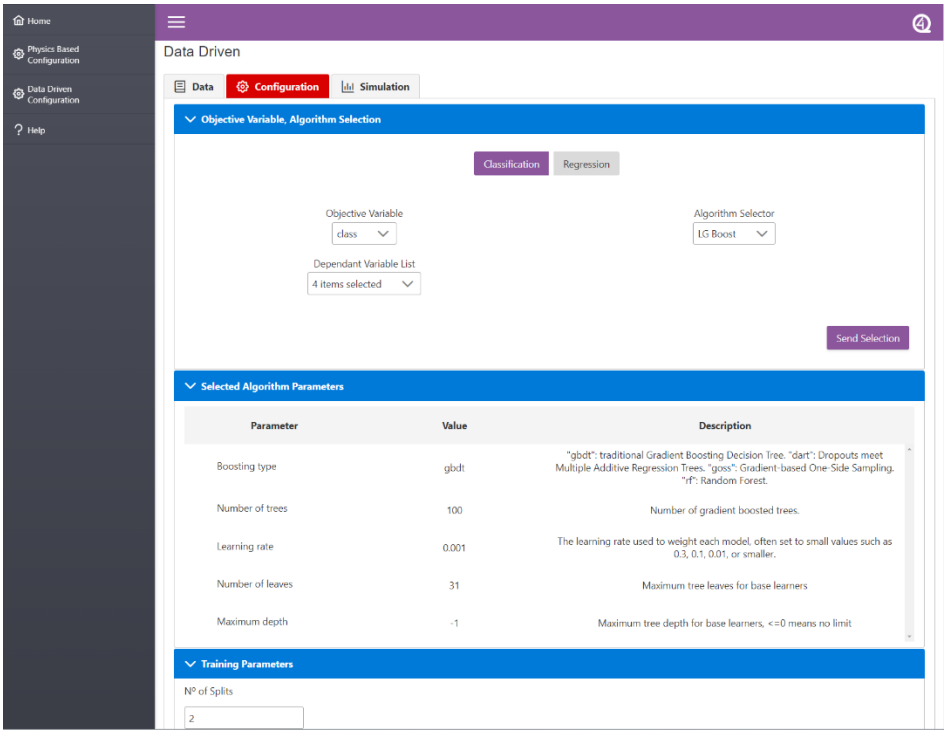

Figure 5. Data-Driven i4Q[DT] front-end screenshot

- Implementation of the data driven workflow interface:
  - Upload Data - the user can select a .csv file with the data.
  - Dataset visualization - data is displayed as a table.
  - Target Variable Selection - the user can select the desired target variable.
  - AI algorithms selector - The user can select the desired algorithms from a list. The list of selected algorithms is sent to the backend to get their parameters.

- Algorithm parametrization - The user can parametrize each parameter of every selected algorithm. Once configured, the data is sent to the backend.
- Start Training Process Order - the user will start the training process if a button is clicked.
- Show training results.
- Save Button to send the save command, train model is saved in the backend folder with the given name.
- Simulation:
  - Model Selection:
    - List with available models.
    - Model characteristics.
    - Upload a model from local files.
    - Upload button to send the selected model to the backend.
  - Results:
    - Show results in table format, with file exports.

- Implementation of 3D visualization of the simulation outputs.
  - Represent in a graphical way the movement of certain assets defined by STL files.

- Dockerization of front-end

5.1.4 Solution features analysed and mapping with user requirements

A set of features has already been developed for i4Q^{DT}, based on the set of user requirements referring to i4Q^{DT} (Deliverable D1.9) and in line with the functional viewpoints (Deliverable 2.6). Similar requirements have been assigned into common categories of tasks based on an extensive technical study conducted on user requirements, available datasets, etc., introduced to ensure the generalization abilities of the i4Q^{DT} solution.

- PC2r9 “Gather the machine condition during the data recording (test cycles)” is covered by the feature of being able to establish the relationships between the inputs of the model and the collected data (contextualization).
- PC3r1.1.2 “Prediction Conformity Area” is covered by the feature of being able to run simulations of machine learning models. These models can be uploaded and simulated to predict the product conformity in its performance.
- PC3r1.2 “Virtual tester” is covered by the feature of being able to build models. These models should represent the products resulting from the manufacturing process.
- PC4r3.4.2 “Improve the process” is covered by the feature of being able to run simulations of different models. Any process can be improved running the appropriate model that represents its physical behavior or the appropriate algorithm that predicts some feature of the outcome.
- PC4r8 “Create a Map of the plant: Instance sections, machines (types, properties, sensors, ranges)” is covered by the feature of being able to load, store and update individual models representing the different sections and machines of a plant. Each

of the models covers a specific section of the digital layout and can be edited over time as the company evolves.

5.2 History

This section provides the version history of the i4Q^{DT} solution implementation up to M24, which is summarized in the table below. The version history reports about software version including all the developments related to software implementation for all the system involved.

Version	Release date	New features
V0.0.1	16/12/2021	Tools for FMU reading, parameterization and simulation in Python
V0.0.2	22/01/2022	Development of fmiSim Python library
V0.0.3	14/02/2022	Add simple two-model co-simulation example
V0.0.4	07/03/2022	Develop physics based back-end model API
V0.0.5	14/03/2022	Develop physics based back-end master API
V0.0.6	28/03/2022	General solution front-end
V0.0.7	11/04/2022	Physics based solution front-end
V0.0.8	28/04/2022	Dockerization of front-end
V0.0.9	04/05/2022	Dockerization of back-end
V1.0.0	23/05/2022	Implementation of the compiling of FMU files for different operating systems in fmiSim library
V1.0.1	20/06/2022	Serialization of the co-simulation to and from file: cosim files
V1.0.2	25/07/2022	Physics based test suite development
V1.0.3	12/09/2022	Development of the functions of the pipeline for the data-driven workflow
V1.0.4	26/09/2022	Data-Driven Backend Flask-based Server API development
V1.0.5	10/10/2022	Implementation of different models to data driven back-end
V1.0.6	24/10/2022	Data driven solution front-end
V2.0.0	28/11/2022	3D visualization of results through STL files

Table 1. i4Q^{DT} Version history

6. Conclusions

Deliverable "D4.16 – i4Q Digital Twin v2" is a technical specification document, providing an in-depth technical overview of the i4Q^{DT} solution. It describes in detail the role, the functionalities, and the conceptual architecture of i4Q^{DT}. It presents a description of the main features of the solution to clarify the key functionalities and objectives of the i4Q^{DT} solution, describing its architecture diagram with respect to i4Q Reference Architecture. The current implementation status of i4Q^{DT} is detailed thoroughly, presenting the significant progress of this overall development. The solution is in a stable state, and there are not major additional developments foreseen for the future.

7. References

[1] *Functional Mock-up Interface*. Available at: https://fmi-standard.org/

8. Appendix I

Provides the PDF version of the i4Q **< Digital Twins >** web documentation, which can be accessed online at: **http://i4q.upv.es/16_i4Q_DT/index.html**