

Project Acronym: STAR
Grant Agreement number: 956573 (H2020-ICT-2020-1 – Research and Innovation Action)
Project Full Title: Safe and Trusted Human Centric Artificial Intelligence in Future Manufacturing Lines
Project Coordinator: INTRASOFT International



Funded by the Horizon 2020
Framework Programme of the
European Union

DELIVERABLE

D5.1 – Digital Models for Human Centric AI-based Production Processes – Initial version

Dissemination level	PU -Public
Type of Document	Report
Contractual date of delivery	31/12/2021
Deliverable Leader	SUPSI
Status - version, date	Final – V1.0, 22/12/2021
WP / Task responsible	WP5
Keywords:	Human digital models, human digital twin, Digital twin

This document is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956573. It is the property of the STAR consortium and shall not be distributed or reproduced without the formal approval of the STAR Management Committee. The content of this report reflects only the authors' view. The European Commission is not responsible for any use that may be made of the information it contains.

Executive Summary

This document provides an overview of the activities and the results achieved within M3-M12 of Task 5.1 - Human-Centred Digital Models for AI Systems and Processes. The goal of T5.1 is to develop a reference model that enables the storage, exchange, and use of technical information that involves humans in AI-based production systems.

The main activities carried out during T5.1 early stage are: 1) the definition of the reference model; 2) the development of the Human Digital Twin (HDT) Core Infrastructure; 3) the development of a functional module to recognize the worker intention (Worker's Intention Recognition Module).

In order to define the reference model, a detailed analysis of the recent advances and applications of HDT in manufacturing has been performed, so as to identify the existing use cases where HDTs are commonly applied. The conducted analysis combined with the requirements gathered in the WP5 modules has been instrumental to the design of the HDT reference model. The resulting model supports the description of an HDT, including human-centred elements, but also contextual elements relevant to the characterization of workers and their environment in a production system. The model consists of 7 main class packages: Common Descriptors Models, Production System Models, Worker Models, Characteristic Models, Measurement Models, State Models, Intervention Models. The reference model is at the basis of the HDT core infrastructure and in particular, is the core element of the HDT Orchestrator.

A first prototype of the HDT core infrastructure has already been implemented and is currently available only for STAR's partners at [this link](#). This implementation is based on a modular architecture specifically designed to use the HDT in different applications, relying on and integrating different types of modules. The HDT Core Infrastructure is divided into three main technological layers (Sensor Layer, Middleware and Digital Twin Architecture) and consists of different components: Agents and Gateways that enable data collection from workers and operational units; IIoT Middleware that supports M2M connectivity and data flows; Data Storage and Time Series Data Storage that support the storage of data flowing to the IIoT Middleware as time series or (quasi-)static data; Orchestrator that is responsible for the management of all the entities in the HDT; Administration Shell that enables the management and configuration of the HDT; GUI which recaps and offers a nice and understandable way to visualize the data flowing in the HDT; Functional Modules (Digital Twin Core - Data processing, analysis and decision modules) aimed at recognizing human states and conditions and computing complex features that enable human and machine decision makers to take human factors into account in their execution and control logics.

Finally, this document reports on the activities related to the Workers' Intention Recognition Model focused on the design and implementation of the capacitive sensors with IMU on both wrists, as well as the collection of video and sensor data through cameras and the developed worker activity recognition sensor. The sensor prototype combines three boards: an Adafruit Feather nRF52840 backend motherboard, a custom human body capacitive sensor board, and a data logger board. The initial activity detection experiments involved twelve volunteers. The preliminary analysis of these signals shows that the types of motion are distinguishable.

Improvements to the work presented in this deliverable are expected in the next period and they will be presented in D5.2 (M24). The improvements include: additional Agents to support more wearable devices (e.g., Garmin), and other kinds of devices (e.g., cobots); the integration of data gateways to support OSs other than Android (e.g., Raspberry Pi OS); the full support for IIoT middleware based on protocols other than MQTT (e.g., FIWARE Orion Context Broker); the release of the Orchestrator Administration Shell as a Web-based application to make the interaction with the Orchestrator API more user-friendly.

About the core data model, it is quite stable and no further extensions are planned, unless new requirements have to be met.

Deliverable Leaders:	SUPSI
Contributors:	DFKI
Reviewers:	JSI, UPRC
Approved by:	Charalampos Ipektsidis (INTRA)

Document History			
Version	Date	Contributor(s)	Description
0.1	01/08/2021	SUPSI	Table of contents
0.2	01/10/2021	SUPSI	First draft (section 1-3)
0.3	09/11/2021	SUPSI	Second draft (section 1-5)
0.4	25/11/2021	DFKI	Third draft (section 6)
0.5	06/12/2021	SUPSI	Consolidated draft
0.6	20/12/2021	JSI, UPRC	Reviewed draft
0.7	21/12/2021	SUPSI	Final version
1.0	22/12/2021	INTRA	QA and creation of the final to be submitted

Table of Contents

EXECUTIVE SUMMARY	2
TABLE OF CONTENTS.....	5
TABLE OF FIGURES.....	7
LIST OF TABLES.....	8
DEFINITIONS, ACRONYMS AND ABBREVIATIONS	9
1 INTRODUCTION.....	10
2 HUMAN DIGITAL TWIN IN MANUFACTURING	11
3 THE HUMAN DIGITAL TWIN REFERENCE MODEL.....	13
3.1 COMMON DESCRIPTORS MODELS	15
3.1.1 <i>AbstractDescriptor</i>	15
3.1.2 <i>UnitOfMeasure</i>	16
3.1.3 <i>Category</i>	16
3.1.4 <i>Taxonomy and TaxonomyItem</i>	17
3.1.5 <i>Scale</i>	17
3.2 PRODUCTION SYSTEM MODELS	18
3.2.1 <i>FactoryEntityModel</i>	18
3.2.2 <i>FactoryEntityModelCategory</i>	19
3.2.3 <i>DeviceModel</i>	20
3.2.4 <i>OutputMap</i>	20
3.2.5 <i>FactoryEntity</i>	21
3.2.6 <i>Factory</i>	22
3.2.7 <i>Session</i>	22
3.3 WORKER MODELS	22
3.3.1 <i>Worker</i>	22
3.3.2 <i>EventDescriptor and InteractionDescriptor</i>	23
3.3.3 <i>Event and Interaction</i>	23
3.4 CHARACTERISTIC MODELS	24
3.4.1 <i>CharacteristicDescriptor</i>	24
3.4.2 <i>CharacteristicValue</i>	24
3.5 MEASUREMENT MODELS.....	25
3.5.1 <i>MeasurementDescriptor</i>	25
3.6 STATE MODELS	25
3.6.1 <i>StateDescriptor</i>	26
3.6.2 <i>FunctionalModule</i>	26
3.6.3 <i>FunctionalModuleInput</i>	27
3.6.4 <i>FunctionalModuleOutput</i>	27
3.6.5 <i>Block</i>	28
3.6.6 <i>ListBased</i>	28
3.6.7 <i>StructBased</i>	28
3.6.8 <i>StringBased</i>	29
3.6.9 <i>NumberBased</i>	29
3.6.10 <i>BooleanBased</i>	29
3.7 INTERVENTIONS MODELS	29
3.7.1 <i>InterventionDescriptor and Intervention</i>	29
4 THE ARCHITECTURE OF THE HUMAN DIGITAL TWIN	31
4.1 THE TECHNOLOGICAL FRAMEWORK FOR HUMAN DIGITAL TWINS	31

4.2	THE CORE ARCHITECTURE	33
4.2.1	<i>Agents and Gateway</i>	36
4.2.2	<i>IIoT Middleware</i>	36
4.2.3	<i>Orchestrator, Administration Shell and GUIs</i>	36
4.2.4	<i>Video image streamer</i>	37
4.2.5	<i>Persistency Layer</i>	37
4.2.6	<i>History Keeper Module</i>	38
4.3	THE FUNCTIONAL MODULES	38
4.3.1	<i>Fatigue Monitoring System</i>	38
4.3.2	<i>Integration with the HDT core</i>	38
4.3.3	<i>Worker's Intention Recognition Module</i>	39
4.3.4	<i>Integration with the HDT core</i>	39
4.3.5	<i>Safety Zones Detection System and AGV Fleet Optimizer</i>	40
4.3.6	<i>Integration with the HDT core</i>	40
5	IMPLEMENTATION OF THE CORE ARCHITECTURE	42
5.1	THE IIoT MIDDLEWARE.....	42
5.1.1	<i>Analysis and benchmark of existing solutions</i>	43
5.1.2	<i>Implementation details</i>	46
5.2	THE ORCHESTRATOR	46
5.2.1	<i>Implementation details</i>	46
5.3	THE HISTORICAL DATA MANAGER	47
5.3.1	<i>Implementation details</i>	47
5.4	THE GATEWAY AND AGENTS.....	48
5.4.1	<i>Implementation details</i>	49
6	WORKER'S INTENTION RECOGNITION MODULE.....	52
6.1	HARDWARE DESIGN.....	52
6.2	USE CASE AND EXPERIMENT DESIGN	54
7	CONCLUSIONS.....	59
	REFERENCES	60

Table of Figures

FIGURE 1 HUMAN DIGITAL TWIN REFERENCE MODEL14

FIGURE 2 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON COMMON MODELS.....15

FIGURE 3 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON PRODUCTION SYSTEM MODELS18

FIGURE 4 DIFFERENT DEVICES, COLLECTING SAME PHYSIOLOGICAL, FEEDING THE SAME MEASUREMENT21

FIGURE 5 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON WORKER MODELS22

FIGURE 6 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON CHARACTERISTIC MODELS24

FIGURE 7 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON MEASUREMENT MODELS.....25

FIGURE 8 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON STATE MODELS26

FIGURE 9 HUMAN DIGITAL TWIN REFERENCE MODEL: HIGHLIGHT ON INTERVENTION MODELS.....29

FIGURE 10 STAR'S ARCHITECTURE31

FIGURE 11 ABSTRACT DIGITAL TWIN ARCHITECTURE31

FIGURE 12 WP5 ARCHITECTURE PROPOSAL V135

FIGURE 13 INTEGRATION OVERVIEW: HDT AND FATIGUE MONITORING SYSTEM.....39

FIGURE 14 INTEGRATION OVERVIEW: HDT, SAFETY ZONES DETECTION SYSTEM AND AGV FLEET OPTIMIZER40

FIGURE 15 INTEGRATION OVERVIEW: HDT AND WORKERS' TRAINING PLATFORM41

FIGURE 16. THE CLASS DIAGRAM OF THE JAVA APPLICATION USED IN THE EXPERIMENT (OTHER PACKAGES THAN "MOSQUITTO" HAVE BEEN OMITTED FOR THE SAKE OF DIAGRAM CLARITY).....44

FIGURE 17. WORKERID INPUT VIEW49

FIGURE 18. AVAILABLE CONNECTIONS49

FIGURE 19 TOP (TL), SIDE (TR), LEFT (BL) AND WORN ON THE WRIST (BR) VIEWS OF THE COMBINED SENSING PROTOTYPE.....53

FIGURE 20 EXAMPLE OF RECORDED DATA.....53

FIGURE 21 DFKI'S SMARTFACTORYKL TEST LAB AND STRUCTURE OF USE CASE.....54

FIGURE 22 FLOW CHART OF THE ACTIVITIES IN USE CASE AT DFKI'S SMARTFACTORYKL TEST LAB55

FIGURE 23 EXAMPLES OF LABELLED SENSOR DATA: MOVEMENTS 0-457

FIGURE 24 EXAMPLES OF LABELLED SENSOR DATA: MOVEMENTS 5-958

List of Tables

TABLE 1. TEST RESULTS WITH 10 PUBLISHERS AND 10 SUBSCRIBERS	45
TABLE 2 TEST RESULTS WITH 100 PUBLISHERS AND 100 SUBSCRIBERS.....	45

Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
WP	Work Package
HDT	Human Digital Twin
BL	Bottom left of the figure
BR	Bottom right of the figure
TR	Top right of the figure
TL	Top left of the figure
CPS	Cyber-Physical Systems

1 Introduction

Industry 4.0 introduced the idea that production system control and decision-making can be realised, even automatically, by relying on Cyber-Physical Systems (CPS) as dual elements composed of a physical production item and its digital counterpart: the digital twin (DT) [REF-01].

Many examples exist where digital copies of machines, devices, products, or entire production systems are used to improve performances, make predictions and take decisions. However, humans, which still have a relevant impact on process quality, performances and continuous improvement [REF-02], have been excluded, up to now, from the digital representation of the factory. The human factor is usually considered only within industrial and workplace design to improve ergonomics, prevent hazards, and to train and educate, rather than for continuous decision-making and production system control. Yet, to create production systems that seamlessly complement human capabilities, the digital factory has to include a precise and realistic digital representation of humans: the Human Digital Twin (HDT).

To this end, Task 5.1 targets STAR's **Objective 4 - Human-centred simulations and digital twins for safe AI systems in manufacturing**, aiming to facilitate the human-machine collaboration, considering, understanding and anticipating the humans in the production system. The task develops an infrastructure, together with its models, to support the creation of human-centred digital twins that comprise digital models of the human workers, including their behaviour and their interactions. The infrastructure is the integration point of the modules developed within WP5, namely Fatigue Monitoring System, Worker's Intention Recognition Module, Safety Zones Detection System, AGV Fleet Optimizer and Workers' Training Platform.

This document sum-ups the activities carried out in the first 7 months and is structured in 7 sections. Section 1 provides an overview of the scope and the structure of this document. Section 2 includes a review of the literature related to HDT in manufacturing and industrial contexts. Section 3 introduces the models on which the STAR's HDT relies on. Section 4 details the HDT's architecture and Section 5 the implementation activities carried out to realise the first STAR's HDT prototype. Section 6 describes the activities carried out to develop the Worker's Intention Recognition Module, which is one of WP5's modules dedicated to predicting workers' behaviours in the production system. Finally, Section 7 provides an overview of the next steps planned in the second part of Task 5.1, which will start from M13.

2 Human digital twin in manufacturing

NASA introduced the concept of DT in 2012 as “an integrated multi-physics, multi-scale, probabilistic simulation of a flying vehicle or systems” [REF-03]. From that moment, this concept has evolved and has been adopted in several domains. Thanks to the technologies introduced by the Industry 4.0 paradigm, DTs became incredibly relevant in the manufacturing industry. DTs have been successfully used to mirror and simulate industrial settings, for predictive maintenance, for virtual commissioning, for anomaly detection, and for optimisation of the product life cycle.

Despite several examples being available in the context of products, devices, and machines, only a handful of works addressing the HDT (or more in general human modelling in industrial contexts) can be found. Workers have been considered, in the overall factory digital representation, mainly from a high-level perspective by creating mere static digital models [REF-04]. However, such models are not enough to dynamically mirror the worker as the DT is more than a representation having to estimate the status and to simulate the behaviour of the things it represents.

According to Segan and colleagues, the HDT can include models fed by dynamic and real-time data merged with static or quasi-static ones, enabling a comprehensive representation of the human entity [REF-05]. To gather real-time information, activities and behaviours performed by humans have to be recorded. Moreover, these real-time data have to be compared with historical data, which are stored and formalised together with information describing human characteristics and conditions [REF-06]. The HDT has not only to provide a digital representation of anthropometric and physiological features but also a representation of a person’s inner state [REF-07].

In [REF-08] a meta-model is defined to realise a modular and tailored HDT comprehensive of all the entities that need to be modelled to create a HDT. These include worker’s characteristics, medical, emotional and psychophysical conditions, psychophysical and geospatial parameters, contextual, functional and decision models. The HDT is a necessary technology to facilitate human worker integration in an Industry 4.0 environment to address communication, data aggregation, simulation and scheduling [REF-09]. The emerging applications of HDT realized in the manufacturing industry in the last 5 years mainly include workers monitoring, production planning and scheduling, human-robot collaboration and adaptive automation.

Worker well-being monitoring

Thanks to the miniaturization and reduction of costs, the adoption of wearables and sensors, has been growing also in the industrial context to investigate workers’ conditions and well-being. Employee’s well-being is a key factor in determining the organization’s long-term competitiveness and it is also directly related to production efficiency. The cumulative effect of positive impacts on the human factor brings economic benefit through productivity increase, scrap reduction and decrease of absenteeism. Few research works have been recently developed, where workers’ physiological data are used to infer the insurgence of phenomena such as fatigue [REF-10][REF-11] and mental stress [REF-12] which, in turn, have a relevant impact on process performance. Another research line adopted eye-trackers, together with wearables and cameras to estimate worker’s attention and stress levels, to

understand assembly sequence and to identify the criticalities in the product design affecting the assembly process[REF-13].

Production planning and allocation

In labour-intensive production systems, workers allocation is one of the key activities as capacity and worker skills are the main factors that affect the production rate [REF-14]. The exclusion of such elements from the problem definition may result in poor performance. To this end, the HDT has been applied to support production planning and allocation. [REF-15][REF-16] propose a HDT to store, organize and communicate workers' skills, preferences, virtualized personality and to enable humans to take part to a decentralized computational decision-making process which leads to an improved task scheduling. However, in this context and application, it is fundamental to consider that, despite the different taxonomies and methods that exist to assess workers' skills like ESCO and O*NET, humans learn and grow, being able to deal with novel challenges. This is a relevant challenge to consider in realising a HDT supporting production planning and scheduling.

Human-robot collaboration and Adaptive automation

In the last decades, adaptive controls systems have been explored, including workers features within real-time control loops[REF-17][REF-18][REF-19]. The HDT is crucial for such kind of approaches. [REF-20] used a Microsoft Kinect sensor to identify and track a worker within a work cell to identify the possible collisions areas with a robot. This information has been used to optimize in real-time robot trajectories in order to reduce collisions. In [REF-21], a HDT, has been adopted to monitor worker fatigue and mental stress by introducing a physiological monitoring system and a smart decision-maker to adjust the level of support offered through a collaborative robot (cobot).

Ergonomics analysis and layout design

Many examples exist where the HDT has been used to analyse the work cell ergonomics and to define the best layout considering worker characteristics, anthropometric ones above all. Many examples exist applying off-line simulation of humans [REF-22]. Moreover, a few real-time examples can be found, where worker posture and characteristics are computed using motion capture tools [REF-23] and even used to feed control systems to perform collaborative and ergonomic tasks [REF-24].

Other context applications outside the manufacturing industry

Extending the scope of the use of HDT outside the manufacturing industry, it is possible to find interesting applications. A digital representation of construction workers has been created collecting automatically physiological parameters (e.g., human heart rate, upper body posture angle, traveling speed) to identify workload severity [REF-25]. In the same sector, workers' thoracic posture and spatio-temporal data have been used to estimate activity types and assess productivity in real-time[REF-26]. In the medical and fitness fields, some applications implement the automatic exchange of data relying on wearable devices to compute health conditions [REF-26][REF-27] and predict athletes' performance [REF-28].

3 The Human Digital Twin Reference Model

This chapter introduces the reference model that has been the basis for the STAR HDT implementation. The model conceptualization derives from a meta-model for the HDT design available in literature [REF-08].

The main goal of the HDT reference model is to provide a common model to describe an HDT, including human-centred elements, but also contextual elements, relevant to characterise the workers and the surrounding environment in a production system. The model and its implementation aim at releasing an **extensible**, **scalable** and **adaptable** HDT. The model has been realised following the Unified Modelling Language (UML) standards and it is grounded on two principles, which are proper to Domain Driven Development (DDD) and Object-Oriented Programming (OOP), namely, inheritance/generalisation, and aggregation/composition. In this way, the advantage for the adopters of the proposed HDT model is two-fold: they can rely on a model built on the robustness of a scientific result; they are provided with ready-made packages of entities to instantiate their own HDT, by including also human-centred aspects (e.g., interactions, events - these aspects are discussed in Section 3.3), and software-based simulations/predictions (e.g., output of functional models predicting the state of factory entities - further details are given in Section 3.6). A set of primitive data and binary formats for serialisation processes (JSON or XML) finally guarantee that the model can be transferred without loss of information and transformed into digital twins by dedicated software.

In the STAR project, the HDT is applied mainly in one use case: Human-Cobot Collaboration improving robust Quality Inspections. For this initial experiment, sensors and specific human features have been selected (e.g., heart rate, accelerations, occupied space) to fulfill the expected objectives (e.g., monitoring worker fatigue). More details about the selection of this features will be provided within *D5.3 - Digital Twins for Security and Safety - Initial version*. Moreover, analysis and integration feasibility assessment are on-going to apply the HDT also in other use-cases (e.g., Human Behaviour Prediction and Safety Zones Detection for Routing and for the Worker Training Platform).

The STAR's HDT is designed to easily integrate new types of sensors and devices and to characterise and model various other features. The model described in this chapter enables the characterization and representation of workers in the digital world, as well as contextual elements of the factory, including machines, sensors, robots and the environment (e.g., work cells).

The proposed HDT UML class diagram is composed of different sets of classes:

- Common Descriptors Models (white)
- Production System Models (yellow)
- Worker Models (green)
- Characteristic Models (violet)
- Measurement Models (red)
- State Models (blue)
- Intervention Models (orange)

All the classes described in the diagram have a property *id* of type UUID identifier that is used to identify class instances within the HDT; however, this property, as well as most of all the class methods, have been omitted for the sake of diagram readability.

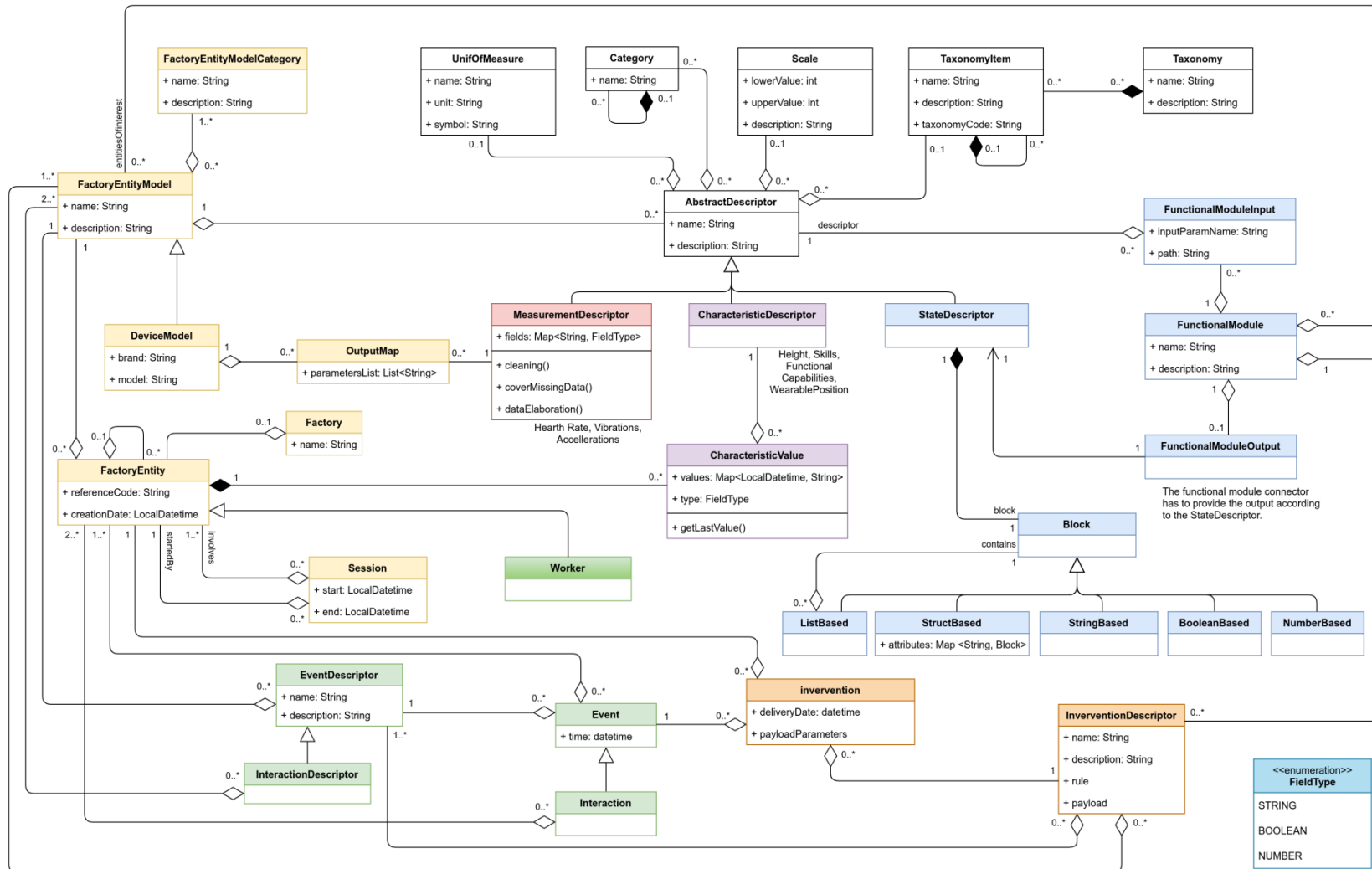


Figure 1 Human Digital Twin reference model

3.1 Common Descriptors Models

The classes belonging to the Common Descriptors allow the description of properties, characteristics, measurements, dimensions and states of the many entities operating in a factory, including workers, machines, robots and devices.

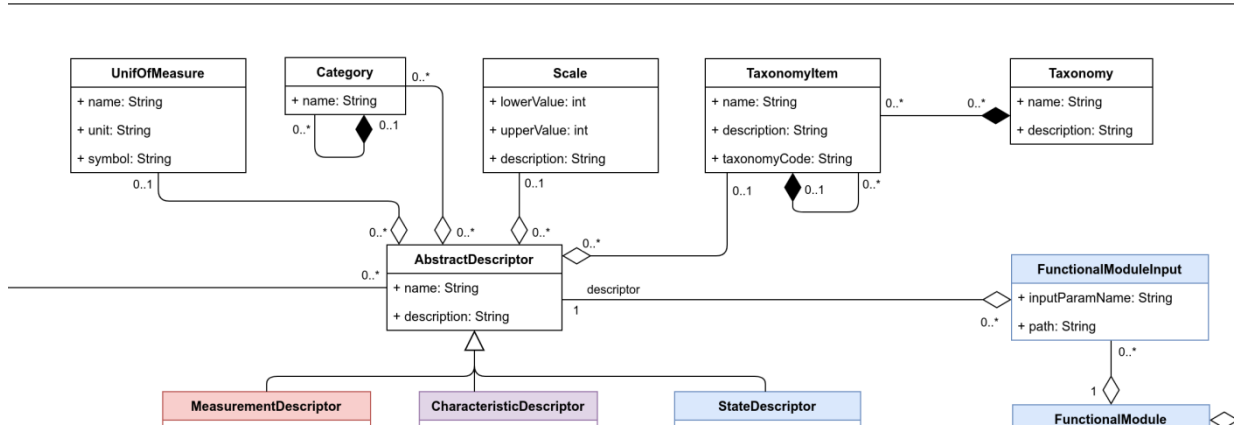


Figure 2 Human Digital Twin reference model: highlight on Common Models

3.1.1 AbstractDescriptor

The **AbstractDescriptor** allows the description of any type of data managed by the HDT: characteristics, measurements and states. The descriptor can be enriched with different information such as its UnitOfMeasure, Category, TaxonomyItem and Scale. Thanks to these auxiliary classes, the **AbstractDescriptor**(s) can be of different nature, depending on the factory thing they describe.

The **AbstractDescriptor** is described by the following attributes:

Attribute	Type	Description
name	String	The name of the element the AbstractDescriptor describes.
description	String	A Human-readable description of the AbstractDescriptor .

The **AbstractDescriptor** has the following relations:

Class	Relation type	Multiplicity	Description
UnitOfMeasure	Aggregation	0..1	An AbstractDescriptor can be related to at most one unit of measure.
Category	Aggregation	0..*	An AbstractDescriptor can be related to zero or more categories.
Scale	Aggregation	0..1	An AbstractDescriptor can be related to at most one scale.
TaxonomyItem	Aggregation	0..1	An AbstractDescriptor can be related to at most one taxonomy item.

FunctionalModuleInput	Association	0..*	An AbstractDescriptor can refer to zero or more input parameters of a functional module ¹ .
-----------------------	-------------	------	---

An **AbstractDescriptor** can be specialized into:

- **CharacteristicDescriptor**: it describes static or quasi-static data characterising entities in a factory (e.g., workers, robots). Examples of characteristics for workers are: a skill, an anthropometric characteristic, a job position. Examples of characteristics for machines are: weight, dimension.
- **StateDescriptor**: it describes the state of entities in a factory. For example, in the case of a worker, possible states are the current task, the next task to perform, the level of perceived fatigue, the current production performance.
- **MeasurementDescriptor**: it describes a measurement collected from a sensor (usually onboarded on a device), which refers to a factory entity. For example, in the case of a worker, a MeasurementDescriptor can describe the heart rate, measured by a wearable device.

3.1.2 UnitOfMeasure

The class **UnitOfMeasure** has been defined to facilitate the handling of the units of measure, which can refer to all the **AbstractDescriptor(s)** (i.e., **CharacteristicDescriptor(s)**, **StateDescriptor(s)**, and **MeasurementDescriptor(s)**). This class allows the definition of a unit of measurement.

The **UnitOfMeasure** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the unit of measure
unit	String	Unit of the unit of measure
symbol	String	Symbol representing the unit of measure

3.1.3 Category

The class **Category** describes an **AbstractDescriptor** by means of an enumeration. For example, it can be "Anthropometric Characteristic", "Ability", "Skill", etc. It facilitates the organization and structuring of **AbstractDescriptor(s)**.

The Category is described by the following attributes:

Attribute	Type	Description
name	String	The name of the category

The **Category** has the following relations:

Class	Relation type	Multiplicity	Description
Category	Composition	0..*	A Category can be composed by a set of sub-categories, that allows to

¹ The functional modules are pluggable components that can be easily added or removed from the HDT. They can be AI modules, but also simpler ones, like ROS applications that describe the state or the behaviour of a robot.

			create a hierarchical structure.
Category	Association	0..1	A Category can have at most one parent category in the hierarchical structure.

3.1.4 Taxonomy and TaxonomyItem

The **Taxonomy** class is used to classify an **AbstractDescriptor** accordingly with a taxonomy (either a well-known taxonomy, like (e.g., O*Net or ESCO for working skills, or a private one, e.g., a taxonomy that organises the roles in the company).

The **Taxonomy** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the Taxonomy .
description	String	Human readable description of the Taxomy .

The **Taxonomy** has the following relations:

Class	Relation type	Multiplicity	Description
TaxonomyItem	Composition	0..*	A Taxonomy is composed by a set of TaxonomyItem (s), representing the single items belonging to the taxonomy.

A **Taxonomy** is composed by a set of **TaxonomyItem**(s), which represent the single items that compose a taxonomy (e.g., for the skills taxonomy, **each TaxonomyItem** represents a skill; in the case of the O*Net taxonomy, "Operation Monitoring", "Quality Control Analysis", and "Reading Comprehension" are TaxonomyItems).

The **TaxonomyItem** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the TaxonomyItem .
description	String	Human-readable description of the TaxonomyItem .
taxonomyCode	String	Code that represents the item in the taxonomy.

The **TaxonomyItem** has the following relations:

Class	Relation type	Multiplicity	Description
TaxonomyItem	Composition	0..*	A TaxonomyItem can be composed by a set of sub-items, creating a hierarchical structure.
TaxonomyItem	Association	0..1	A TaxonomyItem may have one parent item in the hierarchical structure.

3.1.5 Scale

The **Scale** class is used to provide an **AbstractDescriptor** with a scale that narrows its possible values, making it easier to assign, understand, and interpret the measured value in a meaningful way.

Characteristics, states, measurements and interactions of such entity models have to be included in the HDT. It is important to remark that the **FactoryEntityModel** is a generic representation of a factory entity, not its specific instance. For example, a **FactoryEntityModel** could be “assembly operator” or “Cobot UR10”. Specific instances of these models can be described in the **Worker** class, in the case of a worker, and in the **FactoryEntity** in the case of any other type of entity.

The **FactoryEntityModel** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the FactoryEntityModel .
description	String	Human readable description of the FactoryEntityModel .

The **FactoryEntityModel** has the following relations:

Class	Relation type	Multiplicity	Description
AbstractDescriptor	Composition	0..*	A FactoryEntityModel is composed by a set of AbstractDescriptor(s) . They represent characteristics, measurements and states that have to be represented in the HDT for describing the FactoryEntityModel . For example, a FactoryEntityModel could be Cobot UR10 which could be composed by the following AbstractDescriptor(s) : <ul style="list-style-type: none"> • CharacteristicDescriptor: reach, number of joints, installation date, etc. • StateDescriptor: availability, current end-effector, current task, next task to be performed, etc. • MeasurementDescriptor: joints position, work bench vibration, etc.
FactoryEntityModel Category	Composition	1..*	A FactoryEntityModel has at least one FactoryEntityModelCategory .

3.2.2 FactoryEntityModelCategory

The **FactoryEntityModelCategory** specifies the category of a **FactoryEntityModel**. In this way, it is possible to organize **FactoryEntityModel(s)**. For example, the **FactoryEntityModel** “wearable device” may be assigned to two different categories: “wrist band” and “chest band”.

The **FactoryEntityModelCategory** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the FactoryEntityModelCategory .
description	String	Human readable description of the

		FactoryEntityModelCategory.
--	--	------------------------------------

3.2.3 DeviceModel

The **DeviceModel** is used to describe any device that generates measurements, including sensors, machines and wearables. Exactly like the **FactoryEntityModel**, the **DeviceModel** is a generic description of a device (it is not a specific instance). The need for this class arises from the fact that device models (e.g., vibration sensor SW-420, Siemens Simatic S7-1200), including wearables (e.g. Garmin Instinct) collect measurements in different ways and with different data structures. The **DeviceModel** class allows a particular device model to be described, enabling the mapping of physical device outputs to one or more **MeasurementDescriptor(s)** or **StateDescriptor(s)** in the HDT, minimizing the overhead of connecting devices of the same model.

The **DeviceModel** is described by the following attributes:

Attribute	Type	Description
brand	String	Name of the device brand.
model	String	Name of the device model.

The **DeviceModel** has the following relations:

Class	Relation type	Multiplicity	Description
OutputMap	Aggregation	0..*	This is the relation between the deviceModel and the OutputMap that allows the HDT to translate and use as input data collected from a device to feed a MeasurementDescriptor .

3.2.4 OutputMap

The **OutputMap** maps a physical parameter measured by a device with the field described by a **MeasurementDescriptor**, making the descriptor independent from the device models. A **DeviceModel** can produce different types of measurements (e.g., heart rate, blood pressure, etc.). This measurement must be created and described in the HDT using **MeasurementDescriptor(s)** so that other components can interact with it. Therefore, the stream of a device data must be mapped and connected to a **MeasurementDescriptor**. Moreover, different **DeviceModel(s)** may feed the same **MeasurementDescriptor**. For example, two wearable device models (e.g., Garmin Instinct and Polar OH1), may provide the same measurement (e.g., heart rate). However, a user wants to have a unique representation of that measurement in the HDT. Therefore, a unique **MeasurementDescriptor** is created in the HDT, and each wearable model has an **OutputMap**, mapping the different outputs of the wearables to the same **MeasurementDescriptor**.

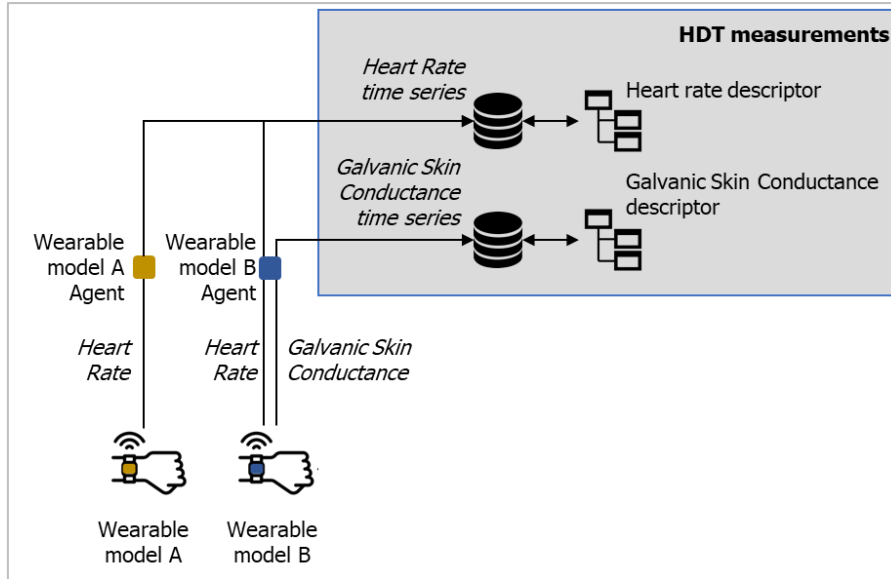


Figure 4 Different devices, collecting same physiological, feeding the same measurement

The **OutputMap** is described by the following attributes:

Attribute	Type	Description
parameterList	List<String>	An ordered sequence of device parameters. This sequence must be met when writing actual measurements to the HDT. For example, if a device produces 3 values for the accelerometer (x, y, z), possible parameterLists are [x, y, z], [z, x, y], [x, y] (if the z value is not relevant for the HDT), and more.

3.2.5 FactoryEntity

The **FactoryEntity** class allows the creation of instances of entities in a factory described through a **FactoryEntityModel**. The **FactoryEntity** is a specific entity that populates the factory.

The **FactoryEntity** is described by the following attributes:

Attribute	Type	Description
referenceCode	String	The code used to refer to the digital entity in the real world. For devices, it could be a serial number, a mac address, or any other label. For workers, it can be a registration number, a badge number, or other anonymous (or anonymized) codes.
creationDate	LocalDatetime	The date when the instance has been added to the HDT.

3.2.6 Factory

The **Factory** class allows the definition of a factory where the **FactoryEntity(s)** act and operate, being able to have HDT representing entities in different production systems.

The **Factory** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the Factory .

3.2.7 Session

The **Session** class allows data collection or working sessions to be defined; in such a way, the HDT tracks exactly when a particular **FactoryEntity** performs a specific activity, and to match **Event(s)**, **Interaction(s)**, and **Intervention(s)** with specific time slots.

The **Session** is composed by the following attributes:

Attribute	Type	Description
start	LocalDatetime	Date and time when the session starts.
end	LocalDatetime	Date and time when the session ends.

3.3 Worker Models

This section contains classes that are relevant for defining worker instances and their interactions with other entities in the factory.

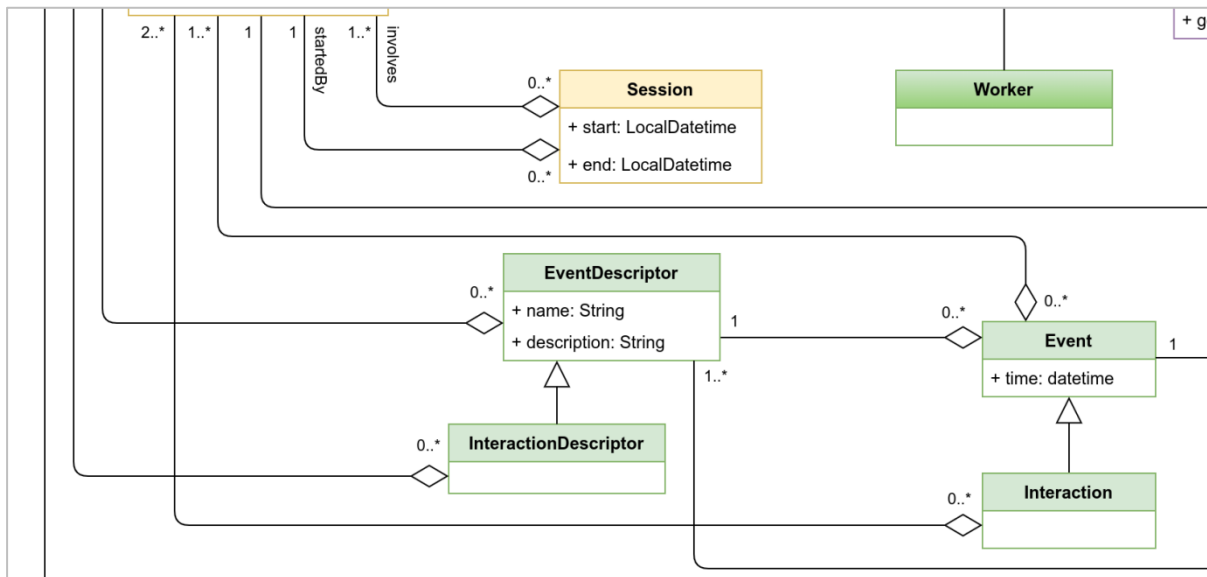


Figure 5 Human Digital Twin reference model: highlight on Worker Models

3.3.1 Worker

The class **Worker** allows the definition of worker instances. This class is used to represent a specific worker in the production system.

As of today, the class just inherits its parent's properties and relationships, but it's worth be noted that having a dedicated class enables a fine-grained management of the HDT entities by the HDT user. For example, measurements related to workers may require additional data preserving rules (e.g., when collecting biometrics data); moreover, workers can be

“anonymized” within the HDT (e.g., the referenceCode may be missing or an anonymous code), while this operation is meaningless for other factory entities in almost all the cases. Moreover, the behaviour of factory entities is in general predictable with well-established models, while the behaviour of workers may be unpredictable (due to non-measurable features, like emotions), and this point plays a crucial role in an HDT. For this reason, it is important to distinguish workers from other nonhuman factory entities.

3.3.2 EventDescriptor and InteractionDescriptor

The **EventDescriptor** class describes the events that change the HDT status, evolving any of its entities or attributes.

The **EventDescriptor** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the EventDescriptor .
description	String	Human readable description of the EventDescriptor .

The **InteractionDescriptor** class specifies the **EventDescriptor**, requiring the event being an interaction between two or more **FactoryEntityModel**(s) (e.g., a collision between a robot and a worker, a worker that loads a pallet on an AGV, etc.).

The **InteractionDescriptor** has the following relations:

Class	Relation type	Multiplicity	Description
FactoryEntityModel	Aggregation	2..*	The FactoryEntityModel (s) that are involved in the interaction..

For example, a **InteractionDescriptor** can be “Impact between cobot and worker”, which aggregate the **FactoryEntityModel**(s) “Assembly Worker” and “Cobot UR10”.

3.3.3 Event and Interaction

The **Event** class defines an event described by an **EventDescriptor**.

The **Event** is described by the following attributes:

Attribute	Type	Description
time	LocalDatetime	Date and time when the interaction event occurs.

The **Event** class has the following relations:

Class	Relation type	Multiplicity	Description
FactoryEntity	Aggregation	1..*	An Event aggregates FactoryEntity (s) and those things have been involved in the event.

As per the **EventDescriptor** class, also the **Event** class is extended by the **Interaction** class, which requires the event to involve at least two **FactoryEntity**(s). Indeed, the Interaction class has the following relations:

Class	Relation type	Multiplicity	Description
FactoryEntity	Aggregation	..*	An aggregates FactoryEntity (s) and those things have been involved in the event.

3.4 Characteristic Models

This set of classes allows to define quasi-static and static data describing workers and factory things.

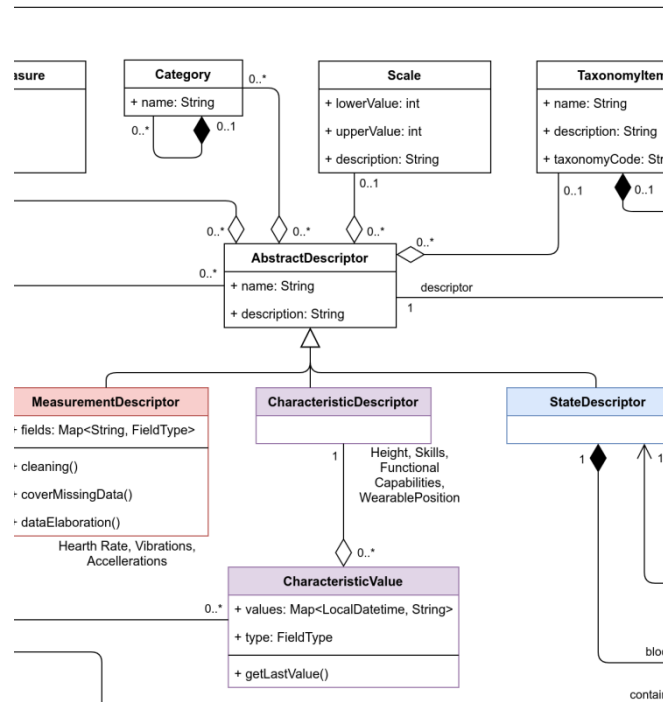


Figure 6 Human Digital Twin reference model: highlight on Characteristic Models

3.4.1 CharacteristicDescriptor

The **CharacteristicDescriptor** allows the extension of **AbstractDescriptor** with a specific class dedicated to (quasi-)static data. The class can include any kind of characteristics relevant for the description of workers and other entities in the factory. For example, the height or the set of skills may be workers’ features modelled through the class **CharacteristicDescriptor**.

3.4.2 CharacteristicValue

The class **CharacteristicValue** allows the definition of the value that characterise a **CharacteristicDescriptor** of a specific factory entity, represented by the class **FactoryEntity**.

The **CharacteristicValue** is described by the following attributes:

Attribute	Type	Description
values	Map<LocalDatetime, String>	A map with all the values the characteristic assumed over time, indexed by the acquisition timestamp. For example, the “weight” characteristic has 3 values if the worker has been weighted 3 times.
type	TypeField	The actual type of the values listed in the values Map. It is useful to

		correctly parse the content of the string value.
--	--	--

The **CharacteristicValue** has the following relations:

Class	Relation type	Multiplicity	Description
CharacteristicDescriptor	Composition	1	The CharacteristicValue is always composed by a CharacteristicDescriptor , to which the value refers to.

3.5 Measurement Models

This set of classes allows describing measurements and data collected from workers and things in the factory.

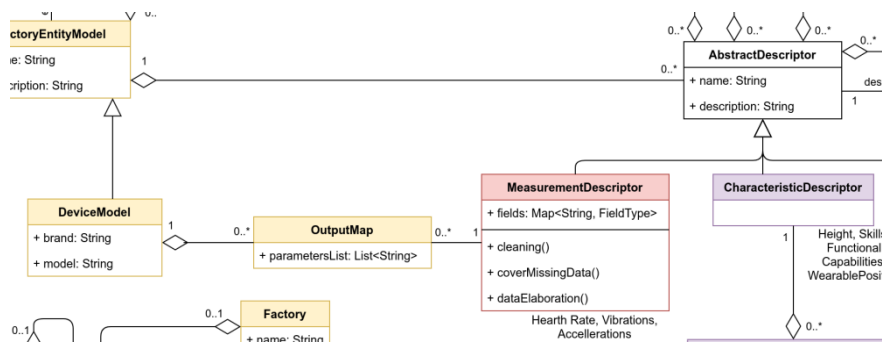


Figure 7 Human Digital Twin reference model: highlight on Measurement Models

3.5.1 MeasurementDescriptor

The **MeasurementDescriptor** extends the **AbstractDescriptor** with a specific class dedicated to dynamic data, streamed by wearable devices, sensors and PLCs. These can include any kind of relevant data collected from factory entities. For example, heart rate, galvanic skin response, vibrations, accelerations, temperature could be modelled through the class **MeasurementDescriptor**.

The **MeasurementDescriptor** is composed by the following attributes:

Attribute	Type	Description
fields	Map<String, FieldType>	This map relates the field data type with the field itself. For example, acceleration is composed by three fields: accX, accY and accZ. This attribute specifies the type of each field.

3.6 State Models

This set of classes aims at describing states characterizing factory entities. States are computed by functional modules, which describe all those computational processes that can elaborate entities and attributes, making the HDT capable of simulating, predicting, reasoning, and deciding. The class **FunctionalModule** aggregates one or more events that can be used to update the state of the HDT depending on the result of the computation performed. Inputs are defined and mapped through the class **FunctionalModuleInput** to

the entities needed by the model (e.g., physiological parameters and worker conditions for detecting fatigue level). Internally, functional models can employ any means of data processing and calculation such as mathematical functions, machine learning, or empirical models. These, if necessary, can be stored as binary blobs and annotated to be managed correctly.

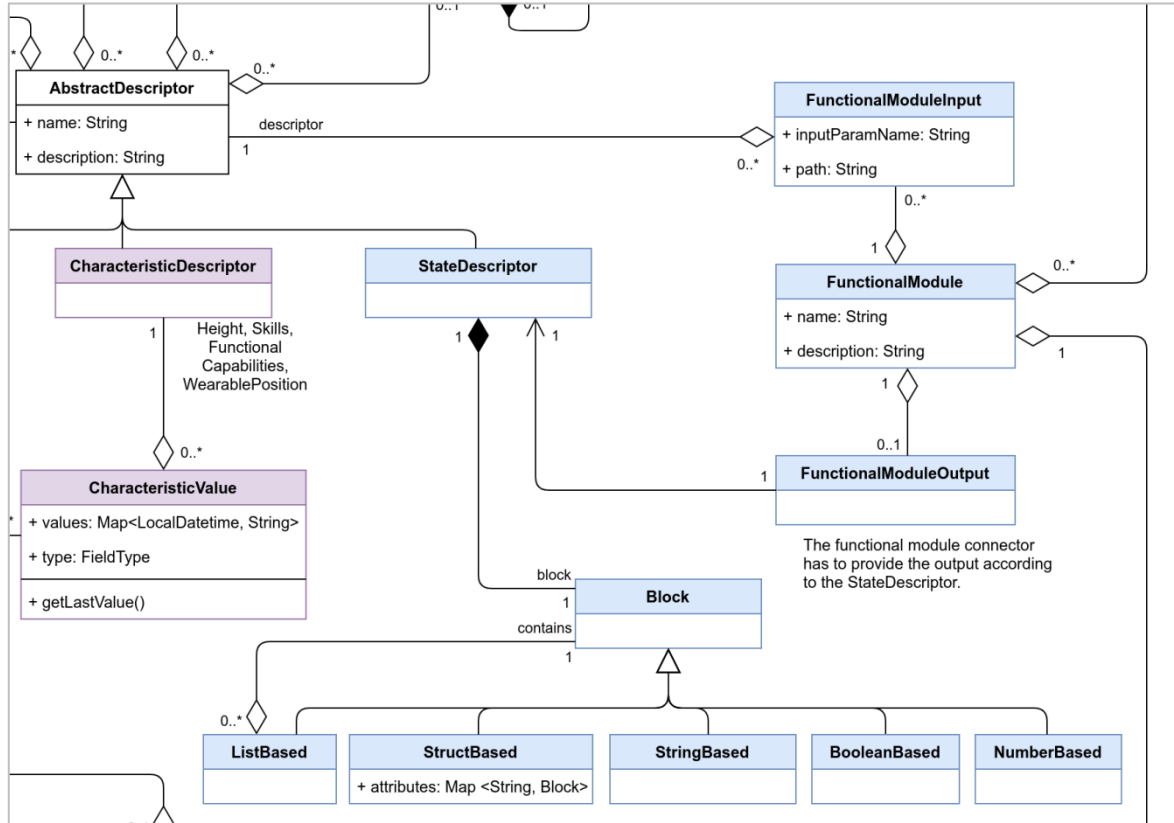


Figure 8 Human Digital Twin reference model: highlight on State Models

3.6.1 StateDescriptor

The **StateDescriptor** extends the **AbstractDescriptor** with a specific class dedicated to the description of states of both workers and things in the factory. These states are computed by the **FunctionalModule(s)**. The **StateDescriptor** can describe the state of a cobot in the form of the current pose, current part program, or the position or the fatigue level of a worker.

The **StateDescriptor** has the following relations:

Class	Relation type	Multiplicity	Description
Block	Aggregation	1...1	The StateDescriptor relates with a Block which is the output schema description provided by a module

3.6.2 FunctionalModule

The **FunctionalModule** class allows to describe any model that generates states. A model could be an AI algorithm or a simple data processor. The **FunctionalModule** defines a model that acts in the HDT and produces outputs that feed the HDT state. An example of **FunctionalModule** is the Fatigue Monitoring System or an Activity Detection Module.

The **FunctionalModule** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the FunctionalModule .
description	String	Human readable description of the FunctionalModule .

The **FunctionalModule** has the following relations:

Class	Relation type	Multiplicity	Description
FunctionalModuleInput	Composition	0..*	This is the relation between FunctionalModule expected inputs, and HDT available AbstractDescriptors. This relation allows the HDT to translate the module input parameter names to actual MeasurementDescriptor(s) , CharacteristicDescriptor(s) and StateDescriptor(s) .
FunctionalModuleOutput	Composition	0..*	This is the relation that allows the HDT to map, to FunctionalModule 's outputs to a StateDescriptor .
InterventionDescriptor	Composition	0..*	This is the relation between the FunctionalModules supporting the decision making and the InterventionDescriptor , that allows to trigger interventions on the production system.
FactoryEntityModel	Composition	0..*	This relation specifies the factory entities that are relevant for the FunctionalModule , i.e., that must be monitored by the FunctionalModule . For example, the Fatigue Monitor System is interested in monitoring new workers.

3.6.3 FunctionalModuleInput

The **FunctionalModuleInput** supports the mapping that relates an **AbstractDescriptor** modelled into the HDT to an input that feeds the **FunctionalModule**. It allows the HDT to integrate modules that require input data with different structures than those described by **MeasurementDescriptor(s)**, **CharacteristicDescriptor(s)** and **StateDescriptor(s)**.

3.6.4 FunctionalModuleOutput

The **FunctionalModuleOutput** supports the mapping that relates the outputs of a **FunctionalModule** and the related **StateDescriptor(s)** into the HDT. It allows the HDT to integrate modules that provide output data with different structures than those described as a **StateDescriptor**.

3.6.5 Block

The **Block** class is a utility entity that enables the formalization of the schema that describes the **StateDescriptor(s)**, thus allowing the HDT to be aware of the data format. This helps **FunctionalModule**'s output validation against the schema, and also to translate/map an output schema of a module to an input schema of another module.

The **Block** structure describes every object's schema which contains coherent lists (a list that comprises elements of the same kind). For example, the location of different workers could be represented as follow (for simplicity the JSON format is used):

```
{
  roomNumber: 10,
  workers: [
    {id: 10, x: 10, y: 20, pose: "stand"},
    {id: 20, x: 23, y: 40, pose: "sit"},
    {id: 30, x: 10, y: 20, pose: "lying down"},
  ]
}
```

This example of output could be easily described through the use of the Block class:

```
StructBased workerBlock:
  "id" → NumberBased
  "x" → NumberBased
  "y" → NumberBased
  "pose" → StringBased
ListBased workersListBlock describes a list of workerBlock
StructBased overallBlock describes the whole object as follows:
  "id" → NumberBased
  "workers" → workersListBlock
```

3.6.6 ListBased

The **ListBased** class inherits from **Block** and describes a list that contains elements structured as a **Block**, enabling the definition of a coherent list that comprises data of the same type.

3.6.7 StructBased

The **StructBased** class inherits from **Block** and describes a structure of **Block(s)**. It can be seen as a dictionary with string keys and Block values.

The **StructBased** is described by the following attributes:

Attribute	Type	Description
attributes	Map<String, Block>	The mapping of a String key to a Block value

3.6.8 StringBased

The **StringBased** block describes a simple attribute of type string, which can be part of a **StructBased** or **ListBased**.

3.6.9 NumberBased

The **NumberBased** block describes a simple numerical attribute, which can be part of a **StructBased** or **ListBased**.

3.6.10 BooleanBased

The **BooleanBased** block describes a simple boolean attribute, which can be part of a **StructBased** or **ListBased**.

3.7 Interventions Models

This set of classes allows to describe interventions to orchestrate the production system and the things acting within it, optimising performance and/or increasing workers' wellbeing.

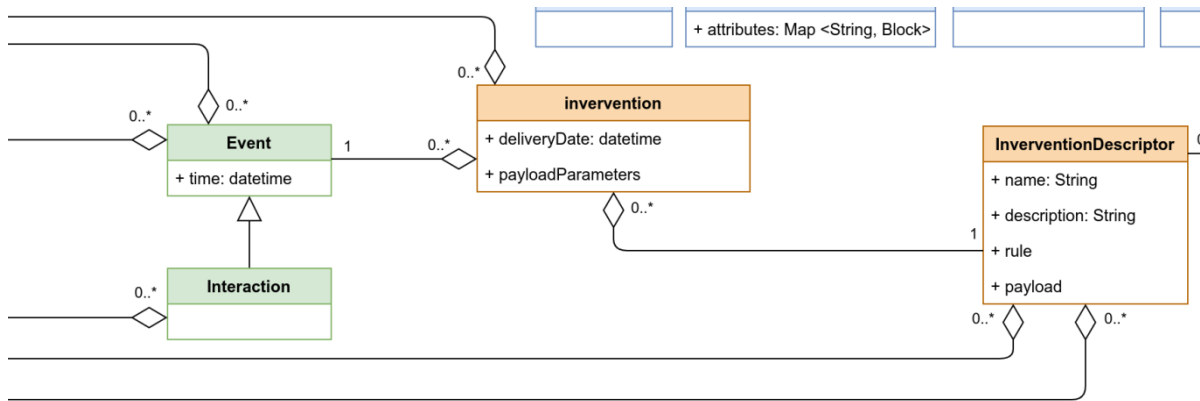


Figure 9 Human Digital Twin reference model: highlight on Intervention Models

3.7.1 InterventionDescriptor and Intervention

The class **InterventionDescriptor** allows the definition of interventions that can be triggered to orchestrate the production system and its entities. Examples of intervention descriptors are: to deliver a notification to the operator, to set-up and activate a robot part-program, to turn-on a tool or to adjust the speed of a spindle. Interventions are fired by the **FunctionalModule(s)** in charge of decision-making, defining workers and the things of the factory to be triggered, the **payload** and the parameters to be delivered.

The **InterventionDescriptor** is described by the following attributes:

Attribute	Type	Description
name	String	Name of the FunctionalModule .
description	String	Human readable description of the FunctionalModule .
rule		Rule/Condition that fires the intervention.
payload		The payload is the part of transmitted data that is the message to be interpreted by the things in the factory, triggering the physical intervention.

The **InterventionDescriptor** has the following relations:

Class	Relation type	Multiplicity	Description
FactoryEntityModel	Composition	1..*	Relation with factory entity models affected by the intervention.
EventDescriptor	Composition	1..*	The EventDescriptor of the event that is triggered by the Intervention .

The class **Intervention** describes a triggered and actuated intervention.

The **Intervention** is described by the following attributes:

Attribute	Type	Description
payloadParameters		Parameters included in the payload characterising the intervention.
deliveryDate	datetime	Date and time when the intervention has been delivered to the factory entity.

The **Intervention class** has the following relations:

Class	Relation type	Multiplicity	Description
FactoryEntity	Composition	1..1	The FactoryEntity targeted by the Intervention
Event	Composition	1..1	The Event triggered by the Intervention .

4 The architecture of the Human Digital Twin

This section provides an overview of the architecture of the STAR’s HDT, its main components and how it will be used to support the integration of WP5’s modules.

The HDT architecture presented in this section and the model presented in section 3 are positioned in the STAR overall architecture respectively in the green and orange boxes depicted in Figure 10.

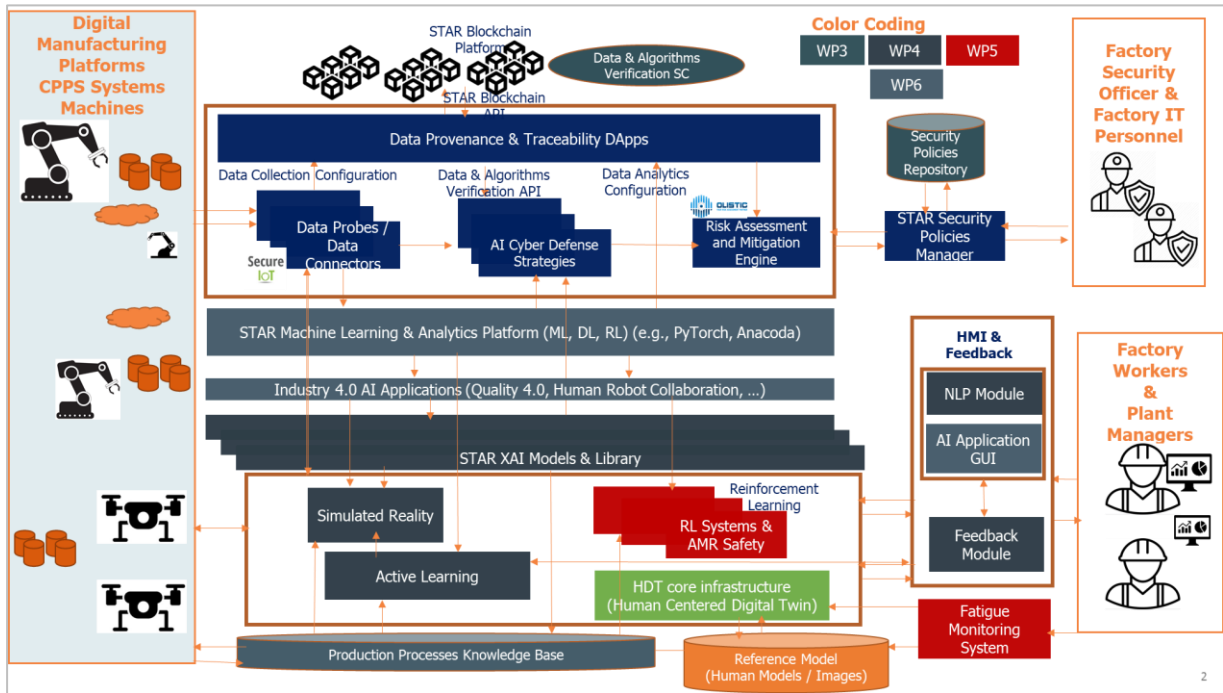


Figure 10 STAR's architecture

4.1 The technological framework for human digital twins

Taking inspiration from works describing the technologies and architectures behind the creation of a HDT [REF-30][REF-29] and, more in general from those dedicated to DT in a broader sense [REF-31][REF-32], 3 technological layers, as shown in Figure 11, are fundamental to realise a HDT, embracing all the cutting edge technologies and methodologies that enable the digitization of a worker, together with his/her characteristics, conditions and behaviours.

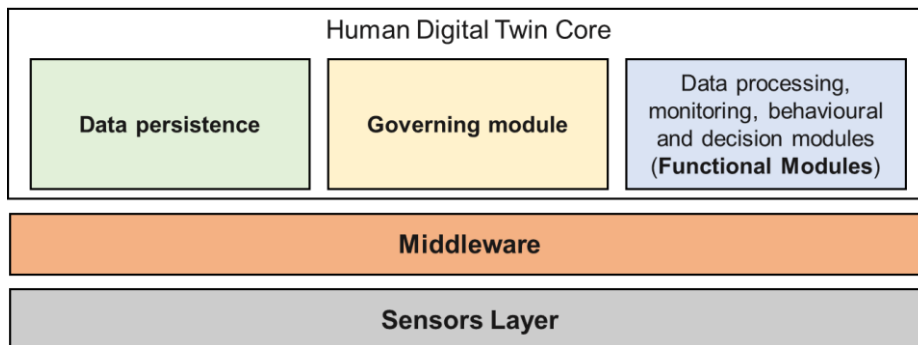


Figure 11 Abstract Digital Twin Architecture

The **Sensors Layer** represents a connection from the physical to the digital world, in charge of the **creation of data** from the physical production system and of the **communication** in quasi-real time using standard data formats. In the case of the human, the installed devices to realise part of such connection are mainly wearable sensors that fetch psychophysical parameters like Heart Rate (HR), Skin Conductance (SC) or Galvanic Skin Response (GSR), while the machinery present at the shop floor needs to be fitted with sensors that generate useful data to be shared with the upper layers. In the case of a HDT, to connect wearables and sensors with the upper layers, a gateway is often the best solution to be adopted. [REF-33]. This is particularly helpful in case of limited battery capacity on wearables side, allowing to have these devices smaller, more comfortable and cheaper. A similar concept to the gateway is applied to machinery and robots, since most of them use industrial standard protocols like OPC-UA, Euromap, or "simply" offer a PLC communication interface. In this case, the gateway acts as a middleware that translates the machine protocols into the unique standard communication protocol known by the whole HDT. This approach allows to realise a simple plug-and-play architecture, requiring only to build specific gateways without changing anything upstream. Gateways can be installed on many types of devices, including smartphones, computers, or a raspberry. The key factor to consider is that these devices need to be able to join a network connection and to have enough capacities to keep a constant flow of data to and from the Middleware, which is the next layer that resides between the HDT Core and the Sensors Layer. In this regard, the NTN 5G represents a relevant enabler that allows to have low latency and high reliability data streams, without the need for wired connections [REF-34]. **Low-level computation** or **pre-processing** of data may reside in the Sensing Layer, depending on computational requirements and applications. It is necessary to consider that, if not properly performed, pre-processing may lead to information loss. However, if executed correctly, relevant benefits can be obtained in terms of communication efficiency, security and scalability.

The **Middleware** enables an active connection from and to the HDT Core. Since it has to manage to withstand high-frequency data coming from multiple sensors installed in the Sensors Layer, the Middleware has to be well designed and implemented to **exploit data flows** from physical to digital layer. It has to be capable to empower the **coherent integration** between the models and the physical architecture, enabling a seamless usage of data for **verification** and **validation** of complex behaviours. There are various tools and technologies that support the implementation of this layer. The most commonly used solutions are based on MQTT [REF-35], a well-known and established data exchange protocol already widely used in the IOT and industrial world. One of the alternatives that implement MQTT is Apache Kafka [REF-36], which offers many useful functions including short-term memory to keep a backlog of the last exchanged messages. However, in some use cases, a more simple and light approach is suggested to the favour of performance, such as the one adopted by Mosquitto [REF-37].

The third layer is the core of the HDT, where **data persistence**, **governing module** and the various **data processing**, **monitoring**, **behavioural** and **decision modules (Functional Modules)** are located. In the data persistence, it is possible to find all the data **storage functionalities** for storing streams from sensors, including databases and data-models, describing the entities and features and historical data. For this latter functionality, it is recommended to use a time-series database like InfluxDB [REF-38] or QuantumLeap [REF-39], optimized for this kind of data. Finally, there are functional modules, which are capable to **process**, **simulate**, **predict**, **reason** and **decide**. The best

approach to include data processing, monitoring, behavioural and decision modules is to adopt a series of plug and play services. The key benefit of such approach, obtained thanks to a proper architecture design, is that modules can be easily removed, added, or extended [REF-40]. An alternative is the development of specific plugins. However, this requires the development of dedicated SDK and imposes a specific programming language. Meanwhile using services, the developer is free to select whichever programming language since the communication interface is universal.

Functional Modules can be of different nature. They can be focused on the data post-processing. This can include data validation, classification, aggregation, sorting, and cleaning. Monitoring Modules focus on specific features and attributes, monitoring their evolution and elaborating raw data to compute more complex information and detecting possible deviations. The Decision Modules identify decisions through the HDT in order to intervene in the digital and/or in the physical world. The Behavioural Modules elaborate on the current status of the HDT to make predictions and simulate its evolution.

Artificial Intelligence (AI) plays a relevant and prominent role in the creation of such kind of modules. One of the main goals of AI is to build software systems, models and algorithm capable to perform complex tasks [REF-41]. The behaviours and variables related to human being are much more complex than those that can characterize machines and robots. Furthermore, humans in a system are infinitely more unpredictable and have greater degrees of freedom than a machine, which is usually stationary, always performs the same tasks, has a set of standard components. For these reasons, AI is of major relevance for creating a HDT, which allows to create models without actually knowing the relationship between the inputs and the outputs. Without the help of AI, it would be very complex to define heuristic relationships between, for example, given physiological data such as HR, HRV, etc. and physical or mental stress. However, AI alone is not enough. It is necessary to consider that AI needs valuable data to build effective models and algorithms. Moreover, when humans are modelled with AI, it is also fundamental to consider related ethics and explainability issues.

4.2 The core architecture

The HDT of STAR is organised into three main technological layers according to the Abstract Digital Twin Architecture shown in Figure 11 and is composed of the following components:

- **Shop-floor entities, agents and gateways (Sensors Layer):** sensors, wearables and PLCs collect and stream data from the shop floor. In order to facilitate data collection from workers and shop-floor entities, the HDT integrates both agents and gateways meant to provide data collection, harmonisation, and accessibility from heterogeneous sources, creating bridges between these sources and upper layers.
- **IIoT Middleware (Middleware):** this layer supports M2M connectivity and is based on the lightweight messaging protocol like MQTT. It enables bidirectional communication under a publish-subscribe mechanism and the organisation of large amounts of heterogeneous data into multiple topics. Each user can be associated to a set of topics into which the data are streamed and gathered by the modules for further computation.
- **Data storage and Time Series Data Storage (Human Digital Twin Core – Data Persistence):** in the data storage component, all the structural and core information about the HDT are stored. In addition, the quasi-static data of the

workers are persisted in this component. Meanwhile, the Time Series Data Storage acts as a backlog of sensor data accessible by the different entities of the HDT for making predictions or extract features for computations.

- **Orchestrator and Models (Human Digital Twin Core - Governance module):** this component is responsible for managing all the entities in the HDT. It knows exactly what kind of data each sensor produces, who are the workers online and where their data are published. In addition, it knows the active modules, what information they take as input and what output they publish. Data descriptors, as part of the orchestrator, consist on models, defined by the administrator of the HDT, meant to describe any worker's or contextual feature.
- **Administration Shell and GUI (Human Digital Twin Core - Governance module):** the Administration Shell allows to administrate and configure the HDT. GUI recaps and offers a nice and understandable way to visualize the data flowing in the HDT. It can be extended by adding a dedicated GUI for each Functional Module.
- **Functional Modules (Digital Twin Core - Data processing, analysis and decision modules):** These modules enable the processing of data coming from workers, contextual sensors, or any type of system that publishes data on the IIoT Middleware. These modules aim to detect human state and conditions and compute complex features to allow human and machine decision makers to consider human factors within their execution and control logics.

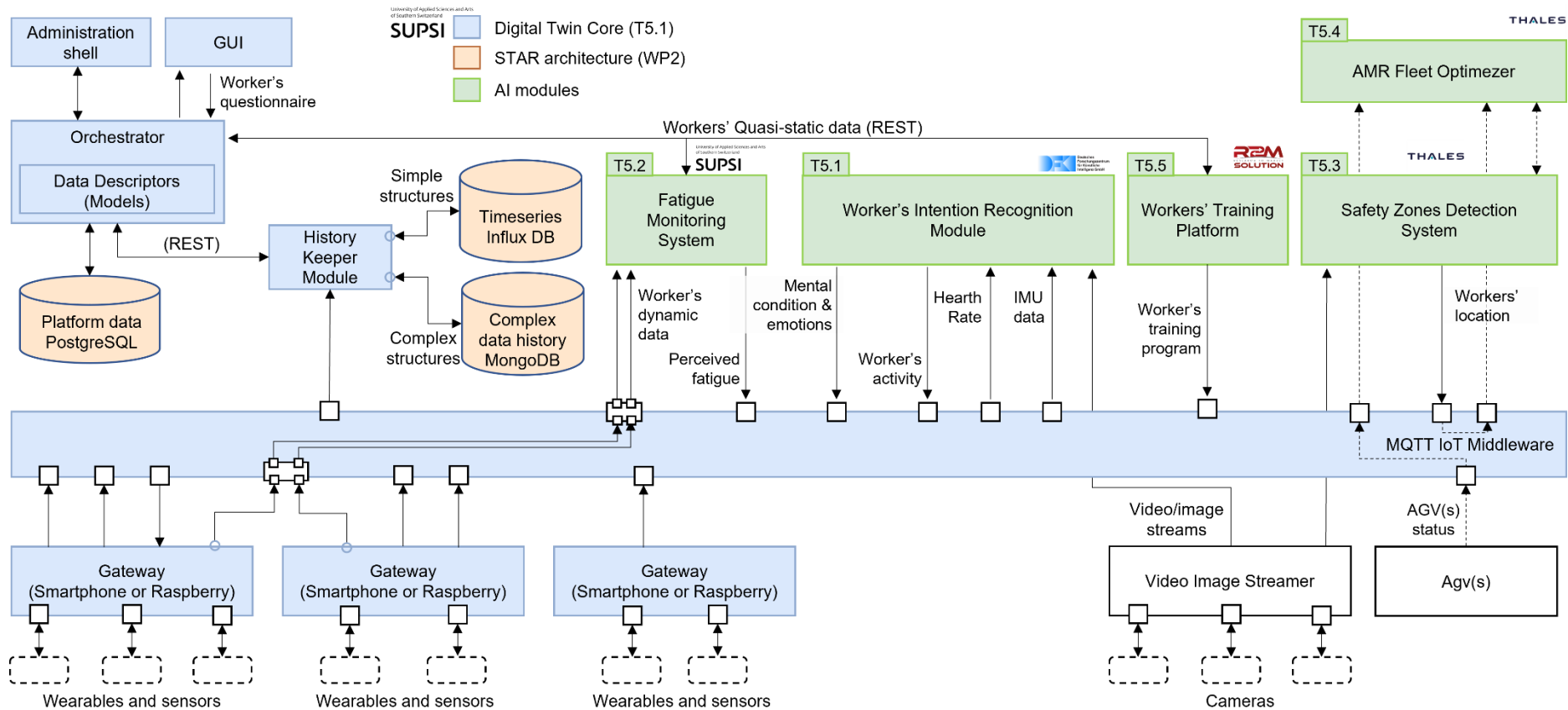


Figure 12 W5 Architecture proposal V1

4.2.1 Agents and Gateway

The **Gateway** is responsible for enabling the connection between the **IIoT Middleware** and the devices, including wearables, creating a secure and reliable communication channel.

The different sensors are connected to the **Gateway** thanks to the **Agents**, software component belonging to the gateway, meant to gather data from the connected devices. If necessary, the **Gateway** could also foresee simple pre-processing algorithms in order to reduce the amount of data transmitted to the **IIoT Middleware (e.g., by averaging chunks of data)**.

Moreover, **Gateways** serve as standard interfaces to access the IIoT middleware, meaning that they are in charge of streaming data to the **IIoT Middleware** according to predefined data formats. In this way, the data harmonisation is preserved by construction, since all the **Gateways** write the information using shared data formats, thus preventing different gateways from writing the same information in different formats. A specific data format is proposed for each kind of information (e.g., heart rate, gsr, etc.), and its documentation will be accessible through the Orchestrator, helping developers to understand how to implement the data gateway properly. Lightweight data formats are preferred (e.g., key-value), in such a way to reduce the overhead needed to transform the raw data coming from sensors into the desired format.

The **Gateway** streams data on a particular topic into the **IIoT Middleware** and feeds a topic for each data provider. As a result, the management of data synchronisation is not needed since data with the same frequency, coming from the same device, are present in a topic. Moreover, the **Functional Modules**, including AI modules like the SUPSI's Fatigue Monitoring System, can work directly with the data coming from the sensors, thus applying their own pre-processing techniques without affecting the data streamed on the **IIoT Middleware**.

4.2.2 IIoT Middleware

The main goal of the **IIoT Middleware** is to make the different data streams available to each component of the architecture. Basically, each connected component can act as a provider and/or consumer. This allows certain components to consume the outputs produced by others (e.g. a **Functional Module** can consume data coming from another **Functional Module**).

The exchanged messages (which structures are described in **Data Descriptors**) are defined in the **Orchestrator** at the component level, so that each connected component can specify the schema of the messages will be published on the IIoT Middleware.

4.2.3 Orchestrator, Administration Shell and GUIs

The **Orchestrator** is the main component responsible for organizing and managing the entire HDT. The administration is done through a series of REST API. The **Orchestrator** knows how the HDT and its architecture are structured. Moreover, it knows who the workers are, which are the installed modules, the connected sensors and the message schemas adopted by the different modules and sensors. The **Orchestrator** provides a set of REST APIs to allow **Functional Modules** to retrieve quasi-static data (e.g., workers skills) from the HDT core.

Moreover, the Orchestrator integrates an **Administration Shell** that allows HDT administrators to define new workers, connect new wearables and sensors, etc. by means of a provided GUI. The latter allows also the visualisation of data, metrics and **Functional Modules** outputs, enabling the user to see the different information exchanged in the HDT. Finally, GUI supports workers' characterisation. To facilitate this activity, a questionnaire is displayed through the GUIs to allow the self-characterization of the workers.

4.2.4 Video image streamer

The video image streamer is intended to allow the various modules to access a video stream of a plant area or work cell. Currently, there are two options:

- Storing the videos into a network video recorder (NVR);
- Using the RTSP protocol to make the video stream available in quasi-real-time.

4.2.5 Persistency Layer

The **Persistency Layer** is provided by the STAR core architecture developed within WP2. To support the HDT implementation, the following databases are integrated:

- PostgreSQL;
- MongoDB;
- Influx.

4.2.5.1 PostgreSQL

PostgreSQL is a powerful and open-source entity-relationship database that leverages and extends the SQL language combined with many features to securely store and virtually scale the amount of data in any application. More information about PostgreSQL can be found at www.postgresql.org.

In the context of the STAR project and the HDT, it is used to store static and quasi-static data of the HDT, mainly described by CharacteristicDescriptor(s).

4.2.5.2 MongoDB

MongoDB is a NoSQL database that stores data in JSON-like documents. Since there is no limit to the schema of each stored document, it is very flexible and easy to use. More information about MongoDB can be found at www.mongodb.com.

The flexibility of MongoDB fits particularly well with the HDT of STAR, where each **Functional Module** can have its specific features and output schema. In the HDT, the main goal of MongoDB is to keep the history of the complex data generated by the various **Functional Modules**, allowing to keep a backlog of every prediction or estimation made by each installed module.

4.2.5.3 Influx

InfluxDB is an open-source database designed specifically for processing time-series data. It is ideal for any big data use case where the time dimension is as important as the data itself (e.g., use cases involving IoT sensors data and real-time analytics). More information about InfluxDB can be found at www.influxdata.com.

In the context of the HDT of STAR, InfluxDB is used in a similar way to MongoDB, but instead of storing the complex data that come from the modules, it supports the storage of high-frequency data coming from the various sensors installed in the shop-floor or worn by workers. The decision to use Influx instead of keeping everything in MongoDB is to allow a more flexible and efficient storage of sensor values. Influx is optimized for fast, high-available storage and retrieval of time series data.

4.2.6 History Keeper Module

The **History Keeper Module** is responsible for storing all data published to the **IIoT Middleware**. It is subscribed to every topic flowing on the **IIoT Middleware**. As soon as a new information is published, **History Keeper Module** stores this data to a dedicated time series in InfluxDB or in a dedicated JSON in MongoDB, depending on the data sources (sensor or **Functional Module**). This allows to retain historical information used especially by those modules that visualize time series data, perform analysis and optimisation, or train artificial intelligence models.

The **History Keeper Module** must also manage the data retention policy, in order to avoid storing data for too long.

4.3 The Functional Modules

The **Functional Modules** are pluggable components, including AI modules, that can be easily added or removed from the HDT. These **Functional Modules** can be subscribed to the IIoT Middleware topics of their interest, in order to use data streams from sensors and other **Functional Modules'** outputs to perform their computations. In addition, **Functional Modules** can also retrieve quasi-static data from the HDT by means of REST API provided by the **Orchestrator**. The **Functional Modules** also have the possibility to publish their outputs to the **IIoT Middleware**.

4.3.1 Fatigue Monitoring System

The **Fatigue Monitoring System** is a software component whose purpose is to detect possible psychological (e.g., loss of attention, mental fatigue) or physical (e.g., tiredness) discomfort or harmful situations for a worker. The **Fatigue Monitoring System** relies on physiological data streamed from wearables to the **Gateways**, and from the **Gateways** to the **IIoT Middleware** to which the module is subscribed. The physiological data are enriched with the quasi-static information stored in the HDT. The **Fatigue Monitoring System** uses these enriched data to compute the perceived exertion level of the worker. The computation output is then published on the **IIoT Middleware** in order to make this information available to the other components.

4.3.2 Integration with the HDT core

The required **Agents** and **Gateways**, to stream data from workers, will be developed to be compliant with and integrated in the HDT core to enable streaming of data (e.g., hearth-rate, accelerations) through the IIoT Middleware. A few **Agents** and **Gateways** will be developed by SUPSI. The **Fatigue Monitoring System** will retrieve data streams from the **IIoT Middleware**. The **Fatigue Monitoring System** will publish computation results on the **IIoT Middleware** to update the HDT according to a specific data structure.

Figure 13 provides an overview of the integration from the point of view of the architecture.

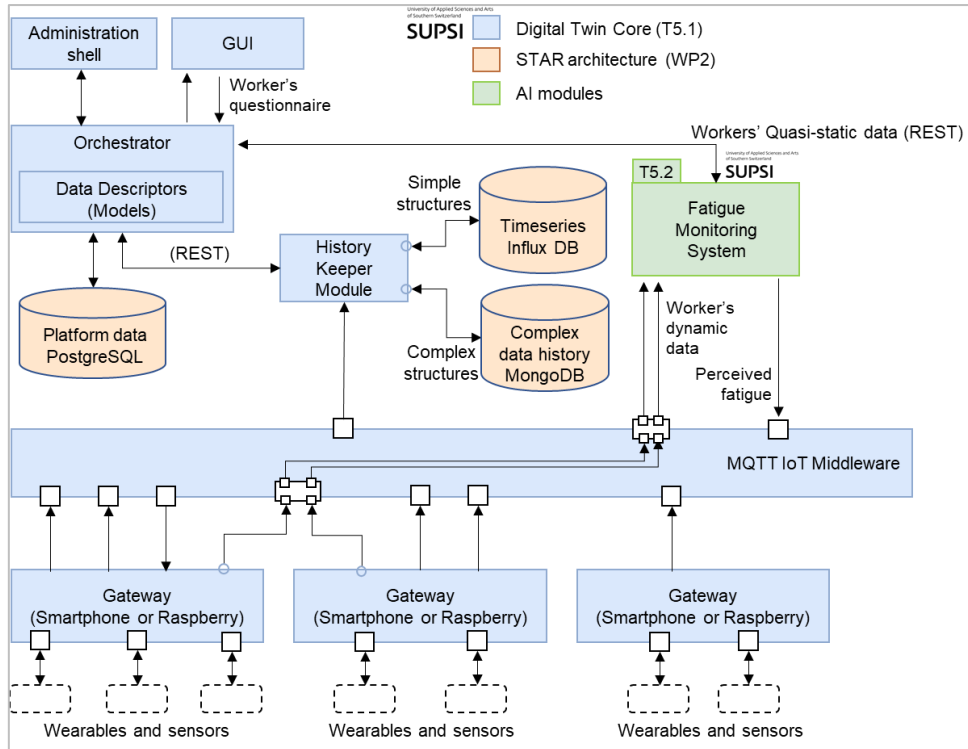


Figure 13 Integration overview: HDT and Fatigue Monitoring System

4.3.3 Worker’s Intention Recognition Module

The **Worker’s Intention Recognition Module** is in charge of:

- Estimating the mental and emotional state of the workers. This is done by acquiring the heartbeat data (EDA) or the electroencephalogram (EEG), the sound signal of the facial muscle movements and the surface pressure mechanomyography (MMG). The source of the aforementioned psychophysical are: smart watches, stethoscope microphones on smart helmets and forehead textile pressure mechanomyography.
- Detecting the type of activity that the operator is performing in a given period of time. The required data come from various IMU sensors such as the Shimmer 3 and also from a smart watch (i.e. Apple watch). This allows to collect inertial and biophysical data that is necessary to perform the worker’s activity recognition task.

The aforementioned sensors require a sampling frequency of 30 fps for the video stream, the EEG at approximately 250Hz, and the facial pressure data at 20Hz (more details in Section 6).

4.3.4 Integration with the HDT core

The required **Agents** and **Gateways**, to stream data from workers, will be developed to be compliant with and integrated in the HDT core to enable streaming of data (e.g., hearth-rate, accelerations) through the IIoT Middleware. Specific **Agents** and **Gateways** may be developed or the ones already developed by SUPSI for its Fatigue Monitoring System will be customized and re-used. The **Worker’s Intention Recognition Module** will retrieve data streams from the **IIoT Middleware**. The latter does not support video/image streams. Therefore, a dedicated solution has to be realised. The **Worker’s Intention Recognition**

Module will publish computation results on the **IIoT Middleware** to update the HDT according to a specific data structure.

4.3.5 Safety Zones Detection System and AGV Fleet Optimizer

The goal of the **Safety Zones Detection System** is to detect and locate people in critical infrastructure. In addition, it detects moving objects (e.g. robots) and static objects that have been moved in the scene and which may be new obstacles in the future. As input, it takes a RTSP stream and it provides as real-time output in a JSON object containing the timestamp, 3D or 2D positions and the type of entity being observed. The goal of the AGV Fleet Optimizer is optimize the control a whole fleet of AMRs, taking into account a dynamic environment, in particular the presence of humans, thanks to video analytics.

4.3.6 Integration with the HDT core

The integration with these two modules will be the most challenging one for the HDT. Feasibility assessments are still on-going. The plan is currently to integrate both modules in the HDT.

The **Safety Zones Detection System** will rely on its own data and video/image stream architecture. **Safety Zones Detection System** will publish only computation results (mainly objects and workers' positions) on the IIoT Middleware to update the HDT according to a specific data structure.

The **AGV Fleet Optimizer** will access to this data via HDT, together with the status of the AGV. Thanks to this information, it will be able to define the best paths for the AGVs. These will be published on the HDT. From the HDT Middleware, the AGVs will access to this data, adapting their behaviour.

Figure 14 provides an overview of the integration from the point of view of the architecture.

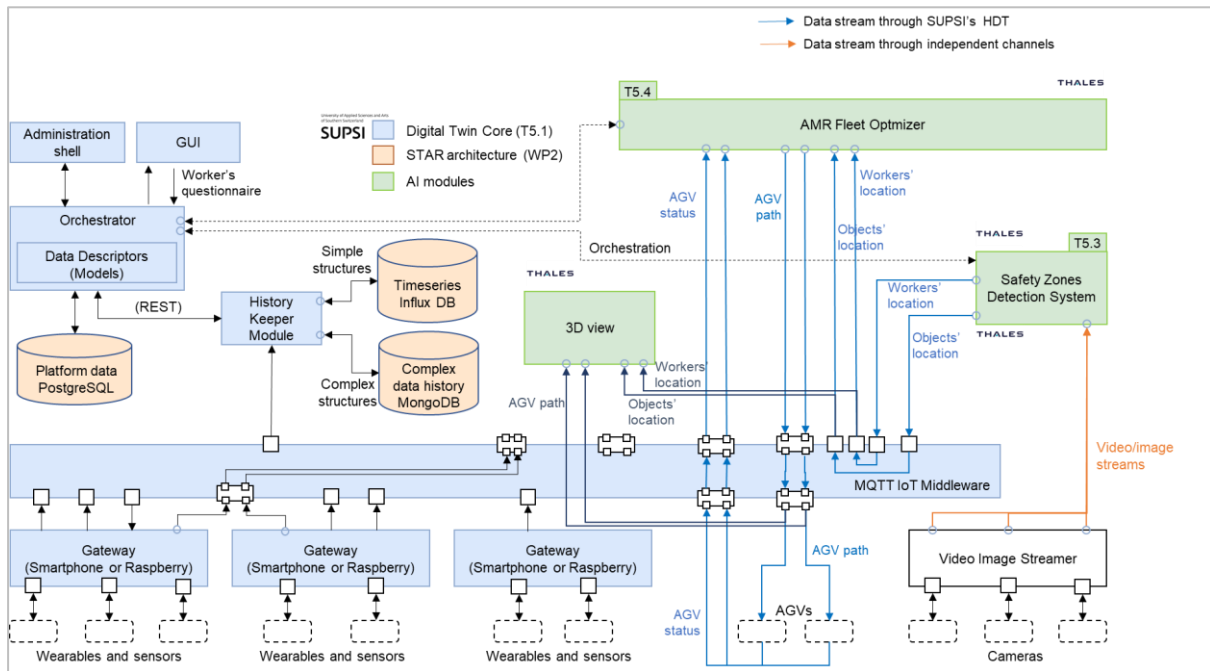


Figure 14 Integration overview: HDT, Safety Zones Detection System and AGV Fleet Optimizer

4.3.6.1 Workers Training Platform

The Workers’ Training Platform is the Web entry point for workers who want to carry out training and continuous learning actions on the different modules and tools that are part of the project. The platform will be composed of 3 main parts:

- Recommendation of training programs;
- Documents and materials on the different modules and solutions developed and used within STAR;
- Access point to the different simulation environments

In order to facilitate the development of each solution and the post project exploitation, the integration will attempt to be as decoupled as possible. The goal will be for the other WP5 modules/tools to reside on their own platforms, and for the Workers' Training Platform to be the entry point to these platforms.

4.3.6.2 Integration options with the HDT core

The **Workers Training Platform** will rely on workers’ quasi-static data stored in the HDT and collected through a dedicated questionnaire (developed in collaboration with SUPSI). The HDT’s GUIs will provide the access to the questionnaire. The questionnaire will be generated by the questions formulated by SUPSI, mainly fitting the requirements of the Fatigue Monitoring System, and stored in the HDT, and the questions formulated by R2M, and retrieved via API from the **Workers Training Platform**. These questions will allow to create a complete characterization of the worker.

A user can access to the questionnaire via the HDT’s GUIs. When he/she completes the questionnaire, his/her answers are collected and saved in the HDT. Specific procedures for anonymization will be applied in case it is necessary, thanks also to the entity class Workers, introduced in section 3.3. The answers collected that are relevant for the **Workers Training Platform** (e.g., user’s job position) will be delivered to this module. It will then elaborate these answers and it will provide the computation results, which will, in turn, be stored in the HDT, to allow other modules and users to access them.

Figure 15 provides an overview of the integration from the point of view of the architecture.

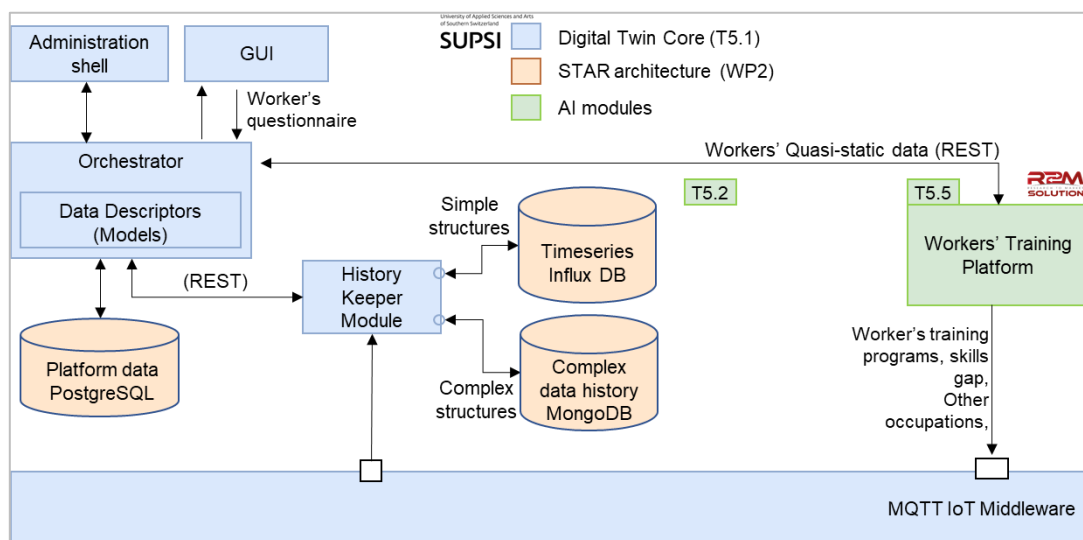


Figure 15 Integration overview: HDT and Workers’ Training Platform

5 Implementation of the core architecture

This section describes the implementation of the core architecture components realised within the first 10 months of task 5.1. Most of all the components are implemented as Java 11 applications, and all of them are released also as dockerized applications, in such a way to ease their deployment in different environments.

The components are available at <https://gitlab.com/star-ai/human-digital-twin-core-infrastructure>.

5.1 The IIoT Middleware

STAR's HDT targets the manufacturing industry and industrial applications. The IIoT Middleware is a crucial component within the STAR HDT, since it is the component in charge of supporting the data exchange among all the satellite modules. Therefore, the IIoT Middleware must be based on a well-established and reliable communication protocol. While different protocols there exist (e.g., WebSocket², CoAP³), MQTT⁴ represents the one that better fits the STAR project requirements.

The MQTT protocol [REF-42] is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc. The MQTT protocol is relevant in the IoT context as it comes with the following properties [REF-43]:

- **Simplified communication:** MQTT reduces communication complexity. It allows a single connection to a message topic. Moreover, data is logically structured and can be processed in a flexible way.
- **Eliminate polling:** MQTT allows instantaneous, push-based delivery, eliminating the need for message consumers to periodically check or “poll” for new information. This dramatically reduces network traffic.
- **Dynamic targeting:** MQTT makes discovery of services easier and less error-prone. A publisher can simply post messages to a topic, instead of maintaining a roster of peers to which an application can send messages.
- **Decoupling and scaling:** MQTT makes solutions more flexible and enables scale. It allows changes in communication patterns, adding or changing functionality without sending ripple effects across the system.

However, the adoption of MQTT as the communication protocol for the HDT core infrastructure does not prevent further extensions to support other communication protocols. The HDT architecture has been designed to be flexible enough and accommodate new protocols by means of specific connectors.

² https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API?retiredLocale=en

³ <https://coap.technology/>

⁴ <https://mqtt.org/>

5.1.1 Analysis and benchmark of existing solutions

Different solutions supporting the MQTT-based data exchange there exist. For this reason, a comparative analysis has been carried out to identify the best available solution fitting the HDT requirements within the STAR projects, namely:

- To manage a large variety of different sensors sending high-frequency data (100-200 Hz).
- To be compatible with different programming languages.
- To persist massive data sets, generated by high-frequency data.
- To support the event-based communication (publish/subscribe mechanism).
- To guarantee high reliability in both receiving/delivering messages/events.

Solutions not fitting the above requirements have been excluded from the analysis (e.g., Apache Kafka⁵, Orion Context Broker⁶, Redis⁷, or ZeroMQ⁸, which do not natively support the MQTT protocol), as well as those ones for which it was not possible to carry out an adequate implementation (i.e., Moscow and JoramMQ, which are particularly validated by the literature, but lack a Java client to support their adoption). At the end of the selection phase, four of the most relevant MQTT-based open-source solutions, currently available, have been taken into account:

- Eclipse Mosquitto⁹
- RabbitMQ¹⁰
- HiveMQ¹¹
- EMQ¹²

To benchmark the middleware, different publishers and subscribers have been implemented (as Java applications) to test the data transmission from/to each of them.

5.1.1.1 The test setup

An instance of each data broker has been deployed on a private server featuring a 8-core CPU and 32GB of memory. Brokers have been configured to use MQTT v3 or v5, depending on the supported version. A proper Quality of Service (QoS) level has been set for the data broker (the QoS level is “guarantee of delivery” agreement between the sender and the receiver). Authentication and security options have not been considered within the analysis, but most of the solutions allow users to set a proper configuration.

The benchmark suite is a Java 11 application implementing broker managers based on generic publisher/subscriber components capable of producing/reading messages to/from the data broker. The generic publisher/subscriber components define the interfaces that actual publishers and subscribers must implement; these components are customized for each individual solution in a separate package (see Figure 16 for more details about the

⁵ <https://kafka.apache.org/>

⁶ <https://fiware-orion.readthedocs.io/en/master/>

⁷ <https://redis.io/topics/introduction>

⁸ <https://zeromq.org/>

⁹ <https://mosquitto.org/>

¹⁰ <https://www.rabbitmq.com/>

¹¹ <https://www.hivemq.com/>

¹² <https://www.emqx.io/>

relationships between components). Each custom component (e.g., Mosquitto publisher and subscriber) is connected to the specific middleware by means of an appropriate connector (which is usually a third-party library released by the solution provider).

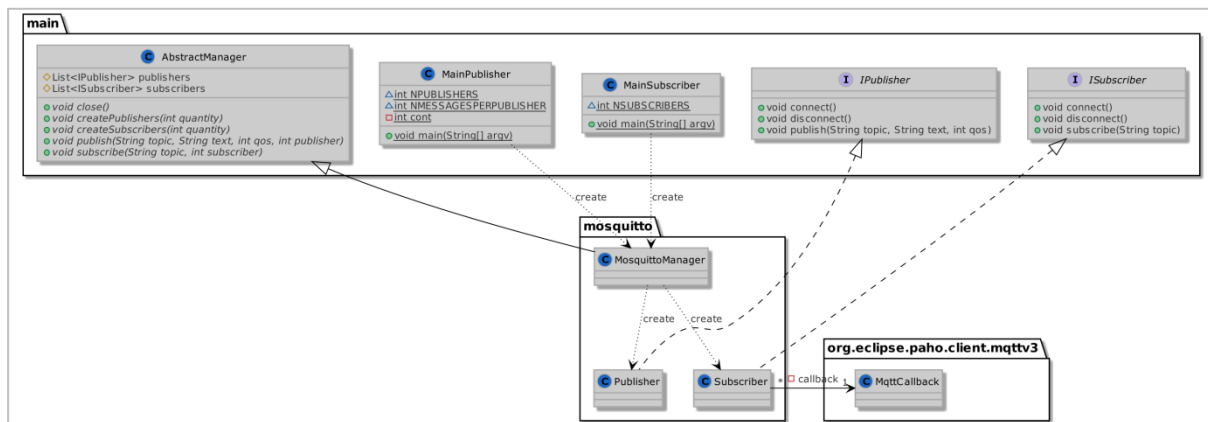


Figure 16. The class diagram of the Java application used in the experiment (other packages than "mosquitto" have been omitted for the sake of diagram clarity)

The Java application instantiates and runs a fixed number of custom publisher/subscriber (which has been set to 10 in our test) by means of managers, which take care of instantiating the lists of publishers and subscribers, calling the creation method of their respective classes.

The benchmark suite has been executed to measure (i) the test duration, and (ii) the number of messages correctly delivered. The values shown in Table 1 and Table 2 are an average of 5 executions. The test has been carried out with different QoS levels. There are 3 QoS levels in MQTT:

- At most once (0) - no guarantee of delivery
- At least once (1) - the message is delivered at least one time to the receiver
- Exactly once (2) - each message is received only once by the recipient

The QoS parameters have been changed in the various executions, as well as the number of messages, publishers and subscribers involved, to see how these factors affect the execution. To measure the run time, the application monitors the time needed by a publisher to publish the required number of messages: it would not have been easy to verify the stop condition for subscribers, while a publisher ends its execution as soon as it finishes to publish the messages. Also, the reported run times do not take into account the initialization of the various components, but only the sending times.

The test has been carried out on groups of 10 publishers / 10 subscribers per broker with a varying number of sent messages (results in Table 1). The test with 10000 messages has been also repeated with groups of 100 publishers / 100 subscribers (results in Table 2). For simplicity, all tests were carried out by publishing and reading from a single topic but involving multiple topics should lead to similar results.

The benchmark suite for the different experiments has been executed on a machine with an Intel Core i7-7700HQ 2.80GHz CPU and 16GB RAM, mounting a Windows 10 Home operating system. The Java version is 11.0.2 LTS. An important premise is that the "Sent" column indicates the total number of messages sent (number of messages per client *

publisher number), while the “Received” columns indicate the total number of messages received, which will be further multiplied by the number of subscribers, being all registered on the same topic. Example: 10 messages per publisher, 10 publishers, 10 subscribers. In an optimal case, Sent = 100, Received = 1000. Long runs (over 10 minutes) are highlighted in orange.

The tests have been carried out using adopting a synchronous approach, where the tested middleware remains blocked until it does not provide a response to the client. It does not process messages in parallel.

Table 1. Test results with 10 publishers and 10 subscribers

Broker	Messages delivered	t QoS 0 (ms)	Messages received	t QoS 1 (ms)	Messages received	t QoS 2 (ms)	Mesages received
EMQ	100	47	995	18777	955	37472	952
	1000	167	9998	192290	9700	> 10 min.	
	10000	886	99982	> 10 min.		> 10 min.	
HiveMQ	100	368	997	3547	914	7603	922
	1000	847	9992	33627	9486	71323	9466
	10000	10084	83004	385806	104444	> 10 min.	
Mosquitto	100	88	999	131	992	159	998
	1000	253	9985	800	9959	1119	9985
	10000	1536	99928	4637	99875	6393	99822
RabbitMQ	100	156	980	169	990	/	
	1000	756	9990	825	9990	/	
	10000	5249	99990	5299	99980	/	

Table 2 Test results with 100 publishers and 100 subscribers

Broker	Messages delivered	t QoS 0 (ms)	Messages received	t QoS 1 (ms)	Messages received	t QoS 2 (ms)	Messages received
EMQ	10000	1597	75477	> 10 min.		> 10 min.	
HiveMQ	10000	5407	99948	345224	95118	648078	95339
Mosquitto	10000	1656	99926	5741	99816	7078	99803
RabbitMQ	10000	6715	99990	6721	99990	/	

Results in Table 1 and Table 2 show that Mosquitto is the best performing solution, followed by RabbitMQ. Moreover, the following aspects have been taken into account while choosing the STAR HDT core middleware:

- **Mosquitto**: it proved to be the best in terms of performance and it is characterized by a fairly simple implementation.
- **RabbitMQ**: it is the second choice, and it requires a quite simple implementation, but more complex than the one required for Mosquitto.
- **EMQ**: despite being simple at the implementation level, the performance is poor. It struggles especially with many publishers / subscribers.

- **HiveMQ:** one of the most complete brokers at the implementation level, since it allows developers to design different types of producers/subscribers (synchronous, asynchronous, blocking, parallel). Despite its completeness, the performance is not optimal.

In this analysis, the performance of the various brokers are not affected by a specific implementation.

5.1.2 Implementation details

Deploying the Mosquitto MQTT middleware does not require any further implementations, while all the components that need to access it (e.g., the Orchestrator and the Historical Data Keeper) have to implement a proper connector (similar to the one developed for running the benchmark). The connector is a component that enables the publish/subscribe mechanism and represents the data abstraction layer for each component. Specifically to the Mosquitto broker, each component will be based on the recommended Eclipse Paho¹³ client for MQTT. The Mosquitto broker can be natively installed on different OSs,¹⁴ or alternatively can be deployed as a Docker container using the official Docker image.¹⁵

5.2 The Orchestrator

The **Orchestrator** is the component in charge of managing the modules attached to the HDT. New functional modules consuming/producing data from/to the HDT can be registered and configured by means of the Orchestrator. Moreover, the Orchestrator manages the creation of new topics on the IIoT middleware. The Orchestrator must support functional modules and gateways in:

- Fetching their configuration, so that to properly initialise themselves; for example, a functional module configuration contains the list of AbstractDescriptor to use as input data.
- Retrieving the location where input data are stored; indeed, the mapping <AbstractDescriptor: Topic> is known only by the Orchestrator (which embeds the logic for creating topics according to a predefined topic structure).
- Getting the location of the topic where to publish new data.
- Discovering the data structure of data produced by other functional modules as output states.

5.2.1 Implementation details

The Orchestrator has been implemented as a Java Spring application. It embeds the HDT data models as introduced in Section 3. The model is directly mapped to a SQL database (i.e., PostgreSQL) to store the HDT configuration. The mapping has been realized via JPA annotations, while the connection to the database is managed by the Hibernate ORM.

The Orchestration Administration Shell is missing from the current release, but the REST API to support its functionalities has been already developed. The API is documented in Swagger, by using the OpenAPI specification standard. The API exposes the CRUD operations for all the HDT model entities, plus additional methods to ease the interaction

¹³ <https://www.eclipse.org/paho/>

¹⁴ <https://mosquitto.org/download/>

¹⁵ https://hub.docker.com/_/eclipse-mosquitto

with the Administration Shell. functional modules, and gateways (e.g., a gateway may need to recover the list of workers that are not involved in a session).

The Orchestrator has been developed by using interfaces and their actual implementation; as a result, it is easier for developers to adapt the Orchestrator to different IIoT middleware (e.g., Kafka). Moreover, since the topic creation logics is completely delegated to the Orchestrator, the external components (i.e., functional modules and gateways) are agnostic from the actual topic structure, making it easier to change it at any time, when needed.

The actual broker connector, the list of topics to use for notifications, measurements, and states, are properties fully configurable from configuration files.

Finally, the Orchestrator is released also as a dockerized application, in such a way to ease its deployment as Docker container.

5.3 The Historical Data Manager

The MQTT protocol represents the de-facto standard to support the publish/subscribe mechanism among IoT devices, which publish data to their subscribers. However, data published by sensors must be exposed to enable external functional modules running analytics and reporting activities within STAR. Thus, historical data must be maintained (i.e., by storing them in a database). Most MQTT brokers do not provide built-in mechanisms to keep the historical data. Within the STAR HDT architecture, this missing functionality is covered by the Historical Data Manager (HDM).

The HDM acts like a data sink within the HDT core architecture: data flowing into the MQTT middleware are transferred to persistent databases, whatever their nature and origin, ready to be queried by external modules. Within the HDT core, data flowing into the MQTT middleware may be of two different kinds: measurements or states (according to the naming convention explained in Section 3.5 and 3.6). While timeseries databases are best suited for measurements, NoSQL databases are more indicated for states, which represents complex objects with arbitrary structures. For this reason, the HDT core architecture features two different databases to support the historical data storage: InfluxDB for measurements, and MongoDB for states.

The HDM has a dedicated module (HDM-Web) exposing also a REST API to make historical measurements accessible as timeseries, given a window (more details are given in the next section). Concerning the historical states, functional modules can directly run queries against the MongoDB database, so that to preserve the expressive power of MQL (MongoDB Query Language).

5.3.1 Implementation details

The HDM module is composed by 2 components implemented as Spring applications, which together offer two main functionalities:

- 1) to forward data from the IIoT middleware to databases storing historical data (InfluxDB and MongoDB)
- 2) to make historical data accessible by other modules

Concerning functionality 1), the HDM embeds three different connectors to the involved databases (i.e., Mosquitto, InfluxDB and MongoDB). Also, in this case, the connectors implement a predefined connector interface, allowing users to extend the HDM to connect to

different data sources (e.g., MongoDB can be replaced with a different NoSQL database and messages can be read from a different MQTT broker).

The application works simple: once started, the HDM announces itself to the HDT Orchestrator, asking for the list of existing MQTT topics, as well as the topic to subscribe for receiving notifications about new created topics. Then, it subscribes itself to all the topics, waiting for new messages.

When a new message arrives on a topic, the HDM just transfers it to the right database. To this end, it is worth noting that the HDM can classify at any time if a message is a measurement or status message, thus it is possible to transfer the message to the appropriate database.

The HDM stores also the acquisition time of each measurement; in this way, the original acquisition time is considered when querying the data as timeseries. This operation is crucial, because the arrival order of messages sent over the MQTT middleware may differ from the original sending sequence.

For functionality 2), a dedicated module (HDT-Web) exposes two specific HTTP requests to allow external modules fetching historical data as timeseries. Having a dedicated module to expose data to third-party modules allows decoupling modules from the actual storage systems, increasing also the security.

External modules can query the HDT-Web module by passing a window that may be of two types:

- a **time** window, specified by the couple <start timestamp, end timestamp>. The HDM return the list of measurements with a timestamp falling into the given time window. The size of the returned list varies depending of the measurements sampled in the given window.
- a **sized** window, specified by the couple <timestamp, n >, where n is a scalar number). The HDM return the list of measurements, starting from the given timestamp (going backward). The size of the returned list is fixed (and it is equal to n).

It is worth remarking that the HDM does not store characteristic descriptors, which are directly stored by the Orchestrator.

As for the Orchestrator, also the HDM is released as a Docker container.

5.4 The Gateway and Agents

The **Gateway** represents the bridge between the actual sensors and the HDT. It is the component in charge of collecting measurements and sending them to the HDT. To read data from sensors, the Gateway relies on specific **Agents** that serve as connectors to the sensor/device collecting the measurements. The main functionalities supported by the Gateway and its Agents are:

- To handle the connections for the agents, assigning them to specific factory entities within the HDT (e.g., to assign a wearable device to a worker).
- To collect measurements from the agents and send them to the IIoT middleware, according to the specific data format declared by the Orchestrator

- To prevent the data loss in cases when the connection to the IIoT middleware is interrupted (i.e., by temporarily storing the measurements in a local buffer)

5.4.1 Implementation details

The Gateway provided within STAR is an Android 11 application for smartphones. The current release comes with a limited set of agents supporting connections to wearable devices only; further agents will be included in the next releases.

On startup, the application configures itself by issuing some requests to the Orchestrator, querying for:

- the list of workers
- the list of wearable devices, together with:
 - their OutputMap (which is used to format the data according to the format expected by the HDT)
 - their topic (i.e., the topic on the IIoT middleware where to write the measurements provided by the device)

After the startup phase, the application asks the user (assumed to be a worker) to enter her worker ID number (Figure 17), in such a way to set the link between the recorded measurement and the worker (the current release of the application does not provided a proper login phase, which will become available in next releases). Then, the user can establish multiple connections to different wearable devices by clicking the related button in the application UI (Figure 18).



Figure 17. WorkerID input view

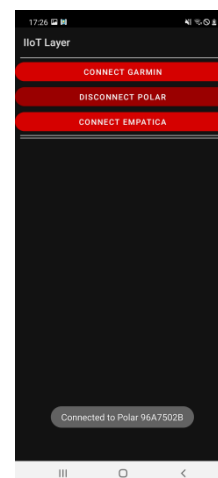


Figure 18. Available connections

To establish the connection, the Gateway instantiates an Agent by passing to it the OutputMap needed to format the received measurements. In this way, data coming from the Agents are already formatted as expected by the HDT, thus the Gateway has only to send the data to the IIoT middleware.

Once the worker is fine with the activated connections, she can start a new Session. In this stage, the Gateway must communicate the initial timestamp of the session to the Orchestrator, together with the worker ID and the list of connected devices.

The Gateway collects a certain amount of data before trying to send them to the IIoT middleware, in such a way as to reduce the remote requests to the middleware. However,

the connection between the Gateway and the IIoT middleware may be unstable, due to temporary network faults. For this reason, the Gateway stores the collected measurements in a local buffer that is emptied when:

- The network is available again and the data can be sent to the IIoT middleware. The waiting period before checking for network availability is a configurable parameter.
- The measurements stored in the local buffer are no more “fresh”, i.e., their timestamp is outside of a given time window (e.g., 10 minutes before). The actual size of the buffer is not fixed, but it depends on the acquisition frequency. The time window of the buffer is a configurable parameter of the application.

The data collection stops when the user decides to end the active session by pushing a button in the application UI. The Gateway communicates the ending timestamp to the Orchestrator, deletes the active Agents, and flushes the current local buffer (if any).

About the Agents, they are components implemented to support the connection to a specific family of devices (the family is usually determined by the device manufacturer). The current Gateway comes with the following Agents:

- **Polar Agent:** this agent is suitable for the devices produced by Polar and it is based on the Polar SDK.¹⁶ The current version of the agent enables the collection of the following metrics: heart rate (HR), RR interval (RR), accelerometer (ACC) and [electrocardiogram](#) data (ECG). The HR and RR metrics are collected every second, while the ACC and ECG data are streamed at a given frequency (25, 50, 100, or 200 Hz).
- **Empatica Agent:** this agent supports the data collection from Empatica devices. As the previous one, it is based on an official SDK for Android.¹⁷ The SDK requires a developer key that can be obtained on the Empatica website by registered a purchased device. This agent is able to collect the following metrics: RR interval (RR), photoplethysmogram (PPG), galvanic skin response (GSR), accelerometer (ACC), and skin temperature (ST).
- **Garmin Agent:** this agent enables the connection with wearable devices produced by Garmin. The agent relies on the *Connect IQ SDK*,¹⁸ which however comes with a few limitations:
 - It requires the *Garmin Connect* application, which may be not available for Gateways that are not released as smartphone applications.
 - A dedicated application must be deployed to the wearable device in order to read the data directly from the onboarded sensors, then pushing them to the Garmin Agent.
 - It supports one connection at a time to a single wearable device.
 - It supports a limited number of wearable devices.

A prototype of this agent has been included in the current Gateway release, allowing users to collect the following metrics: heart rate (HR), oxygen saturation (SAT), and values from the accelerometer (ACC) and the magnetometer (MAG). However, given the above limitations, the next release of the Gateway will possibly include a new

¹⁶ <https://www.polar.com/en/developers/sdk>

¹⁷ <https://developer.empatica.com/#android>

¹⁸ <https://developer.garmin.com/connect-iq/overview/>

version of this agent, based on the *Garmin Health SDK*.¹⁹ Unfortunately, as of today there not exist a free-to-use version of this SDK, thus HDT adopters will have to pay the SDK license to use this agent.²⁰

¹⁹ <https://developer.garmin.com/health-sdk/overview/>

²⁰ For research purposes only, Garmin promotes the usage of this SDK by releasing a limited amount of licenses for free.

6 Worker's Intention Recognition Module

The Worker's Intention Recognition Module predicts user's movement trajectory based on AI models and helps to decide whether a mobile robot in the manufacturing line should move faster, slower, or completely stop to prevent collision between worker and the robot. The module receives sensor and video data which are attached to the user's body and captured by cameras in the manufacturing line. The Worker's Intention Recognition Module is composed of two different modules:

- **Emotion Detection Module:** This module aims to estimate the mental and emotional states of the worker. This is done by acquiring the heartbeat data (EDA) or the electroencephalogram (EEG), the sound signal of the facial muscle movements [REF-44] and the surface pressure mechanomyography (MMG)[REF-45]. The source of the aforementioned psychophysical are smartwatches, stethoscope microphones on smart helmets and forehead textile pressure mechanomyography.
- **Activity Recognition Module:** This module is to recognize worker's activities by using time-series sensor data and vision data from wearable sensors, such as IMU sensors, capacitive sensors, and gyroscope, and camera.

6.1 Hardware Design

The sensing prototype used in the experiment combines three boards: an nRF52840 backend motherboard from Adafruit Feather [REF-46], a customized human body capacitance sensing board, and a data logger board, as depicted in Figure 19.

The nRF52840 board supplies three axes of IMU data, including acceleration, gyroscope, and magnet. The customized body capacitance board is verified efficiently to sense both the body movement [REF-47] and the environmental context [REF-48] by measuring the skin potential signal. The sensor data is stored in an SD card on the data logger board with a rate of 20Hz. Since the environment in the factory is full of 2.4G Hz wireless signals (e.g., WiFi, Bluetooth) to avoid the data package loss, an SD card has been used to record the data locally and finally synchronized by checking some predefined actions. The sensing component, the IMU and body capacitance sensor, consumes the power with a level of sub-mW. A 3.7V chargeable lithium battery is used for the power supply.

Figure 19 provides the different views of the combined sensing prototype.

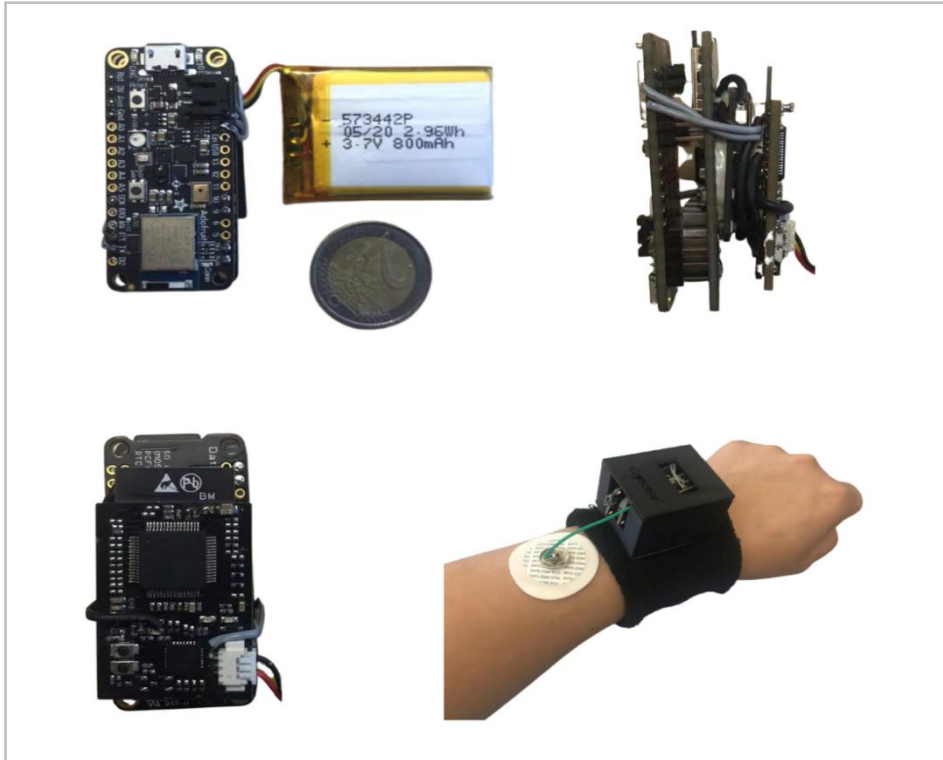


Figure 19 Top (TL), side (TR), left (BL) and worn on the wrist (BR) views of the combined sensing prototype

Figure 20 depicts the signals of the used sensors when the subject repetitively performed actions in an office like opening the door, opening the windows, touching the desktop, and touching the floor lamp. As the plot shows, the actions of the body can be sensed by the four selected sensing modalities efficiently.

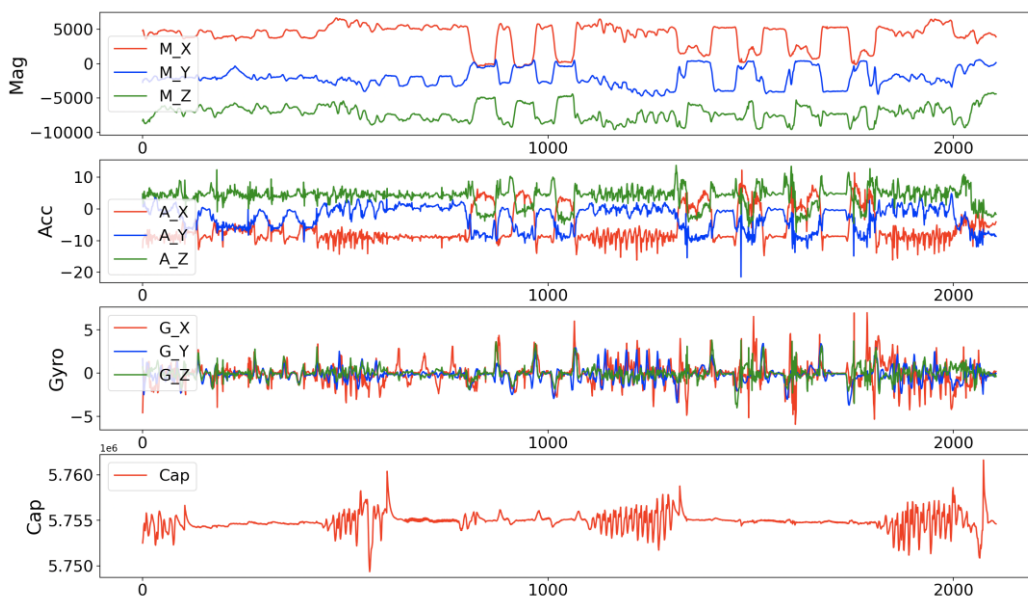


Figure 20 Example of recorded data

6.2 Use case and Experiment Design

To test the Worker’s Intention Recognition Module, a specific use case has been designed. This use case aims to predict human movements intention to plan an optimal robot route without collision risks. In the industrial environment, some workflow processes are done by workers to respond to production reconfiguration needs. It is essential to have the prediction of human behaviour to configure the mobile robot later to avoid possible collisions, as well as to create safety zones. It will be recognized at DFKI’s SmartFactory test lab, in Kaiserslautern, where the latest developments in the industry are tested. The use case will convert algorithm-based human activity into an AI approach and predict their next actions based on their daily activities. The goal of the use case is to keep the production level high, maintaining human safety during robot activity.

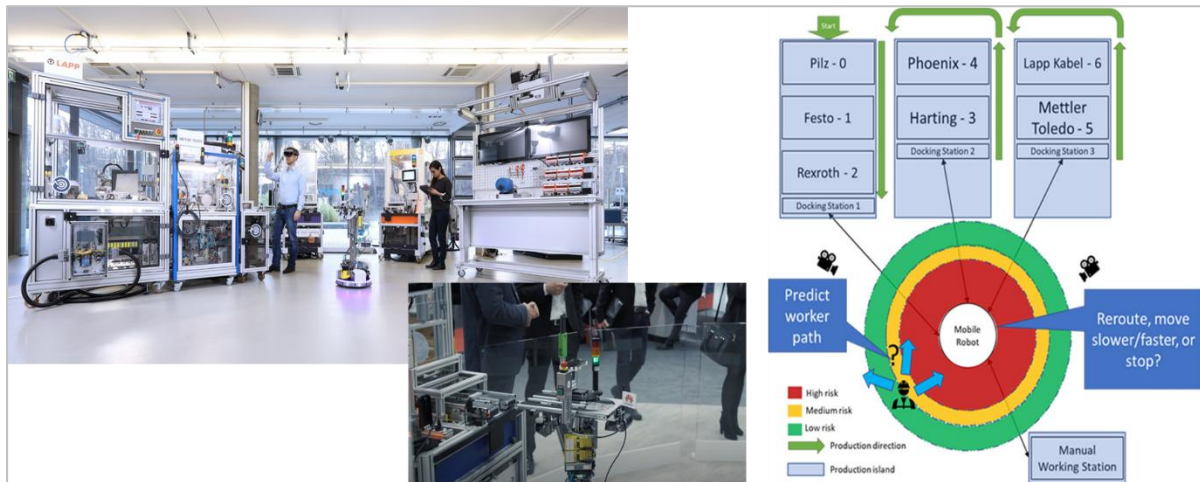


Figure 21 DFKI's SmartFactoryKL test lab and structure of use case

In this use case, it is assumed the presence of a mobile robot and maximum of two people in the testbed at the same time. In addition, a flow chart of worker’s activities has been used in this use case, as shown in Figure 22.

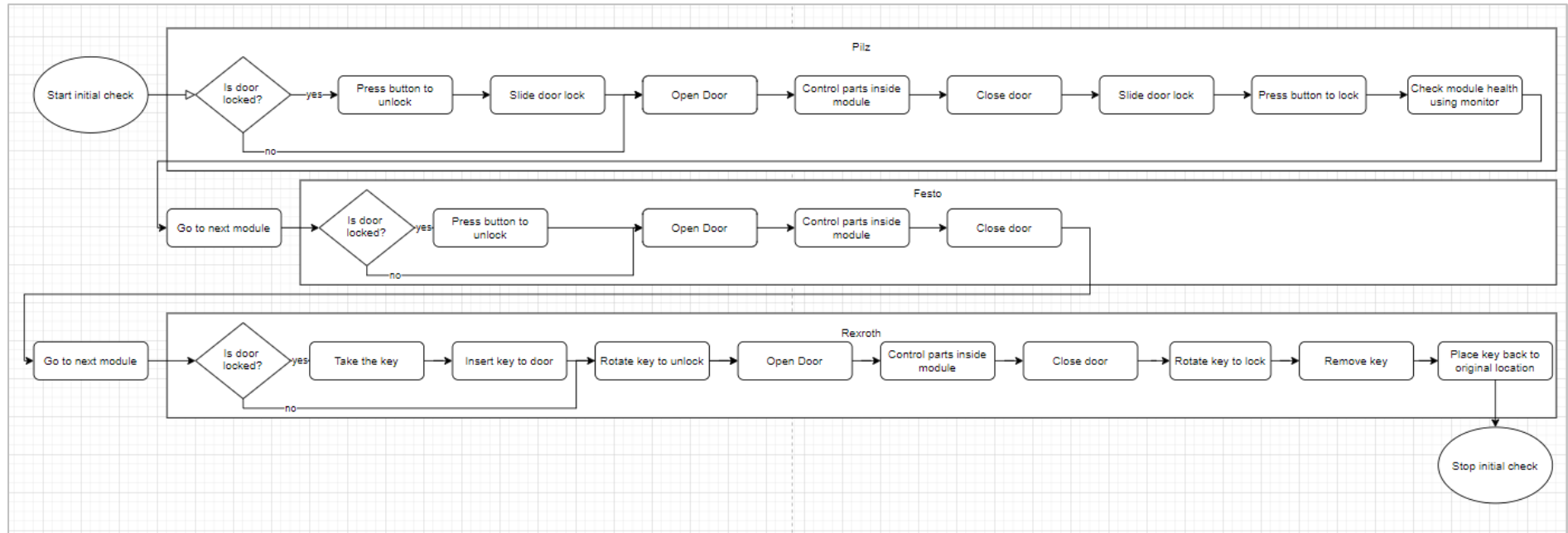


Figure 22 Flow chart of the activities in use case at DFKI's SmartFactoryKL test lab

6.2.1.1 Definition of the use case activities

Following the flow chart depicted in Figure 22, the use case considers 10 worker's activities involving the upper body movements. The activities are defined as follows:

- **Press button:** worker press various types of a physical button on the machine.
- **Slide door lock:** To open the door of the machine on the front side, some doors have a sliding door latch. This activity is sliding the door lock to open the door of the machine.
- **Open door:** To check electric devices or parts in the machine and fix the problems in the machine, the worker should open the upper door or lower door of the machine.
- **Close door:** After checking inside the machine, the worker closes the door.
- **Check parts inside the module:** The worker operates the task of checking and maintaining the inside of the equipment using his/her hands.
- **Go to the next module:** The worker moves to the next module after checking and maintaining the module of the machine.
- **Take the key:** Several doors are locked and require the key to be opened. And, the key is plugged in another hole. Thus, the worker needs to take the key from the hole.
- **Rotate the key:** After taking the key, the worker rotates the key to open the door. And, the worker rotates the key after closing the door.
- **Place key back to original location:** After locking the door, the worker places the key back to the original location/hole.
- **Check door lock:** Before opening the door, the worker checks whether the door is locked or unlocked.

6.2.1.2 Experiment Design

Twelve volunteers were asked to follow the flow chart of the activities with the prototype of the wearable sensors on both wrists. The experiment was divided into five sessions. Each session took around 2-3 minutes and some sessions were conducted and recorded in a different direction from the flow chart. The participants consist of two women and ten men. There were two left dominant hand users and ten right dominant hand users. The experiment was conducted at DFKI's SmartFactoryKL test lab and without any calibration per user. All participants signed an agreement following the policies of the university's committee for the protection of human subjects. The experiment was video recorded for further confidential analysis. The observer and the participant followed an ethical/hygienic protocol according to the public health guidelines.

The collected data have 10 channels: three channels (axes) of acceleration, three channels of gyroscope, three channels of a magnet, and one channel of body capacitance data. We have synchronized recorded video data, left wrist sensor data, and right wrist sensor data. Based on the video data, we have annotated the user's activities as the defined 10 different activities manually.

The acquired sensor data are shown in Figure 24. Each class has obvious signals according to its motion type and is somehow differentiable. We plan to design neural networks based on convolution neural networks (CNN) [REF-49], long short-term memory (LSTM) [REF-50], Transformer[REF-51], and other representative models to recognize the user's activity. We expect that we will be able to predict the worker's next action with the recognized current

activity and flow chart. Red indicates magnet data from left wrist sensor, magenta indicates magnet data from right wrist sensor, green indicates acceleration data from left wrist, yellow indicates acceleration data from right wrist sensor, blue is gyroscope data from left sensor, cyan is gyroscope data from the right wrist, and black and grey indicate capacitance data from left and right wrist sensors, respectively.

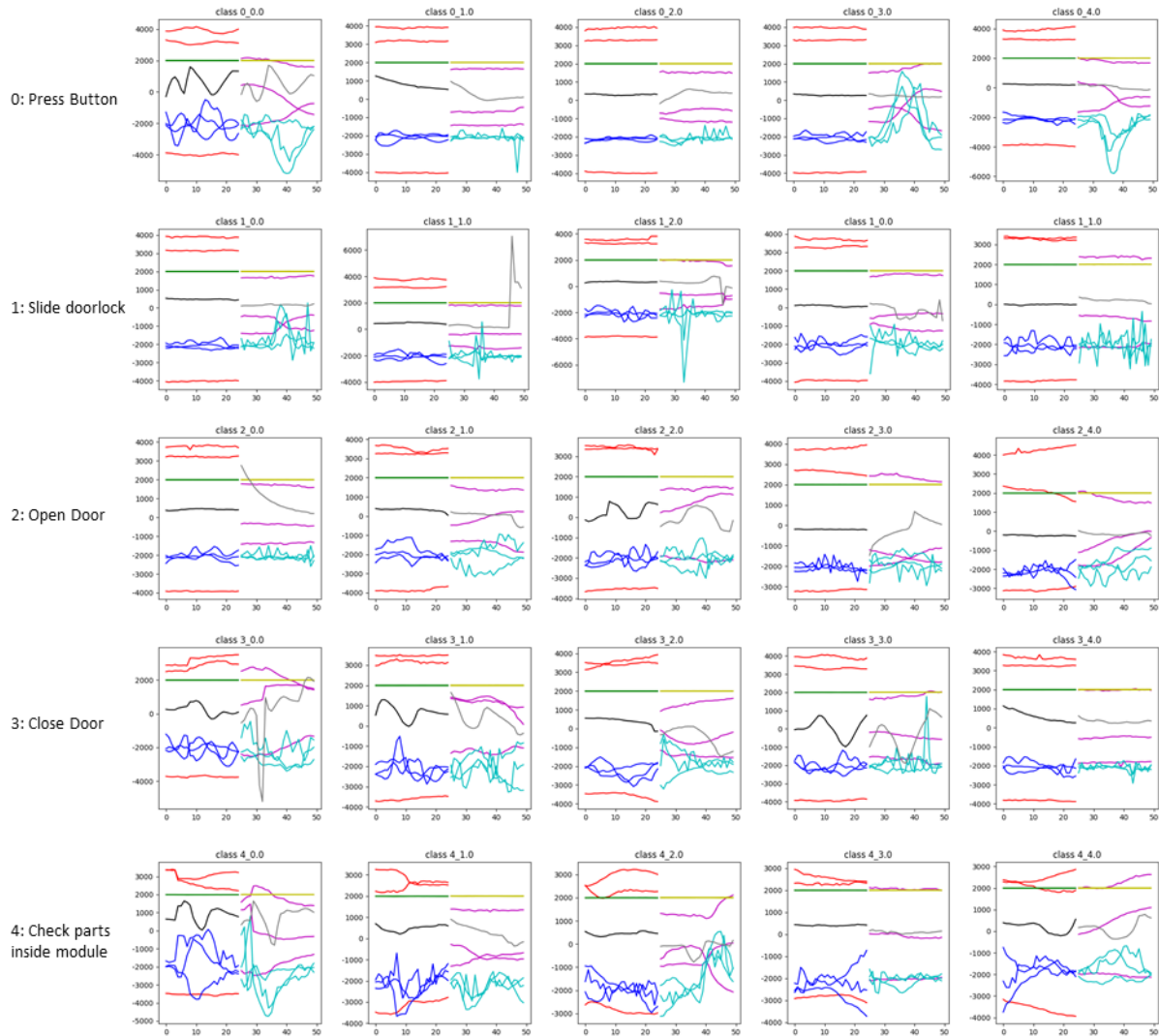


Figure 23 Examples of labelled sensor data: movements 0-4

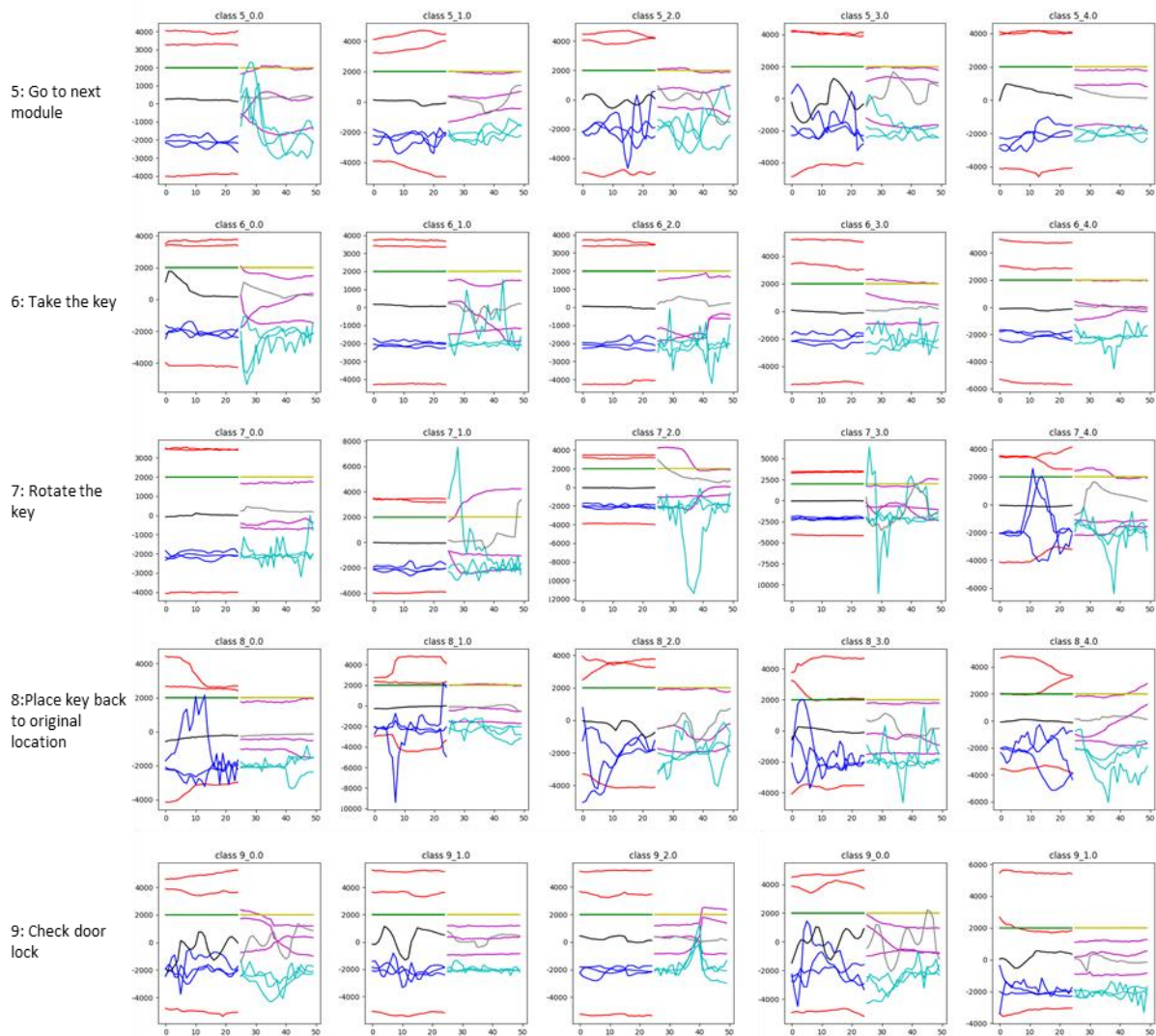


Figure 24 Examples of labelled sensor data: movements 5-9

To further improve the performance of the activity recognition, we plan to apply generative adversarial networks (GAN) [REF-52] and domain generalization methods [REF-53][REF-54]. If the performance of the activity recognition by using sensor data only is not quite good, we will combine the sensor data with video data and feed it into the neural networks.

7 Conclusions

This document has presented a key cornerstone for STAR and WP5 goals: the envisioning of a reference model for the development of Human Centric AI-based Production Processes. This model allows developers of the different AI modules to rely on a commonly agreed formalization of the entities. Moreover, the proposed infrastructure supports the instantiation of HDTs, facilitating the integration and the data sharing between the different AI modules, relying on a unique source of truth for workers and related contextual data.

The next steps will involve the extension of the currently available solution, mainly adding new connectors and gateways and integrating additional brokers. Finally, specific support will be provided to WP5 to use and rely on the HDT core infrastructure to support the development of HDT relying on their components.

References

- [REF-01] Negri, E., Fumagalli, L., & Macchi, M. (2017). A review of the roles of digital twin in CPS-based production systems. *Procedia Manufacturing*, 11, 939-948.
- [REF-02] Stadnicka, D., Litwin, P., & Antonelli, D. (2019). Human factor in intelligent manufacturing systems-knowledge acquisition and motivation. *Procedia CIRP*, 79, 718-723.
- [REF-03] E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and US Air Force vehicles," in Paper for the 53rd Structures, Structural Dynamics, and Materials Conference: Special Session on the Digital Twin, 2012.
- [REF-04] W. Kritzinger and e. al., "Digital Twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, pp. 1016-1022, 2018.
- [REF-05] Sengan, S., Kumar, K., Subramaniaswamy, V., & Ravi, L. (2021). Cost-effective and efficient 3D human model creation and re-identification application for human digital twins. *Multimedia Tools and Applications*, 1-18.
- [REF-06] Berisha-Gawłowski, A., Caruso, C., & Harteis, C. (2021). The Concept of a Digital Twin and Its Potential for Learning Organizations. In *Digital Transformation of Learning Organizations* (pp. 95-114). Springer, Cham.
- [REF-07] Kawamura, R. 2019. A Digital World of Humans and Society—Digital Twin Computing. *NTT Technical Review* 18(3):11–17
- [REF-08] Montini Elias, Bettoni Andrea, Ciavotta Michele, Carpanzano Emanuele, Pedrazzoli Paolo. (2021). A meta-model for modular composition of tailored human digital twins in production. *Procedia CIRP* 104C (2021) pp. 688-694
- [REF-09] Dale, S., Kruger, K., & Basson, A. (2019, January). Human Digital Twin for Integrating human workers in Industry 4.0. In *International Conference on Competitive Manufacturing*, Stellenbosch University, South Africa.
- [REF-10] Z. S. Maman, M. A. A. Yazdi, L. A. Cavuoto and F. M. Megahed, "A data-driven approach to modeling physical fatigue in the workplace using wearable sensors," *Applied ergonomics*, vol. 65, pp. 515-529, 2017.
- [REF-11] Z. S. Maman, Y. J. Chen, A. Baghdadi, S. Lombardo, L. A. Cavuoto and F. M. Megahed, "A data analytic framework for physical fatigue management using wearable sensors," *Expert Systems with Applications*, 2020.
- [REF-12] V. Villani, M. Righi, L. Sabattini and C. Secchi, "Wearable Devices for the Assessment of Cognitive Effort for Human–Robot Interaction," *IEEE Sensors Journal*, vol. 20, no. 21, pp. 13047-13056, 2020.
- [REF-13] Peruzzini, M., Grandi, F., & Pellicciari, M. (2017). Benchmarking of tools for user experience analysis in Industry 4.0. *Procedia manufacturing*, 11, 806-813.
- [REF-14] Egilmez, G., Erenay, B., & Süer, G. A. (2014). Stochastic skill-based manpower allocation in a cellular manufacturing system. *Journal of Manufacturing Systems*, 33(4), 578-588.
- [REF-15] Graessler, I., & Pöhler, A. (2017, December). Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)* (pp. 289-293). IEEE.

- [REF-16] Graessler, I., & Poehler, A. (2018). Intelligent control of an assembly station by integration of a digital twin for employees into the decentralized control system. *Procedia Manufacturing*, 24, 185-189.
- [REF-17] D'Addona, D. M., Bracco, F., Bettoni, A., Nishino, N., Carpanzano, E., & Bruzzone, A. A. (2018). Adaptive automation and human factors in manufacturing: An experimental assessment for a cognitive approach. *CIRP Annals*, 67(1), 455-458.
- [REF-18] Paredes-Astudillo Y A, Jimenez J F, Zambrano-Rey G, Trentesaux D. Human-Machine Cooperation for the Distributed Control of Hybrid Control Architecture. In: *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, 98-110, 2019.
- [REF-19] Carpanzano E, Bettoni A, Julier S, Costa J C, Oliveira M. Connecting Humans to the Loop of Digitized Factories' Automation System. In: *International Conference on the Industry 4.0 model for Advanced Manufacturing*, 2018.
- [REF-20] A. Bilberg and A. A. Malik, "Digital twin driven human-robot collaborative assembly," *CIRP Annals*, vol. 68, no. 1, pp. 499-502, 2019.
- [REF-21] Bettoni, A., Montini, E., Righi, M., Villani, V., Tsvetanov, R., Borgia, S., ... & Carpanzano, E. (2020). Mutualistic and adaptive human-machine collaboration based on machine learning in an injection moulding manufacturing line. *Procedia CIRP*, 93, 395-400.
- [REF-22] Baskaran, S., Niaki, F. A., Tomaszewski, M., Gill, J. S., Chen, Y., Jia, Y., ... & Krovi, V. (2019). Digital human and robot simulation in automotive assembly using siemens process simulate: a feasibility study. *Procedia Manufacturing*, 34, 986-994.
- [REF-23] Bortolini, M., Faccio, M., Gamberi, M., & Pilati, F. (2020). Motion Analysis System (MAS) for production and ergonomics assessment in the manufacturing processes. *Computers & Industrial Engineering*, 139, 105485.
- [REF-24] Ferraguti, F., Villa, R., Landi, C. T., Zanchettin, A. M., Rocco, P., & Secchi, C. (2020). A Unified Architecture for Physical and Ergonomic Human-Robot Collaboration. *Robotica*, 38(4), 669-683.
- [REF-25] X. Shen, I. Awolusi and E. Marks, "Construction equipment operator physiological data assessment and tracking," *Practice Periodical on Structural Design and Construction*, vol. 22, no. 4, 2017.
- [REF-26] T. Cheng, G. Migliaccio, J. Teizer and U. Gatti, "Data Fusion of Real-time Location Sensing (RTLS) and Physiological Status Monitoring (PSM) for Ergonomics Analysis of Construction Workers," *Journal of Computing in Civil Engineer*, 2013.
- [REF-27] L. Liu, Y. Zhang, L. Yang, L. Zhou, F. Ren and Wang, "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE Access*, vol. 7, 2019.
- [REF-28] B. R. Barricelli, E. Casiraghi, J. Gliozzo, A. Petrini and S. Valtolina, "Human Digital Twin for Fitness Management," *IEEE Access*, vol. 8, pp. 26637-26664, 2020.
- [REF-29] Bettoni, A., Montini, E., Righi, M., Villani, V., Tsvetanov, R., Borgia, S., ... & Carpanzano, E. (2020). Mutualistic and adaptive human-machine collaboration based on machine learning in an injection moulding manufacturing line. *Procedia CIRP*, 93, 395-400.
- [REF-30] Nikolaos Nikolakis, Kosmas Alexopoulos, Evangelos Xanthakis, and George Chryssolouris. The digital twin implementation for linking the virtual representation

- of human-based production tasks to their physical counterpart in the factory-floor. *International Journal of Computer Integrated Manufacturing*, 32(1):1{12, 2019.
- [REF-31] Redelinghuys, A. J. H., Kruger, K., & Basson, A. (2019, October). A six-layer architecture for digital twins with aggregation. In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing* (pp. 171-182). Springer, Cham.
- [REF-32] Steindl, G., Stagl, M., Kasper, L., Kastner, W., & Hofmann, R. (2020). Generic Digital Twin Architecture for Industrial Energy Systems. *Applied Sciences*, 10(24), 8903.
- [REF-33] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. Iot gateway: Bridging wireless sensor networks into internet of things. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 347{352. Ieee, 2010.
- [REF-34] Temesvári, Z. M., Maros, D., & Kadar, P. (2019). Review of Mobile Communication and the 5G in Manufacturing. *Procedia Manufacturing*, 32, 600-612.
- [REF-35] Gaston C Hillar. *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.
- [REF-36] Nishant Garg. *Apache kafka*. Packt Publishing Ltd, 2013.
- [REF-37] <https://mosquitto.org/>
- [REF-38] Mohammad Nasar and Mohammad Abu Kausar. Suitability of influxdb database for iot applications. *International Journal of Innovative Technology and Exploring Engineering*, 8(10):1850{1857, 2019.
- [REF-39] Kamienski, C., Soininen, J. P., Taumberger, M., Dantas, R., Toscano, A., Salmon Cinotti, T., ... & Torre Neto, A. (2019). Smart water management platform: Iot-based precision irrigation for agriculture. *Sensors*, 19(2), 276.
- [REF-40] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On micro-services architecture." *International Journal of Open Information Technologies* 2.9 (2014): 24-27.
- [REF-41] Martinez-Miranda, J., & Aldea, A. (2005). Emotions in human and artificial intelligence. *Computers in Human Behavior*, 21(2), 323-341.
- [REF-42] <https://mqtt.org/>
- [REF-43] <https://www.u-blox.com/en/blogs/insights/mqtt-beginners-guide>
- [REF-44] Bo Zhou, Tandra Ghose, and Paul Lukowicz. "Expressure: detect expressions related to emotional and cognitive activities using forehead textile pressure mechanomyography." *Sensors* 20.3, 2020.
- [REF-45] Hymalai Bello, Bo Zhou, and Paul Lukowicz. "Facial Muscle Activity Recognition with Reconfigurable Differential Stethoscope-Microphones." *Sensors* 20.17, 2020.
- [REF-46] <https://learn.adafruit.com/adafruit-feather-sense/pinouts>
- [REF-47] S. Bian, V. F. Rey, P. Hevesi and P. Lukowicz, "Passive Capacitive based Approach for Full Body Gym Workout Recognition and Counting," 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom, 2019, pp. 1-10, doi: 10.1109/PERCOM.2019.8767393.
- [REF-48] S. Bian, V. F. Rey, J. Younas and P. Lukowicz, "Wrist-Worn Capacitive Sensor for Activity and Physical Collaboration Recognition," 2019 IEEE International

Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2019, pp. 261-266, doi: 10.1109/PERCOMW.2019.8730581.

- [REF-49] Sornam, M., Kavitha Muthusubash, and V. Vanitha. "A survey on image classification and activity recognition using deep convolutional neural network architecture." In 2017 Ninth International Conference on Advanced Computing (ICoAC), pp. 121-126. IEEE, 2017.
- [REF-50] Ordóñez, Francisco Javier, and Daniel Roggen. "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition." *Sensors* 16, no. 1 (2016): 115.
- [REF-51] Haresamudram, Harish, Apoorva Beedu, Varun Agrawal, Patrick L. Grady, Irfan Essa, Judy Hoffman, and Thomas Plötz. "Masked reconstruction based self-supervision for human activity recognition." In Proceedings of the 2020 International Symposium on Wearable Computers, pp. 45-49. 2020.
- [REF-52] Wang, Jiwei, Yiqiang Chen, Yang Gu, Yunlong Xiao, and Haonan Pan. "SensoryGANs: An effective generative adversarial framework for sensor-based human activity recognition." In 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE, 2018.
- [REF-53] Soleimani, Elnaz, and Ehsan Nazerfard. "Cross-subject transfer learning in human activity recognition systems using generative adversarial networks." *Neurocomputing* 426 (2021): 26-34.
- [REF-54] Suh, Sungho, Vitor Fortes Rey, and Paul Lukowicz. "Adversarial Deep Feature Extraction Network for User Independent Human Activity Recognition." arXiv preprint arXiv:2110.12163 (2021).