Production harmonizEd Reconfiguration of Flexible Robots and Machinery

Horizon 2020 – Factories of the Future, Project ID: 680435

Deliverable 2.3

Specification of the Generic Interfaces for Machinery, Control Systems and Data Backbone

Lead Author: UNINOVA

Version: 1.1
Date: 23.01.2017
Status: Final
Dissemination level: PUBLIC

Version	Date	Content
0.1	21.06.2016	Table of Contents
0.2	24.10.2016	First draft of the deliverable. Compilation of the contributions from several partners.
0.3	03.11.2016	Draft description of each of the data model's elements. Added a section regarding the modelling of WP2 Bragança's Workshop cell.
0.4	22.12.2016	Fraunhofer and SmartFactory contribution
1.0	06.01.2017	Review version of the deliverable. Compiled the missing contributions of the different partners. Refined the text from the first draft version and added missing parts.
1.1	23.01.2017	Compilation of feedback on the review version. Aligned references and acronyms.

Author List:

André Hennecke (SmartFactory)
André Rocha (UNINOVA)
Arnaldo Pereira (IPB)
Daniel Schel (Fraunhofer)
Frederik Gosewehr (HSEL)
Jeffrey Wermann (HSEL)
João Matos (UNINOVA)
José Barbosa (IPB)
Mafalda Rocha (UNINOVA)
Olha Meyer (Fraunhofer)
Ricardo Peres (UNINOVA)

Abstract

With the emergence of the Industry 4.0 concept, or the fourth industrial revolution, the industry is bearing witness to the appearance of more and more complex systems, often requiring the integration of various new heterogeneous, modular and intelligent elements with pre-existing legacy devices.

This challenge of interoperability is one of the main concerns taken into account when designing such systems-of-systems, commonly requiring the use of standard interfaces to ensure this seamless integration. To aid in tackling this challenge, a common format for data representation, as well as standard interfaces for data exchange, interoperability and service exposure should be adopted.

Aligned with PERFoRM's Industry 4.0 vision, this document details the design of a common data model and two standard interfaces, supported by the results of previous successful European projects, which can be used as some of the core elements to enable the seamless integration and interconnectivity between both intelligent and legacy systems, taking into account the specific needs of four different use cases representing varied European industry sectors.

Furthermore, the entire modelling process of a demonstration punching cell is thoroughly described, developed as part of the WP2/WP3 Integration Workshop at the IPB facilities.

Table of Contents

List of Figures

List of Tables

Acronyms

Abbreviation	Explanation
AMS	Agile Manufacturing System
API	Application Programming Interface
CAD	Computer Aided Design
CPS	Cyber-physical System
DB	Database
ERP	Enterprise Resource Planning
GRACE	InteGration of pRocess and quAlity Control using multi-agEnt technology
HTTP	Hypertext Transfer Protocol
HW	Hardware
JSP	JavaServer Pages
KPI	Key Performance Indicator
MAS	Multiagent System
MES	Manufacturing Execution System
OEE	Overall Equipment Efficiency

PLC	Programmable Logical Controller
PRIME	Plug and produce intelligent multi-agent environment based on standard technology
QA	Quality Assurance
R&D	Research and Development
RE	Requirements Engineering
REST	Representational State Transfer
RMS	Reconfigurable Manufacturing Systems
ROA	Resource Oriented Architecture
SCADA	Supervisory Control and Data Acquisition
SE	Simulation Environment
WP	Work Package
XML	eXtensible Markup Language
AMS	Agile Manufacturing System
CAD	Computer Aided Design
CPS	Cyber-physical System

1. Introduction

1.1. Objective of the document

This deliverable contains the outcome of Task 2.3, entitled "Design of Standard Interfaces for Machinery, Control Systems and Data Backbone", which entails the design of the common PERFoRM data model, as well as of the standard generic interfaces, which will act as the main drivers of interoperability and seamless exchange of information between the different system elements.

To this end the requirements derived from WP1 and WP2, as well as the deliverables from each of the use cases, namely D7.1 [1], D8.1 [2], D9.1 [3] and D10.1 [4] were used as guidelines throughout the developments of the task.

In line with the overall PERFoRM vision, results from previous successful R&D projects in the field were also taken into account, more concretely the FP7 PRIME and GRACE projects were used as a basis for the work documented hereafter.

1.2. Structure of the document

The document is divided into eight main sections. Starting from Section 2, a summary of the generic requirements derived from WP1 is presented, followed by a further analysis in Section 3 of the specific requirements from the tools being developed as part of WP4. Afterwards, in Section 4 the main enablers of seamless system interoperability and data exchange are introduced, along with an assessment of existing data exchange formats and a methodology to select the most appropriate to fulfil the project's requirements. Following this, the actual design of PERFoRMML, PERFoRM's data model and its standard interfaces is described in Section 5, with each of its composing elements being fully detailed. After these descriptions, Section 6 showcases an application of the data model, developed for the WP2/WP3 Integration Workshop at IPB. Finally, Section 7 further describes the interactions of the data model and standard interfaces with the remaining relevant architectural elements from WP2, ending with Section 8 which summarizes some conclusions regarding the contents of this document.

2. Summary of Generic Requirements derived from WP1

The validation of the PERFoRM system will be accomplished in 4 uses cases, covering a wide spectrum of the European industrial force, ranging from home appliances to aerospace and from micro electrical vehicles to large compressor production. Several requirements were collected from the use cases in previous tasks of the PERFoRM project, namely in WP1 [5], D2.1 [6], D7.1 [1], D8.1 [2], D9.1 [3] and D10.1 [4]. These requirements were then clustered to be considered in the specification of the system architecture, namely focusing the flexibility, reconfigurability and general issues (see Table 1).

Table 1 - General requirements: flexibility and reconfigurability overview

General Requirements		**Others Requirements**
Flexibility	**Reconfigurability**	**Necessary to flexibility and reconfigurability**
• Ability to Change Raw material • Ability to Change Processes • Ability to obtain Process interactions • Agility production • To facilitate Mobility, including comparison among different units e.g. OEE, micro-flow-cells) • Cycle time reduction • Cycle , cost reduction	• To obtain feedback from production to design • To obtain Final test feedback to Robot system configuration • To obtain feedback to the process, based on failure control • Cost saving in reconfiguration • To obtain new part reprogramming/setup through CAD critical paths • Self-configuring system, which can define the root-cause based on pattern recognition. • Set-up time reduction	100% Traceability and identification of single products up to the supply chain
		Ability to enable Simulation, Model and prototype in the CPS environment (i.e. process parameters interaction, global factory behaviour, predictive failure)
		Increase the amount of data collected and data availability
		Automatic (semi-automatic) data gathering of machine condition
		Full integration and quick communication among different departments and functions (i.e. scheduling system and maintenance system integration, machine condition and maintenance

		tasks, production and process planning, etc.)

The primary aim of the requirements analysis phase was to identify the needs of the manufacturing industry to progress from the traditional control approaches towards an intelligent and dynamic manufacturing control systems based on plug and produce production systems and self-adjusting devices implementation. An iterative methodology (derived from formal Requirements Engineering (RE)) was followed whilst identifying the requirements and details of this approach has been presented in Deliverable D1.2 [5]. This methodology includes four phases (i.e. Elicitation, Analysis, Specification and Validation). The first iteration was used to identify the requirements, analysing each objective, each context and constraints, whereas the second iteration focused the requirements validation.

The flexibility cluster identifies a set of requirements related to the ability to change production processes in an agile manner, and to adapt the cycle times and their associated costs, namely the ability to change raw material, ability to change processes, ability to obtain process interactions, agility of production, easy mobility, reduction of the cycle time, and reduction of the cycle cost.

The second cluster considers requirements related to the reconfigurability aspect, namely the need to have several feedback loops between the different phases of the production process and a decrease of setup times due to the system reconfiguration. Additionally, there is the need to have the possibility to integrate the robot programming with the technical Computer Aided Design (CAD) drawings, obtain feedback from production to design, obtain final test feedback to the robot system configuration, obtain feedback to the process based on failure control, cost saving in reconfiguration, obtain new part reprogramming/setup through CAD critical paths, self-configuring system, and reduction of the set-up time.

The *General Requirements* include only the requirements regarding flexibility and re-configurability that are required across all four use cases, whilst the *Other Requirements* column lists requirements which are specific to certain use cases and which lead to obtain those aspects of flexibility and reconfigurability grouped in the *General Requirements*.

It is observable that product and production traceability is mandatory, as well as the automatic gathering of data and the use of simulation tools in the process chain. Furthermore, the integration of systems from different company's departments is of major concern. To accomplish the requirements, the architecture proposed in Deliverable D2.2 [7] identifies the need of develop a common data model and generic standard interfaces. In fact, and in line with the general vision for the Industrie 4.0 platform, one of the key challenges to tackle the requirements addressing flexibility and reconfigurability is the aspect of interoperability in real industrial environments, dealing with the representation and seamless exchange of data originating from a wide array of entities, often from very different, albeit related, actions levels.

The interconnection of heterogeneous legacy HW devices (e.g. robots and the respective controllers) and SW applications, such as databases, SCADA applications and other management, analytics and logistics tools, is one of the main goals currently being pursued in this vision.

To this effect, the PERFoRM architecture employs the adoption of standard interfaces as the main drivers for pluggability and interoperability, aiming at enabling the connection between such devices and applications in a seamless and transparent manner. These interfaces should provide the devices, tools and applications with the means to fully expose and describe their services in a unique, standardized and transparent way to enhance the seamless interoperability and pluggability, fully specifying the semantics and data flow involved in terms of inputs and outputs required to interact with these elements.

Therefore, these interfaces should provide a set of functionalities related to a standardized service invocation, i.e., the definition of the list of services to be implemented by the interface, the contract implementation of each service (i.e., the name, input parameters and output parameters), and the definition of the data model handled by the services. Note that an important requirement for the design of the standard interfaces is the usage of service-orientation to expose the device/system functionalities as services.

For this purpose, a common data model is also adopted, serving as the data exchange format shared between the PERFoRM-compliant architectural elements. This data model need to cover the semantic needs associated to each entity, which in the particular case of industrial automation, means that the requirements related to each of the ISA-95 layers and their respective needs are considered.

From the above described requirements it is foreseen that the development of such standard interfaces is not only mandatory and crucial but also needs to cover a wide spectrum of data needs. In fact, the standard interfaces need to cover the modelling of data coming from sensorial data up-to data generated in ERP systems.

The specification of the data model composed by the standard interfaces will be performed in the next chapters of this document. These chapters will further detail the data needs in each of the use cases and of the tools being applied to address the use case needs. Combining these two views, the complete data is to be derived.

3. Analysis of Specific Requirements

Besides the requirements derived from WP1 (see Section 2) and the specific necessities identified in each of the use cases' respective deliverables the design of the PERFoRM common data model needs to take into consideration the specific requirements from the tools being developed or applied as part of WP4, thus making sure that the information that is necessary to be exchanged between the different system actors can be modelled and represented in a common, interchangeable format. These requirements will be analysed in Subsection 3.1.

3.1. WP4 Tools

The main goal of this chapter is to give a preliminary specification for the tools and other solutions, which are to be developed or applied in WP4, concentrating on the interface connectivity layer, its requirements and technical specifications. In general this chapter aims to describe the following items:

1. A short overview of tools, which are planned to be developed in WP4 and how they are grouped together, i.e. the main purpose of a tool's development and its application area in PERFoRM;
2. A list of functions and services which are going to be provided by the tools and how these can be accessed via an interface;
3. A list of specific inputs and outputs of the connected systems, especially with regard to the middleware and its connectivity layer;
4. A description of technological solutions or protocols will be used or required to establish the communication between a tool and the middleware to transport required messages.

3.1.1. Solution Matrix and Tool Clustering

The key objectives of the WP4 are harmonization, development and prototyping of the simulation and visualization environment, planning tools and decision rules, which can be applicable in a flexible production systems and support the given use case.

To clarify the necessary solutions and approaches across the four use cases a solution matrix was prepared in WP4. This matrix is based on the worked out documentation delivered in WP1 and which contains specified requirements and the technology analysis of existing solutions and their tools. At this point it is important to mention, that the majority of the existing tools and technologies are brought into PERFoRM project exclusively by consortium members or were developed by them in other European projects. This means that PERFoRM solutions, which are to be developed or adopted in WP4, establish a very good cross-linkage to other European projects, have a good standardisation background and, finally, are based on prove-of-concept ideas.

Tool clustering is a special step during which the collected tools were divided into three main groups taking into account their main criteria: simulation cluster, planning cluster and decision cluster (see Table 2).

Table 2: Tool and methodology clustering.

Simulation Cluster	Planning Logic Cluster	Decision Support Cluster
/SC-001/ Simulation Environment (SE)	**/PLC-011/** Dynamic scheduling of production orders and maintenance tasks	**/DSC-021/** Support for KPIs and energy consumption correlation
The simulation Environment encompasses: • **/SC -002/** Predictive Maintenance Simulation • **/SC-003/** Agent-based Simulation (RMS[1] and AMS[2] in discrete event environment) • **/SC-004/** Boundaries specification for simulation method optimization	**/PLC-012/** Multi-objective planning & scheduling for dynamic, flexible manufacturing systems	**/DSC-022/** KPI monitoring with what-if-game functionality
	/PLC-013/ Reconfiguration management in flexible manufacturing systems	**/DSC-023/** Min-Max Data Mining Toolbox
	/PLC-014/ Agent-based Reconfiguration Tool	**/DLC-024/** Data Mining
	/PLC-015/ Energy based planning with rescheduling	**/DLC-025/** Automatic Monitoring and visualization of KPIs
		/DLC-026/ Bayesian Diagnostics & Prognostics for Manufacturing Equipment
		/DLC-027/ Universal web based KPI visualization
		/DLC-028/ Methodology and algorithm defined for KPIs identification

The following abstracts give a short description of the tools or methodology and their application area with regard to the PERFoRM use cases.

[1] Reconfigurable Manufacturing Systems (RMS)
[2] Automated Manufacturing Systems (AMS)

Simulation Cluster

The *Simulation cluster* is the first group. It contains several tools and approaches, which aim to develop simulations and simulation techniques or support these with a simulation environment, which comprises a modular simulation structure and enables flexible tool concepts.

Agent-based Simulation (RMS and AMS in discrete event environment): This tool will be embedded into the Simulation Environment (see below) and is to consider the reconfigurable manufacturing systems and automated manufacturing systems in discrete event environment to support optimal scheduling and avoid boundary deadlocks or breakdowns.

Predictive Maintenance Simulation: Based on data and information from machines and devices together with the current production and maintenance schedule, this simulation considers the effects of foreseeable machine breakdowns to the remaining production schedule. In particular, the simulation tool calculates final KPI´s for a list of possible manufacturing schedules. Based on the final KPI´s it will be possible to make a ranking of the manufacturing schedules. Because of a generic structure, the simulation is applicable for every use if the required input data are available in the right data model.

Simulation Environment (SE): This task aims at creating an effective and efficient working environment in which factory-level simulations can be performed. In detail, the SE tool is used, on the one hand, to support simulation of factory and production, given the current status of all installed machinery and all planned product orders within the simulation horizon, and on the other hand, to support the optimization of certain aspects of a factory. Thus, this tool is going to support a number of such important requirements as e.g. scheduling and verification. This solution integrates three following tasks:

a) *Workflow for SE Execution;*
b) *Interfaces from Middleware to SE;*
c) *Integrating control logic configuration into SE.*

Boundaries specification for simulation method optimization: This approach provides analytical decisions and special recommendations, which will help to identify the boundary conditions and essential parameters in both effectiveness and efficiency of simulation cluster with regard to the four use cases. According to this approach, the main steps will include such actions as identification of relevant KPIs; calculation of decision variables; analysing topology requirements and specifying possible reconfiguration. The approach will specify the type of simulation with regard to each use case, give the overview of its main outputs and core elements, provide information to workflow data and interface specific settings (e.g. needed and available data, how is data accesses and in which format) . The final results will provide a possibility to identify a “simple story”, which can describe an overall simulation activity across the four use cases.

Planning Logic Cluster

The second group is the *Planning Logic Cluster*. It contains tools, which couple the planning and the scheduling layers of the production system and are responsible to provide functionalities to guarantee the real time and on-demand reactions to unplanned disturbances. The main goals of this cluster are focused on the deployment of agent-based mechanisms into the PERFoRM system, provision of basic ground for multi-objective scheduling tools for production orders and maintenance tasks and to support the reconfigurability principles.

Dynamic scheduling of production orders and maintenance tasks: The dynamic scheduling tool aims to provide a solution to the workshop throughout the organisation of maintenance tasks and production tasks in the same scheduling environment. Within the Siemens use case, this tool is used to organise all tasks dynamically according to the production needs, based on the information received from the middleware. This tool supports the requirements of flexibility and dynamism within the production environment. Its main goal is to reduce the time to finish all tasks complying with the current demand of products and maintenance, introduce the maintenance task between the production tasks to present a clearer idea to the production managers and operators about the availability of a given machine.

Multi-objective planning & scheduling for dynamic, flexible manufacturing systems: The multi-objective planning and scheduling tool aims to provide a solution to the workshop by organising the process sequence of all operations. With regard to the GKN use case this tool is used for the logical scheduling for the dynamic workshop, i.e. it provides multi-objective planning and scheduling techniques and allows to access data from the middleware. It supports the following requirements as e.g. a flexible, dynamic and production workshop concept; reduced time to finish all the production according to current demands; reduce energy consumption to finish all the production according to current demands.

Reconfiguration management in flexible manufacturing systems: The reconfiguration management tool aims to provide a solution to the workshop and cell level by organise the process sequence of all operations.

Agent-based Reconfiguration Tool: The agent-based reconfiguration tool aims to provide an automatic process re-organization by automatic recognition of the current processes and establishing the appropriate actions for the robotic cell coordination and operation. This tool will be used for the logical re-organisation of the *uFlow* cell concept (see WP10) applying the agent-based reconfiguration techniques and accessing data from the OPC UA Servers, supporting the following requirements: a flexible, reconfigurable and scalable production cell concept*;* reduction of time for changeovers and adaptation to current demands*;* increasing the level of automation for implemented processes*;* improvement reconfigurability.

Energy based planning with rescheduling: This tool is based on the existing solution called XETIXS LEAN provided by XETICS. Its main goals are, firstly, the integration of energy saving measures in accordance with DIN EN ISO 5001, secondly, the collection and visualization of energy consumption in detail and, lastly, the identification of the "energy consumption" and the process-induced peaks in consumption referring to the products, raw materials used, machines and tools. Furthermore, this tool helps to introduce specific measures to improve energy efficiency and develop new planning strategies. In detail these are management of energy meters and hierarchical structures; collection, visualization and monitoring of energy consumption; automatic targeting system for infringing target values; analysis of consumption in correlation with other production parameters; consumer Profiles to Identify Power Peaks; energy KPIs and Planning Strategies for improving energy consumptions.

Decision Support Cluster:

Support for KPIs and energy consumption correlation: This tool delivers a decision-making approach, which is going to support the KPIs and energy consumption correlations. The first steps of the tool analysis include collection and aggregation of typical machine status (e.g. idle, set up or break down). In a second step a matching of the collected variables of time and applied effort of a machine can be done. This will help to understand how many effort one particular machine has spent and how this effort can be reduced to optimize the production.

KPI monitoring with what-if-game functionality: This is a web-based visualization tool to support decision-making strategies, by KPI monitoring and by performing what-if-game based on the variation of several KBFs. The main features of this tool are the visualization and monitoring of KPIs; detection of trends and deviations and, finally, *What-if-Game* capability. The monitoring and calculation of KPIs is done by using raw data from the system, within the scope of the use case requirement, from a SQL-database. This tool supports such requirements as e.g. mobility and improvement of production optimization.

Min-Max Data Mining Toolbox: This tool is used to calculate, on the one hand, the maintenance suggestions from electricity measurements and, on the other hand, to determine the changes in machine behavior or state. Thus, using this tool the following two requirements, i.e. scheduling and accounting, will be taken into consideration.

Data Mining: The data mining techniques belongs to the group of tools concerning the automatic monitoring and visualization of KPIs and will be used to predict the failure probability. Apart from the failure prediction mechanisms the visualization of current status of the machines is also going to be realized within this tool.

Automatic Monitoring and visualization of KPIs: This tool is another feature provides by the XETICS LEAN solution. It offers automatic data collection on production and packaging lines. Thus, the tool captures short interrupt events, freeing line operators from manual data collection tasks, and provides immediate feedback on unexpected or trend conditions. Clear indications of line bottlenecks, visibility into real-time performance metrics, and drill down analysis tools enable stakeholders to have useful information to effectively increase line performance and operational efficiency. Furthermore, the tool will deliver real-time monitoring of production lines and combines with tracking of equipment utilization for a full line performance solution.

Tracking work order execution provides an adequate categorization of unplanned downtime versus product change or downtime or other scheduled events.

Bayesian Diagnostics & Prognostics for Manufacturing Equipment: This tool aims at calculating the probability of damage of a machine (minimal solution) or its components (targeted solution) within a specific time range (or multiple time ranges). It will be to predict machine failure based on three possible data: Failure logs (failure notifications); operators' maintenance logs and Sensory Data (e.g. vibrations, electrical consumption). This tool will support such requirements as e.g. flexibility and track & trace.

Universal web based KPI visualization: This tool is used for the visualization of maintenance suggestions from electricity measurements and calculated changes in machine behaviour or state, thus, supporting the requirements for scheduling and accounting.

Methodology and algorithm defined for KPIs identification: This tool provides a methodology and specific algorithm, which will support the previously described decision-making approach (see *Support for KPIs and energy consumption correlation*). Its main goal is evaluation and visualization of the identified KPIs and delivery of the "best practise" advices.

3.1.2. Interface Protocols and Communication Channels

In order to understand how the interfaces can be configured in the PERFoRM project and what communication channels are essential esp. to the middleware, we collected information about the application technologies and their tools for each solution (seeTable 3: Application Technologies of Developed Tools).

Table 3: Application Technologies of Developed Tools

Simulation Cluster	Planning Logic Cluster	Decision Support Cluster
/SC-001/ will be integrated in the SE solution (see SE).	/PLC-011/ Dynamic scheduling of production orders and maintenance tasks using algorithms implemented in Java.	*/DLC-022/* will be a JSP (JavaServer Pages) web application.
/SC-002/ uses the native interfaces of AnyLogic and REST protocols.	*/PLC-012/* will be written in Java, using the JMetal (Java Meta-heuristic Framework) solution based on the native REST API implementation	*/DLC-023/* is homemade software based on NodeJS implementation. REST will be used as an exchange technology.
/SC-003/ uses C++/C#/java executable that wraps a simulation tool (Siemens PlantSimulation, AnyLogic, others) and REST as a standard interface with no external adaptors.	*/PLC-013/* will be also written in Java, using the JMetal (Java Meta-heuristic Framework) solution based on the native REST API implementation	*/DLC-024/* is going to use R for data cleaning and predictive model generation.

	/PLC-014/ will be a peer-to-peer agent based application, written in Java and using the JADE (Java Agent Development Framework) solution.	*/DLC-025/* uses native REST API.
	/PLC-015/ uses native REST API.	*/DLC-026/* is applied as python packages.
		/DLC-027/ is homemade software based on NodeJS implementation. REST and MQTT will be used as an exchange technology.

The PERFoRM middleware interface exchange infrastructure offers the RESTful (or REST)[3] standard interface, which follows the Resource Oriented Architecture (ROA) principles. The PERFoRM interface infrastructure itself follows the generic principles and can be expanded with other interface components if required. REST belongs to well-defined interfaces for distributed functionalities, which use specific protocols and communication channels independent of the platform settings, operating system or a programming language. RESTful offers a perfect solution for the implementations with greater flexibility and lover overhead and can enrich the PERFoRM system.

REST decouples clients and servers and keeps the interface structure clear to avoid extensions and overloads (see Figure 1). Real encapsulation of clients and servers allow independent development, flexible deployment and scalability on both sides. The common REST APIs are based on the Hypertext Transfer Protocol (HTTP) request methods. HTTP is "an application-level protocol for distributed, collaborative, hypermedia information systems […]", which allows negotiating systems "[…] to be built independently of the data being transferred"[4].

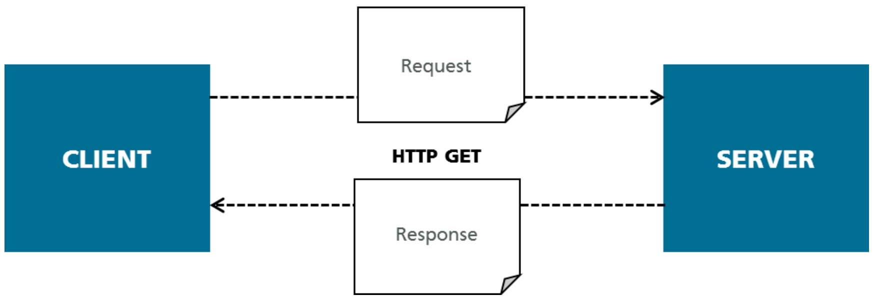


Figure 1: REST client-server architecture

The message transfer is based on simple exchange calls as e.g. POST, GET, PUT, etc.. For example, the method PUT is used to change or update the state of recourse, which can be an

[3] Representational State Transfer (REST)
[4] http://www.rfc-base.org/rfc-2616.html

object, file or a text block. The command GET is used to retrieve a resource message. Contrary to GET method, POST sends the message to the recipient.

A common REST API encrypts a transaction message to create small series. These series take control of each specific operation while transaction process. REST uses two possible data formats: XML and JSON. JSON format derives from the JavaScript standards and is used for simple as well as complex message transfer.

3.1.3. Description of the Tools' APIs

This subchapter gives an overview of the functions and messaging structure of the developed tools in WP4. This chapter describes only a preliminary version because the tools are still in development process and can contain some changes or adaptions to each use case till the end of M20. Nevertheless, this information is being already used as a backbone for the development of the standard interfaces and the results are being utilized in T2.4 for the development and specification of the interface infrastructure of the industrial manufacturing middleware.

The communication channels between the middleware solution and the collaborative tools or *applications* (see 3.1.1) can be described as it is shown below in the Figure 2. On the one side, there is a "*sender*" tool; on the other hand, there is a "*receiver*" tool. These are able to do two main operations, i.e. send messages (*outbound operation*) and receive messages (*inbound operation*) and contain their own application programming interface (API). The API provides a special gateway for an application to get an access to the middleware and establish a secure communication session.

The intermediate communication level is governed by the *middleware* solution. The middleware fulfils such important actions as routing of the messages, their overall management (as for example queueing operations) and, of course, resolving of the interface protocols and their management.

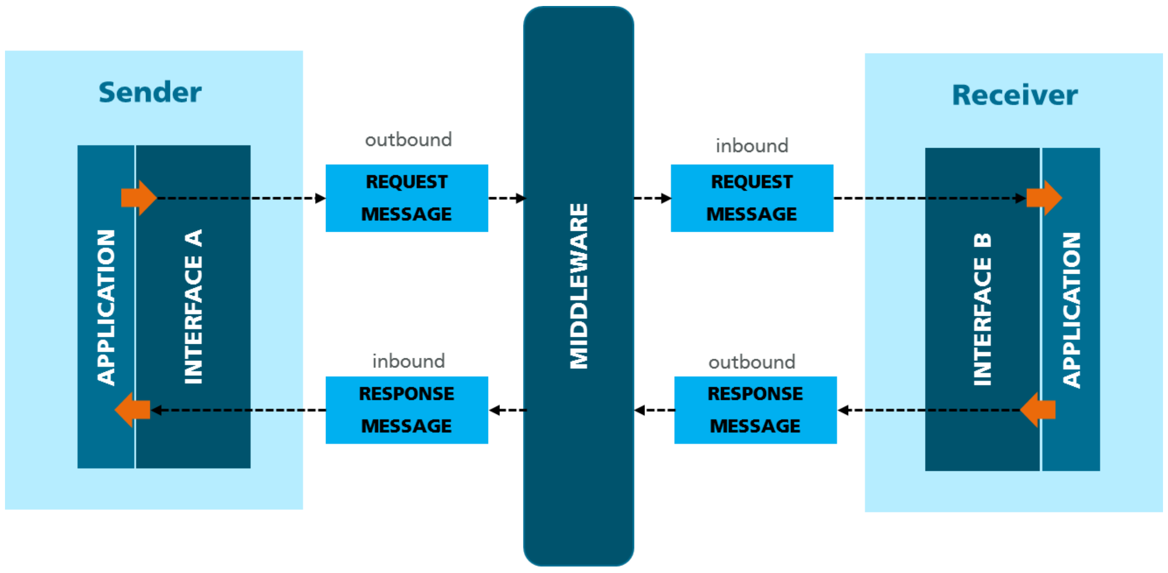


Figure 2: Separation of Outbound and Inbound Interfaces

An API can be presented in many forms. The most usual is to describe it as a library, i.e. give a list of methods with their return functions. We followed this recommendation and analysed the tools APIs according to the six main fields. Thus, the first two fields in our description contain a "sender" and a "receiver" type. Function name and its short description follow in the next two fields. The last fields describe, firstly, what kind of message is sent and, secondly, what message is expected as a callback.

The APIs have the aim to support the development of the PERFoRM applications in WP4 as each application requires an interface as a gateway to establish the connection to the middleware. The actual design of the application interfaces depends on the requirements, which are set for each use case and which are set by the middleware interface structure.

The transformation of the messages and their format also plays an essential role and must be configured beforehand to avoid unpredicted failures in the communication between a sender and a receiver. For example, if the simulation or visualization tool is going to use the information of the machines in process, the IDs of these entities, their status, current time and other important information should be transferred between the tools using the specified format.

During the development of the middleware and, of course during the actual implementation of the tools themselves, one should deeply understand, what functionalities the application APIs offer or request. For this reason we collected the APIs' functions and message descriptions as it is shown in Appendix I. Generally, it is a set of defined methods or rules of communication between different software components which are e.g. normal software applications or software interfaces and hardware devices.

4. Core elements for Seamless Interoperability

An important key issue to ensure the interoperability in real industrial environments, interconnecting heterogeneous legacy hardware devices, e.g., robots and Programmable Logic Controllers (PLCs), and software applications, e.g., Supervisory Control and Data Acquisition (SCADA), MES and databases, is the adoption of standard interfaces.

These aim to define the bridge between devices and applications in a unique, standard and transparent manner, ensuring the transparent pluggability of these heterogeneous devices. For this purpose, a standard data representation should be adopted by the interface that should also define the list of services provided by it, and the semantics data model handled by each service.

In this definition, and particularly for industrial automation, several ISA 95 layers addressing different data scope and requirements should be considered, namely the machinery level covering mainly L1 (automation control) and L2 (supervisory control) layers, and the backbone level covering the L3 (manufacturing operations management) and L4 (business planning and logistics) layers.

Additionally, the achievement of complete interoperability and pluggability requires to complement the use of standard interfaces with adapters to transform the legacy data representation into the native standard interface data representation.

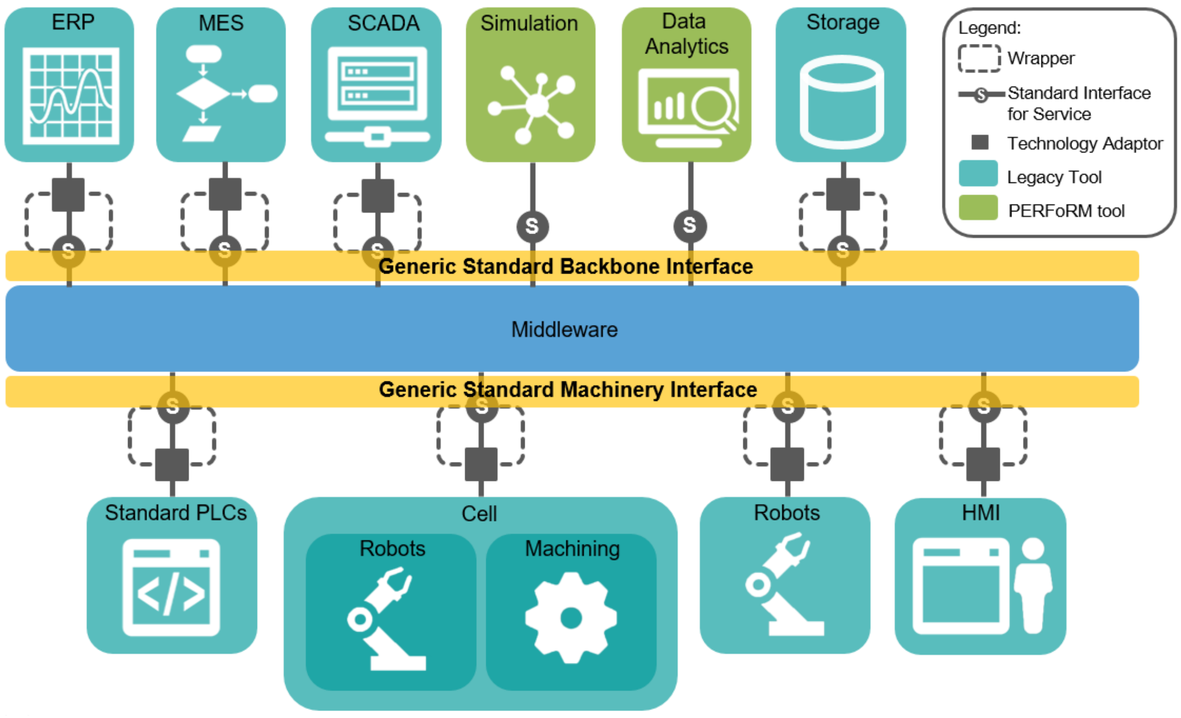


Figure 3 - Overview of Task 2.3 Elements

4.1. Solutions for Data Representation and Modelling

One of the main challenges presented for Industry 4.0 is the representation and seamless exchange of data originating from heterogeneous elements, often from very different, albeit related, action levels. A clear example using the ANSI/ISA-95 standard [8] terminology would be the harmonization of data pertaining to the Enterprise Resource Planning (ERP), to the Manufacturing Execution System's (MES) layer and to Supervisory Control And Data Acquisition (SCADA) systems.

Coincidentally, the subject of standardization is consistently indicated by the industry as one of the major obstacles for the industrial acceptance of disruptive technologies [9]. In fact, several European funded projects have already made some efforts to push towards this goal. Some examples include the BatMAS [10] and FP7 PRIME [11] projects. The latter highlights the importance of a common semantic language and data representation in order for proper interoperability and pluggability to be achieved, due to the plethora of heterogeneous entities involved in these intelligent, complex manufacturing systems [12].

In fact, over the last few years several industrial standards have emerged, each providing a set of semantic definitions for data modelling and exchange across different areas of the manufacturing industry. An example is the IEC 62264 standard [13], which is based on the aforementioned ANSI/ISA-95. IEC 62264 entails a framework aimed at facilitating the interoperability and integration of both enterprise and control systems.

Other existing standards include IEC 61512 [14], based on ANSI/ISA-88 and focusing the batch process domain, ISO 15926 [15] aimed at process plants, IEC 62424 [16] for the exchange of data between process control and P&ID tools and IEC 62714 [17], centred on industrial automation systems engineering data.

As a direct consequence of this emergence, some mostly XML-based implementations of the specifications defined in these standards have been developed. The list presented below has been selected from the pool of technologies currently available and documented in the literature which stand out as potentially fulfilling some or most of PERFoRM's semantic needs.

- **IEC 61512 BatchML (*T_1*)** - An XML implementation of the ANSI/ISA-88 Batch Control family of standards. It offers a variety of XML schemas written in XML Schema Language (XSD) that implement the ISA-88 specifications.
- **IEC 62264 B2MML (*T_2*)** - Implements the ANSI/ISA-95 family of standards for Enterprise-Control system integration via XML schemas written in XSD. The latest versions of BatchML's schemas were integrated into the B2MML namespace, now using the B2MML common and extension files. Despite this fact, for the purpose of this study both implementations will still be considered separately.
- **ISO 15926 XMplant (*T_3*)** - Provides access to process plant information in a neutral form, following the ISO 15926 specifications, supporting structure, attributes and geometry of schematics and 3D models.
- **IEC 62424 CAEX (*T_4*)** - Computer Aided Engineering Exchange (CAEX) [18] is an object-oriented, neutral, XML-based data format that allows the description of object information, such as the hierarchical structure of a plant or series of

components. Its scope spans across a wide variety of static object types, such as plant, document and product topologies as well as petri nets.

- **IEC 62714 AutomationML (*T_5*)** - AutomationML is an XML-based data format that builds upon other well established, open standards spanning several engineering areas, aiming at interconnecting them. More specifically, CAEX serves as the basis of hierarchical plant structures, while COLLADA and PLCopen XML are the foundations for geometry/kinematics and control applications, respectively [19].
- **OPC UA's Data Model (*T_6*)** - OPC UA defines a very generic object Data Model (DM) supporting relationships between objects (references) and multiple inheritance. It is used by OPC UA to represent different types of device data, including metadata and semantics.
- **MTConnect (*T_7*)** - MTConnect is a manufacturing standard [20] presenting an XML-based format for data exchange between the shop-floor and software applications for monitoring and analysis. This includes device data, identity, topology and design characteristics such as axis length, speeds and thresholds. It also possesses a set of specifications to ensure interoperability with OPA UA.

4.2. Selection Criteria and Matching

Due to the large amount of available solutions, a pre-selection process is required in order to thin the amount of possibilities for the implementation phase. For this reason, during the literature review process the following selection criteria were identified with the specific goals of the PERFoRM project in mind. As such, each of them relates to a given specific area of focus targeted in the project. A general description of each of these criteria is presented below.

- **Process domain-specific concepts (*C_1*)** - Covers the aspects associated to specific methods of production, including for instance batch, flow and job production.
- **Performance analysis (*C_2*)** - Entails information that enables the assessment of production performance, including start time, end time, location or status such as the percentage of completion.
- **Quality monitoring (*C_3*)** - Concepts enabling the monitoring of production quality, ensuring that products consistently meet the expected quality requirements. As an example, this can include reject and scrap tracking structures, inspection data, and quality tests.
- **Material resource management (*C_4*)**- Relates to the existence of specifications for material classes, material lots or sub-lots and even QA (Quality Assurance) tests that may be exchanged between business systems and manufacturing operations systems.
- **Production planning and scheduling (*C_5*)** - This criterion relates to the capacity to describe information to be exchanged and used by for instance (using ISA-95

terminology) ERP systems and MES, detailing production goals and schedules to achieve said production targets.

- **Recipe management (*C_6*)** - Inclusion for instance of master recipes, recipe formulas or recipe ingredients.
- **Product description (*C_7*)** - This relates to the capacity to describe information associated to a product, such as production rules, assembly instructions, bill of materials and bill of resources.
- **Maintenance (*C_8*)** - Maintenance descriptions should detail information regarding maintenance operations, such as requests, responses and work orders. Relevant associated information should also be present, which can include dates, times, personnel involved, descriptions, status and technical information, among others.
- **Failure and alarm management (*C_9*)** - Deals with information structures that enable the handling and management of failures and alarms, such as categories, definitions, priorities, timestamps and hierarchies.
- **Engineering life-cycle data (*C_{10}*)** - Information pertaining specifically to the engineering life-cycle domain, namely system design or simulation (e.g. CAD models).
- **Supply-chain data (*C_{11}*)** - This criterion encompasses information related to the supply chain. A few examples include shipment data, orders, distributor information and transactions.
- **Extendibility (*C_{12}*)** - Possibility to extend and add further information *a-posteriori*.
- **Process control (*C_{13}*)** - This relates to process control at the PLC-Level, including pertinent data such as signals, I/O and control sequences.

Through an analysis of current literature, as well as of each technologies' own documentation, it is possible to relate each of them to the aforementioned criteria. The result from this process is summarized in Table 4. Blank spaces indicate that either a given criterion is not covered, or that no reference to it was found in neither the respective technology's documentation nor in the literature. Criteria marked with a check sign are fully addressed, while those marked with *"-/X"* are either partially or not directly covered.

Table 4 - Analysed standards and the associated differentiation criteria (adapted from [11])

	T_1	*T_2*	*T_3*	*T_4*	*T_5*	*T_6*	*T_7*
Process domain specific concepts	✓	-	✓	✓	✓	-	-
Performance analysis	-	✓	-	-	-	-	-
Quality monitoring	-	✓	-	-	-	-	-

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Material resource management	-	✓	-	-	-	-	-
Production planning and scheduling	-	✓	-	-	-	-	-
Recipe management	✓	-	-	-	-	-	-
Product Description	-	✓	-	-/X	-/X		-
Maintenance	-	✓	✓	-	-/X	-	-/X
Failure and alarm management	-	✓	-	-	-	-/X	-/X
Engineering life-cycle data	-	-	✓	-	✓	-	-
Supply-chain data	-	-/X	-	-	-	-	-
Extendibility	✓	✓	-	✓	✓	✓	✓
Process control (PLC-Level)	-/X	-	-	-	✓	✓	✓

A clear conclusion that can be drawn from the analysis of Table 4 is the fact that no single standard covers the entire spectrum of relevant criteria to match PERFoRM's needs. As a consequence, a possible solution could be derived from the combination of two or more of these technologies, hence the need for a proper selection methodology to be developed.

4.3. Selection Methodology

The selection of the adequate technology to perform a specific task is, most of the time, a complex and subjective process. Its complexity is mostly related with the product's characteristics, and how they correlate to the consumer's wishes. For each customer, the product's number of features and their importance are the key analysed elements. Hence, for a technology assessment, this is not a simple process either. There are several factors that must be analysed and should be taken in consideration for every step of the decision process. Therefore, the decision process is defined by the following five steps.

- **Step 1** - *Criteria definition and description*: The first step of the presented methodology is the criteria definition and description, necessary to evaluate each technology. Each criteria, C_i, where $i \in \mathbb{N}$, must be provided by the literature review and should represent the end users' wishes. Each criteria evaluated by only two values, "0" or "1", which are translated into the existence or non-existence of each specific feature.

- **Step 2** - *Relevance definition*: In this second step, the objective is to define the level of importance of each criteria for the end user. This factor is defined by W_i, where $i \in \mathbb{N}$. For each criteria the weight must be defined by a scale, from 1 to 10. In this step the end users are asked to, through a questionnaire, provide the importance of each criteria to be present in the final product.

  After the definition of these two decision factors, criteria and relevance, it is necessary to evaluate the technology.

- **Step 3** - *Technology assessment*: The third is the technology assessment process, defined by $T_{Score\ k}$, where $k \in \mathbb{N}$. In this process the two evaluation factors are combined to create a score. Each technology is evaluated in accordance with the importance, W_i, that each end user gives to each criteria C_i. To proceed to the technology evaluation, eq.1 is applied.

$$T_{Score_k} = \frac{C_1W_1 + \ldots + C_nW_n}{W} = \frac{1}{W}\sum_{i=1}^{n} C_iW_i \qquad (1)$$

  Where W is determined by eq. 2:

$$W = \sum_{i=1}^{n} max(W_i) \qquad (2)$$

  And, where $n \in \mathbb{N}$**.** The results from the usage of eq. 1 correspond to the evaluation of one technology by one end user. So, in order to have a global validation of the technologies, by all end users, it is necessary to aggregate each of their opinions.

- **Step 4** - *Data aggregation process*: The goal of the fourth is to define a methodology to aggregate the end users opinions regarding each technology. To do so, two new factors are determined. Firstly, the average $\overline{X}_{\text{T score k}}$, which is defined by eq. 3.

$$\overline{\text{X}}_{T_{score_k}} = \frac{\sum_{k=1}^{n} T_{score_k}}{n} \qquad (3)$$

  Where $n \in \mathbb{N}$. This factor (average) determines the point in which the opinions are centred. Secondly, the standard deviation $\text{S}_{\text{T score k}}$ which is defined by eq. 4.

$$S_{T_{score_k}} = \sqrt{S^2_{T_{score_k}}} = \sqrt{\frac{\sum_{k=1}^{n}(T_{score_k} - \overline{X}_{T_{score_k}})^2}{n-1}} \quad (4)$$

Where $n \in \mathbb{N}$. The standard deviation defines the level of agreement, by the end users, in the evaluation process.

- **Step 5** - *Ranking of the assessed technologies*: The final step is to rank the technologies based on the combined opinions from the end users. So, in order to combine them, a fuzzy inference system (FIS) is suggested. The FIS combines both presented factors, $\overline{X}_{T\ score\ k}$ and $S_{T\ score\ k}$, in order to define a score for each technology (Figure 4).

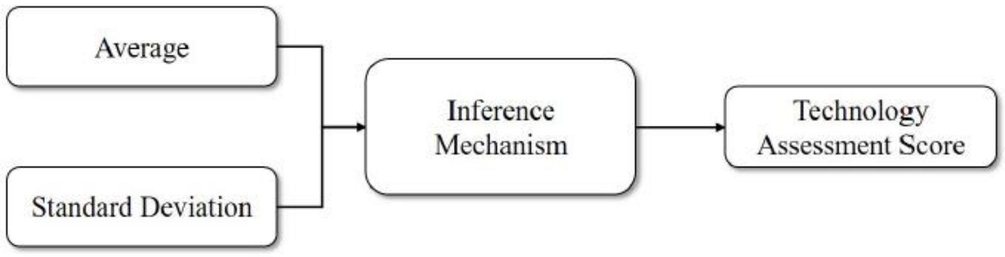


Figure 4 - Consensus-based model (Adapted from [21]**)**

Based on this score, all the technologies are ranked. The technology which presents the higher score is considered the most adequate for the purpose of the PERFoRM project.

4.4. Technology Assessment

During the development of this work, some presumptions were assumed in order to apply the designed methodology.

1) All the criteria marked with "-/X" will be considered as non-existent, since they cannot fulfil the required purpose in its entirety;
2) The model considers that all criteria are self-contained.

For the technology assessment, the methodology was applied as follows.

- **Step 1** - *Criteria definition and description*: During the development of the present study all the criteria were defined in accordance with the literature review, and with the features defined in section 4.1. Each of the 13 defined features is relevant for the technology assessment developed under the scope of PERFoRM project.
- **Step 2:** *Relevance definition*: To determine the relevance of each criteria for the end users (spanning across different industrial areas), each was asked to answer a

small questionnaire. This questionnaire aimed to establish, from "1" to "10", the importance of each criteria in the decision process. The end users are defined as E_m, where m ∈ {1, 2, 3, 4}. The evaluation of each end user is presented in Table 5.

Table 5 - Importance of the criteria for the end users

	E_1	E_2	E_3	E_4		E_1	E_2	E_3	E_4
W_1	7	10	7	3	W_8	9	10	2	10
W_2	9	10	10	8	W_9	10	7	3	10
W_3	8	10	8	10	W_{10}	3	10	5	10
W_4	6	10	3	5	W_{11}	3	10	7	10
W_5	10	10	1	8	W_{12}	8	10	2	10
W_6	3	10	2	5	W_{13}	6	10	2	10
W_7	6	10	4	3					

After the collection of the presented data, step 3 could then be applied.

- **Step 3** - *Technology assessment*: In this step the technology was evaluated based on the application of eq.1. Table 6 summarizes the opinions of the respective end user for the evaluation of each technology.

Table 6 - Technology assessment (per end user)

	E_1	E_2	E_3	E_4		E_1	E_2	E_3	E_4
T_1	0.14	0.23	0.08	0.14	T_5	0.18	0.31	0.12	0.25
T_2	0.51	0.59	0.25	0.49	T_6	0.11	0.15	0.03	0.15
T_3	0.15	0.23	0.11	0.18	T_7	0.11	0.15	0.03	0.15
T_4	0.12	0.15	0.07	0.10					

Based on this information, it is necessary to aggregate the data.

- **Step 4** - *Data aggregation process*: For the data aggregation, eq.3 and eq.4 were be applied. The results are presented in Table 7.

Table 7 - Data Aggregation

	$\overline{X}_{T_{score_k}}$	$S_{T_{score_k}}$		$\overline{X}_{T_{score_k}}$	$S_{T_{score_k}}$
T_1	0.15	0.06	T_5	0.22	0.08
T_2	0.46	0.15	T_6	0.11	0.06
T_3	0.17	0.05	T_7	0.11	0.06
T_4	0.11	0.05			

- **Step 5** - *Ranking of the assessed technologies*: The last step is to apply the consensus based model, although it is necessary to validate said model through three tests. The first of which being the extreme conditions test Figure 5.

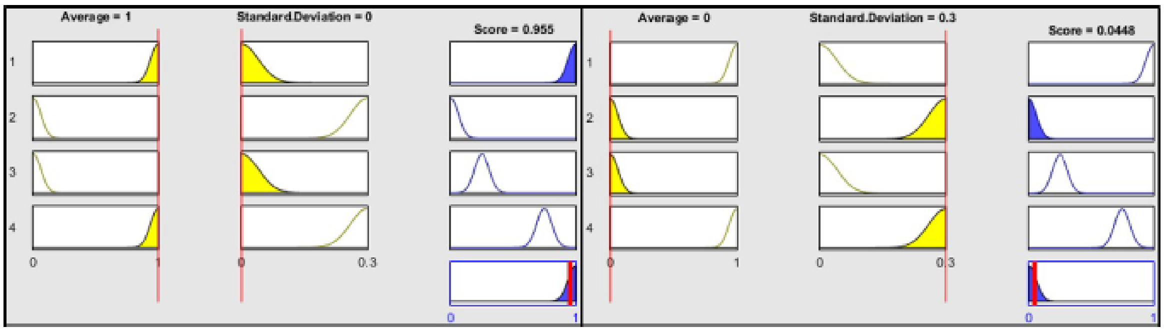


Figure 5 - Extreme Conditions Test

In this test, the model is forced into the most extreme conditions analyse the results coherence. The tests are in accordance with the expected values, near to maximum and minimum, respectively. So, if the average is high and the standard deviation is low, the score is high, and on the contrary, if the average is low and the standard deviation is high, is it expected for the score's result to be low, as can be seen in Figure 5. The following test is the face validity test (Figure 6).

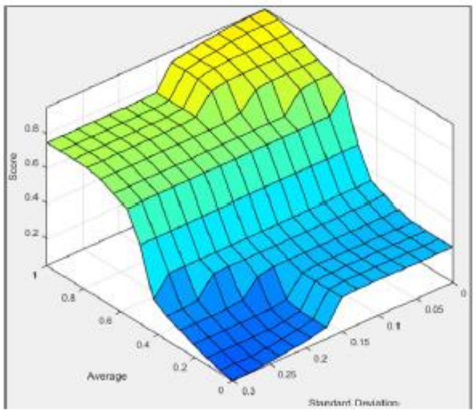


Figure 6 - Face Validity Test

In this test some irregularities can be analysed and corrected, if they exist. If any irregularity is spotted the model should be corrected in order to present an upward/downward trend. As it can be seen in Figure 6, the surface presents an upward tendency, which indicates a well-defined model. The final test is the behavioural test (Figure 7).

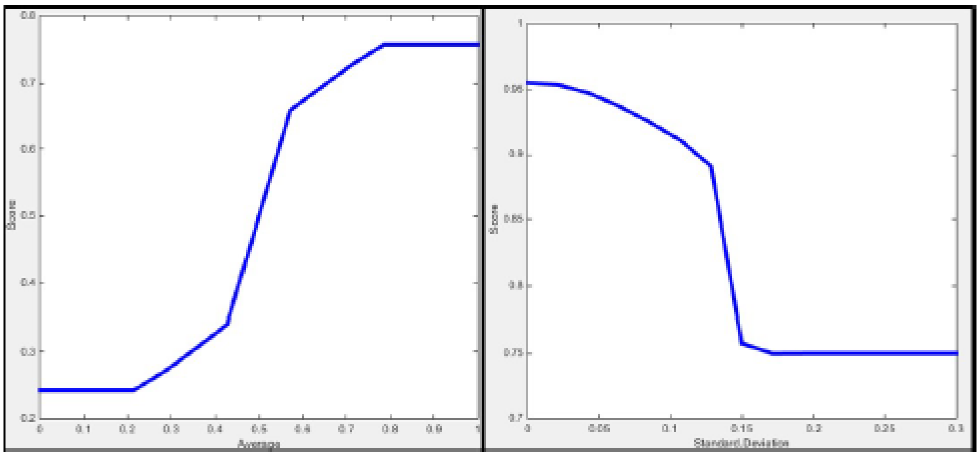


Figure 7 - Behavioural Test

This test, along with the face validity test, indicates the behaviour of the model, and based on it, it is possible to stablish its adequacy. For this specific case, it is possible to identify the upward trend, from the average perspective, and the downward tendency, for the standard deviation point of view.

Thus, once the model is validated, it can be used to analyse the data from Table 7. The results and the ranking are presented in Table 8.

Table 8 Score generated by the consensus-based model and technologies ranking

Ranking Position	Technology	Score (%)
1	B2MML (T_2)	35.7318
2	AutomationML (T_5)	25.0641
3	XMplant (T_3)	25.0051
4	BatchML (T_1)	25.0026
5	OPC UA DM (T_6)	25.0009
6	MTConnect (T_7)	25.0009
7	CAEX (T_4)	25.0008

According purely to the ranking presented in Table 8, the most adequate technology for PERFoRM would be B2MML, followed by AutomationML.

4.5. Discussion of Results

Analysing the results from the Table 8, there are several aspects that may raise some doubts. The values that are presented to rank the technologies present two distinct characteristics:

- The values are very close to each other;
- None of their scores is placed over the 50th percentile, out of 100\%.

These two aspects are fully correlated and based on the fact that the developed methodology is set on three distinct aspects:

- The users interests;
- The importance that each user gives to the evaluated characteristics;
- The number of end users.

The end users' interests are mostly related to the areas in which their own (often very different) production lines' challenges emerge. Taking this into account, the established criteria weights vary in accordance with each vision. This weight variance, which can be seen in Table 5, added to the low number of end users may translate into some instability in the technology assessment (Table 7).

Being this discrepancy so high, technologies which match a high number of criteria (in this case T_2), can be influenced severely (Table 7) by this lack of consensus. This fact is translated in the score (Table 8) by a higher percentage, although, it is still under the 50% mark.

For the other technologies, the low number of characteristics linked to higher weights (Table 4 and Table 5) gives them, based on a uniform average value and a low discordance, close and relatively high score values.

As mentioned in the previous section, solely from analysing the numbers in Table 8 one would conclude B2MML to be the best suited format for the project, maybe coupled with AutomationML (as the second highest ranking format) to tackle its shortcomings.

However, after a more careful analysis of the characteristics of both data formats, it was concluded that AutomationML could be easily extended to cover its missing aspects without requiring any additional format to be used in parallel. Additionally, B2MML appears to be slightly less flexible than AutomationML, and despite it being an XML-based format, any extensions altering or adding to the existing definitions would make it a non-standard, thus defeating the purpose of using the format in the first place, something that does not occur with AutomationML.

With these considerations, AutomationML was elected as the chosen data format to implement PERFoRM’s common data model.

5. Design of PERFoRMML

This section entails the description of PERFoRMML, PERFoRM's common data model. The design of PERFoRMML was conducted in AutomationML taking into consideration all the requirements from the different use cases and tool developers, as presented in Sections 2 and 3. The overall view of the class diagram can be seen in Figure 8.

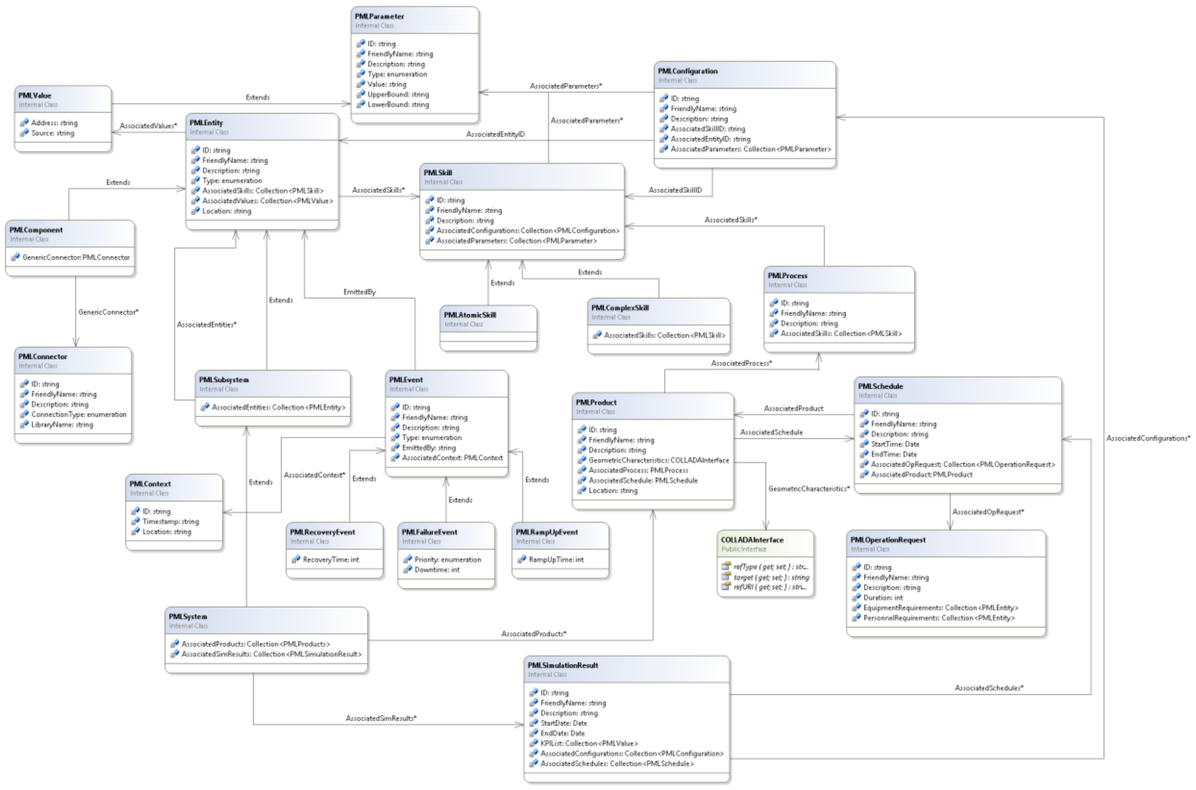


Figure 8 - PERFoRMML Class Diagram

Each of the classes depicted in Figure 8 will be described in further detail in 5.1 and 5.2 (and their respective sub-sections).

5.1. Machinery and Control Systems

The Machinery and Control Systems layer encompasses all the elements necessary to model the system's topology, data types and interaction at physical machinery level. Each of the required concepts is detailed in the subsequent subsections.

5.1.1. PMLParameter and PMLValue

The *PMLParameter* and *PMLValue* elements enable the basic representation of information pertaining to data at the machinery level, namely in terms of parameters for configurations and skills (abilities, functions or tasks performed by shop-floor elements), and shop-floor data to be extracted from various sources such as PLCs and databases, respectively. Both classes can be seen in Figure 9.

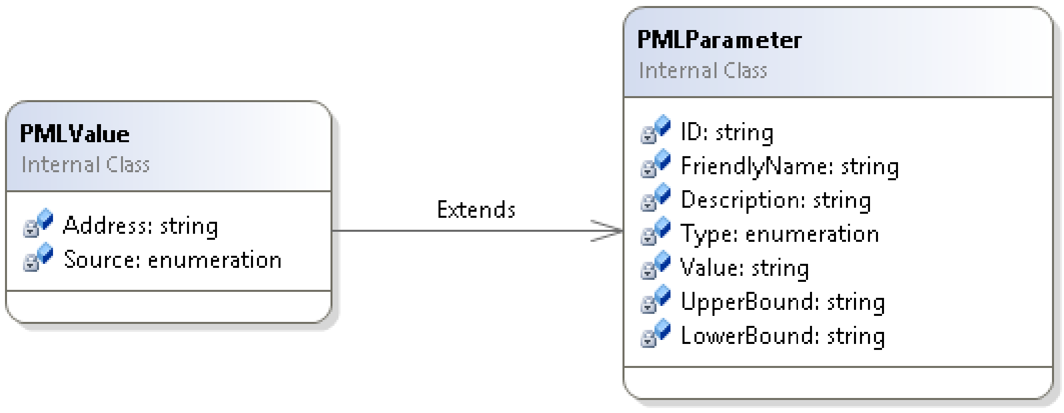


Figure 9 - PMLParameter and PMLValue

The *PMLParameter* concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***Type*** – An enumeration that sub-specifies the type of the specific value or parameter.
- ***Value*** – A generic string containing the actual value.
- ***UpperBound*** – A generic string that when applicable defines the upper bound of the value.
- ***LowerBound*** – A generic string that when applicable defines the lower bound of the value.

As an extension of this concept, *PMLValue* includes not only the aforementioned attributes but also:

- ***Address*** – A string that indicates the address of a value (OPC UA tag, database key, etc.).
- ***Source*** – A string describing where the value can be obtained from (database, PLC, etc.).

5.1.2. PMLEntity

The *PMLEntity* class is the generic representation of shop-floor entities, encapsulating all the necessary information that is associated simultaneously to both components and subsystems (set of components and possibly other subsystems working towards a common goal). Its description can be seen in Figure 10.

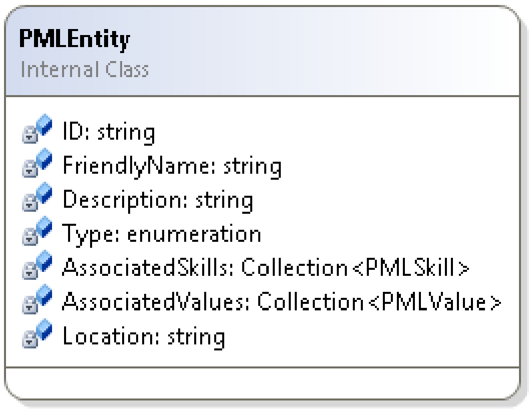


Figure 10 - PMLEntity

Additionally, despite not being directly used, the *PMLEntity* class enables the existence of generic collections of elements that can be either components or subsystems, without *a-priori* knowledge of what the composition of such collections will be. Such a case is described in Subsection 5.1.2.1.

This concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***Type*** – An enumeration that sub-specifies the type of the specific entity.
- ***AssociatedSkills*** – A collection of elements of the *PMLSkill* class, detailing the skills (e.g. pick, place, weld…) of an entity.
- ***AssociatedValues*** – A collection of elements of the *PMLValue* class, detailing the relevant values associated to an entity (e.g. timespans, I/O data…).
- ***Location*** - A generic string indicating the entity's location within the shop-floor (e.g. a pair of x/y coordinates, gps coordinates, etc.).

5.1.2.1. PMLComponent and PMLSubsystem

In order to abstract components and subsystems in the shop-floor, both the *PMLComponent* and *PMLSubsystem* classes extend the characteristics of a *PMLEntity*, as it can be observed in Figure 11.

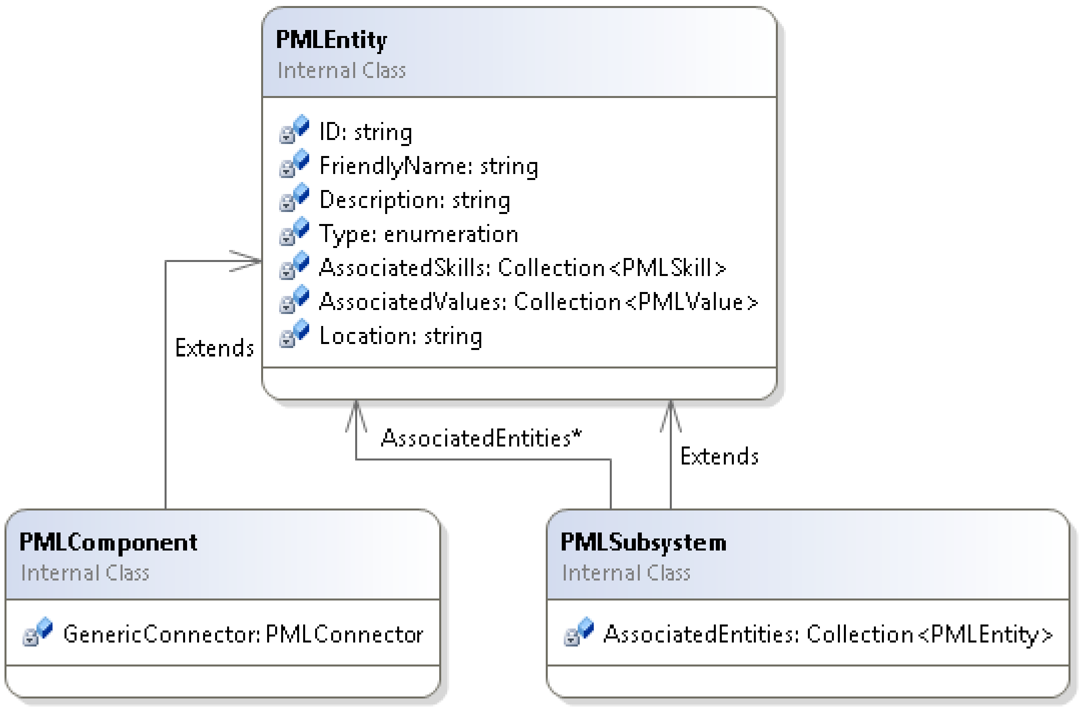


Figure 11 - PMLComponent and PMLSubsystem

The former indicates the finest level of granularity, therefore representing a single component in the shop-floor which may offer a given number of skills (e.g. pick, place, move, weld) and may possess certain values that are relevant to be extracted (e.g. cycle time, energy consumption, sensor data). This representation is generic enough in order for components to be able to refer not only to physical machine resources, but also to virtual ones or even human operators. As such, a *PMLComponent* is essentially a single entity that can perform skills and present relevant data regarding its state and operation.

In turn, the *PMLSubsystem* provides similar functionality, albeit regarding subsystems instead, thus referring to a group of components and possibly other subsystems. It is a recursive element in the sense that it extends the *PMLEntity* class, whilst also being able to contain other *PMLEntities* within.

Furthermore, the generic design allows different levels of granularity to be targeted using the same data model. A system can be modelled in such a way that a robot is the lowest entity in terms of the abstraction level, being regarded as a simple component, while that same robot can be seen as a subsystem within the system itself, encompassing several sensors as its components, depending on the desired level of granularity.

The *PMLComponent* and the *PMLSubsystem* have the following attributes, being an extension of the *PMLEntity* concept:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.

- ***Type*** – An enumeration that sub-specifies the type of the specific entity (in this case component or subsystem).
- ***AssociatedSkills*** – A collection of elements of the *PMLSkill* class, detailing the skills (e.g. pick, place, weld…) of an entity.
- ***AssociatedValues*** – A collection of elements of the *PMLValue* class, detailing the relevant values associated to an entity (e.g. timespans, I/O data…).
- ***Location*** - A generic string indicating the entity's location within the shop-floor (e.g. a pair of x/y coordinates, gps coordinates, etc.).

Additionally, the *PMLSubsystem* includes:

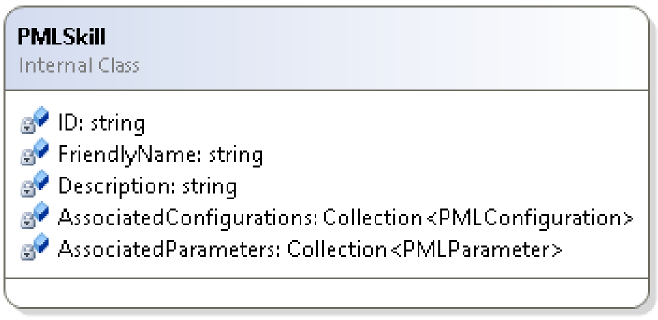
- ***AssociatedEntities*** – A collection of elements of the *PMLComponent* or *PMLSubsystem* class, containing the entities comprising the subsystem.

Finally, the *PMLComponent* also encompasses:

- ***GenericConnector*** – A *PMLConnector*, specifying how to interact with the resource at the shop-floor level.

5.1.3. PMLSkill

The tasks that a given *PMLComponent* or *PMLSubsystem* can perform are exposed as a *PMLSkill*. This class can be seen in Figure 12.



PMLSkill
Internal Class

- ID: string
- FriendlyName: string
- Description: string
- AssociatedConfigurations: Collection<PMLConfiguration>
- AssociatedParameters: Collection<PMLParameter>

Figure 12 – PMLSkill

A skill is characterized by a unique identifier, as well as by a series of associated possible configurations (required for it to be executed) and parameters, enabling the control of its parameterization. Similarly to the *PMLEntity*, the *PMLSkill* is not directly used, enabling instead the creation of generic collections of *PMLAtomicSkill* and *PMLComplexSkill* entities, which will be described in Subsection 5.1.3.1.

This concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.

- ***AssociatedConfigurations*** – Collection of elements of the *PMLConfiguration* class, representing the possible configurations associated with a given skill.
- ***AssociatedParameters*** – Collection of elements of the *PMLParameter* class, representing the possible parameters associated with a given skill.

5.1.3.1. PMLAtomicSkill and PMLComplexSkill

Conceptually, skills can be divided into two categories, atomic skills and complex skills, abstracted in the data model by the *PMLAtomicSkill* and *PMLComplexSkill* classes (Figure 13).

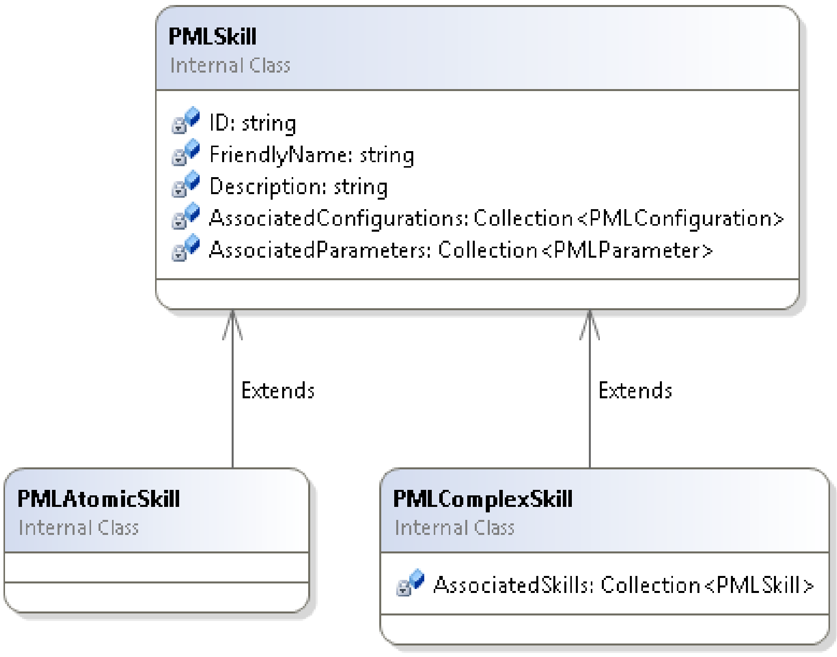


Figure 13 - PMLAtomicSKill and PMLComplexSKill

The former embodies the simplest form of a skill, representing a single action performed by a given entity. The latter is used to specify a skill which consists in the combination of multiple skills, which can be both atomic and complex, thus also being recursive in its definition.

The *PMLComplexSkill* element has the following attributes, being an extension of the *PMLSkill* concept:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***AssociatedConfigurations*** – Collection of elements of the *PMLConfiguration* class, representing the possible configurations associated with a given skill.

- ***AssociatedParameters*** – Collection of elements of the *PMLParameter* class, representing the possible parameters associated with a given skill.
- ***AssociatedSkills*** – Collection of elements of the *PMLSkill* class, which can thus be either atomic or complex skills. These describe the combination of skills of which the respective complex skill is composed of.

5.1.4. PMLConfiguration

The *PMLConfiguration* class provides a high-level description of a possible configuration to execute a given skill, according to a set of specified parameters. Its composition is represented in Figure 14.

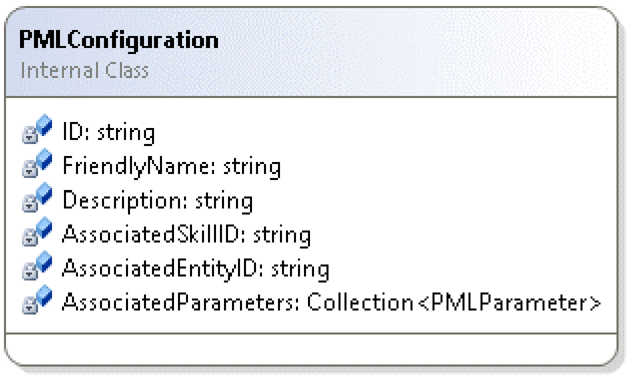

PMLConfiguration
Internal Class
ID: string
FriendlyName: string
Description: string
AssociatedSkillID: string
AssociatedEntityID: string
AssociatedParameters: Collection<PMLParameter>

Figure 14 – PMLConfiguration

The *PMLConfiguration* concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***AssociatedSkillID*** – A string containing the unique identifier of the associated skill.
- ***AssociatedEntityID*** – A string containing the unique identifier of the associated entity.
- ***AssociatedParameters*** – Collection of elements of the *PMLParameter* class, representing the possible parameters associated with a given configuration, enabling the parametrization of different configurations.

5.1.5. PMLProduct

The *PMLProduct* class provides an abstraction of a given product variant, along with its core-defining characteristics to enable a process-oriented description of the product, as seen in Figure 15.

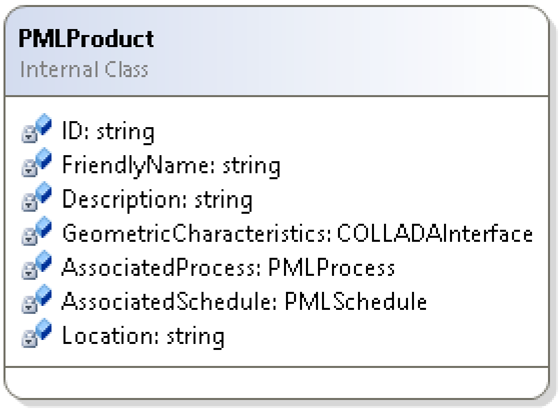

PMLProduct
Internal Class
ID: string
FriendlyName: string
Description: string
GeometricCharacteristics: COLLADAInterface
AssociatedProcess: PMLProcess
AssociatedSchedule: PMLSchedule
Location: string

Figure 15 – PMLProduct

The product is defined by a mandatory unique identifier, along with a full description of the required steps of its production process and respective schedule. Additionally, it can contain a link to an external description of its geometric characteristics, using for this purpose the COLLAborative Design Activity (COLLADA) interchange format, an open standard XML schema for the exchange of 3D assets among software applications.

The *PMLProduct* concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***GeometricCharacteristics*** – A *COLLADAInterface*, an AutomationML element connecting the *PMLProduct* to an external COLLADA file containing the description of the products geometric characteristics.
- ***AssociatedProcess*** – A *PMLProcess* detailing the ordered tasks (recipe) necessary to produce the associated product.
- ***AssociatedSchedule*** – A *PMLSchedule* describing the product's production schedule.
- ***Location*** – A generic string indicating the product's location for traceability purposes (for example it can be not only a pair of coordinates, but also the ID of a resource or station, depending on the agreed semantics).

5.1.6. PMLProcess

The process presents a description of the ordered steps required for the production of an associated product. This representation is abstracted by the *PMLProcess* class, as seen in Figure 16.

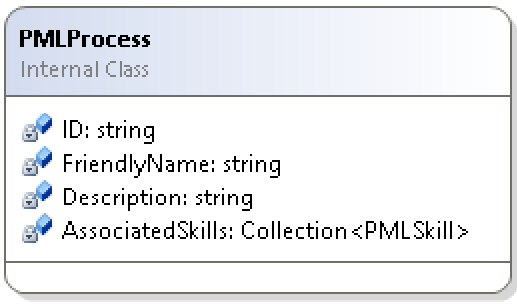


Figure 16 – PMLProcess

This concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***AssociatedSkills*** – An ordered collection of elements of the *PMLSkill* class, indicating the list of production steps associated to a given production process.

5.1.7. PMLConnector

The *PMLConnector* class encapsulates the information required in order to communicate with a component in the shop-floor. This is a generic abstraction allowing the data model to be applicable to different systems, regardless of the wide array of possible communication protocols that can be used. Thus, as shown in Figure 17, *PMLConnector* indicates both the connection type (e.g. OPC UA, PROFINET), as well as the interface that should be used to communicate with said component.

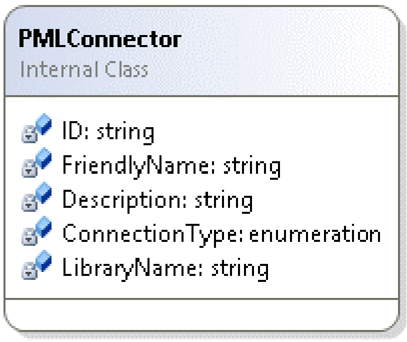


Figure 17 – PMLConnector

The *PMLConnector* concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.

- ***ConnectionType*** – An enumeration specifying the type of connection (e.g. OPC UA, SQL Database)
- ***LibraryName*** – A string indicating the name of the library to be used for communication.

5.1.8. PMLEvent

Events deal with certain relevant occurrences in production that may require the attention of the system or its users. The *PMLEvent* class depicted in Figure 18 abstracts this concept and supports the definition of the concrete types specified in Subsection 5.1.8.1.

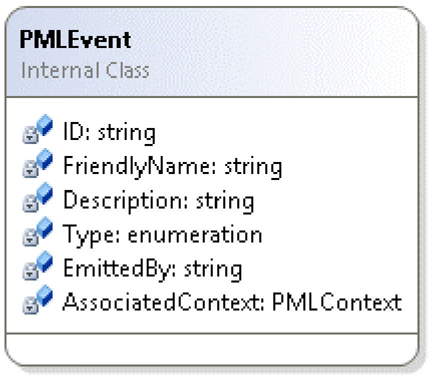


Figure 18 – PMLEvent

These concepts have the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***Type*** – An enumeration that sub-specifies the type of the event.
- ***EmittedBy*** – A string referencing the source of the event.
- ***AssociatedContext*** – A *PMLContext* referring to the context in which the event took place.

5.1.8.1. Event Types

The different event types extend the *PMLEvent* class, adding relevant information to its definition, including data regarding the occurrence of failures, component or subsystem ramp-ups and recovery from failures during production. Each of this is represented in Figure 19, along with the information added to the initial definition.

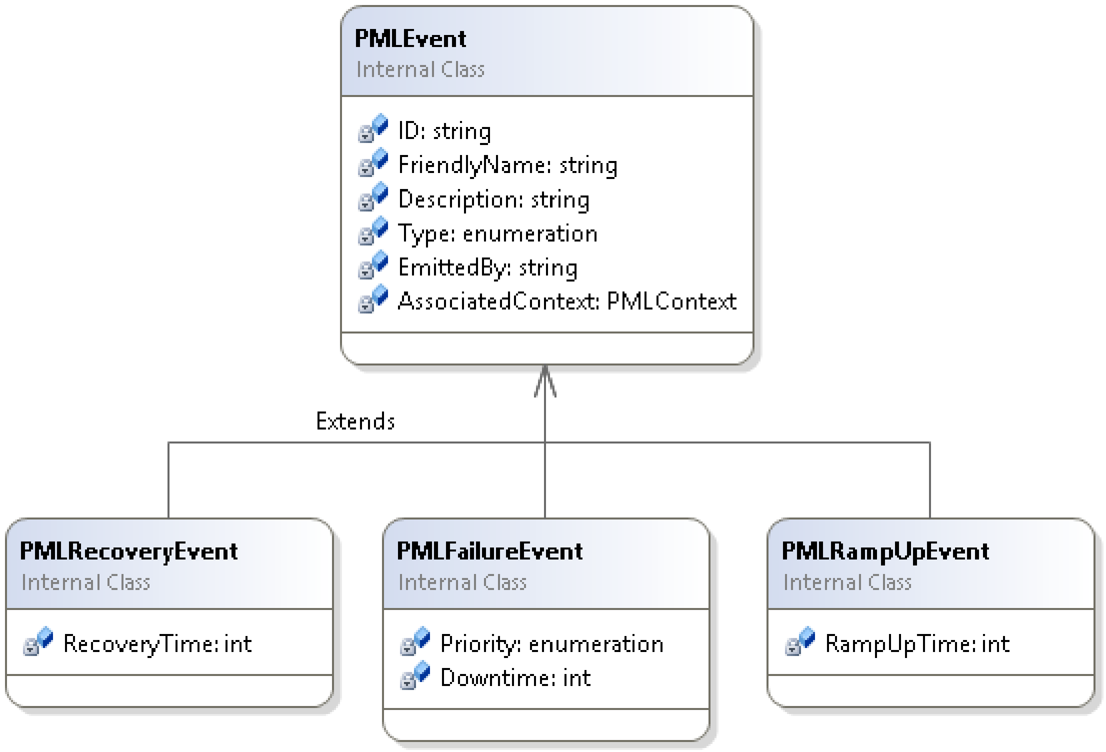


Figure 19 - Event Types

As an extension of the *PMLEvent*, these concepts add the following attributes:

- ***RecoveryTime*** *(PMLRecoveryEvent)* – The overall time required to recover the system from a failure to a normal operation state.
- ***Priority*** *(PMLFailureEvent)* – An enumeration indicating the priority of a given failure event (e.g. low, medium, high).
- ***Downtime*** (*PMLFailureEvent*) – The overall downtime caused by a failure event.
- ***RampUpTime*** (*PMLRampUpEvent*) – The overall time required to ramp-up a procedure.

5.1.9. PMLContext

The PMLContext concept provides the general context in which a PMLEvent type took place, more specifically regarding the recorded time and location of the occurrence, as shown in Figure 20.

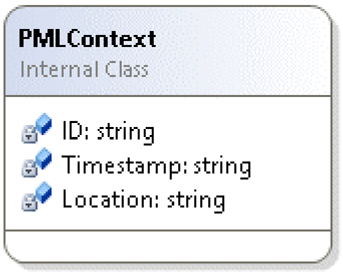


Figure 20 – PMLContext

This concept has the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***Timestamp*** – The timestamp indicating the instant in which something took place.
- ***Location*** – A generic string indicating the location at which something took place (for example it can be not only a pair of coordinates, but also the ID of a resource or subsystem, depending on the agreed semantics).

5.2. Data Backbone

The Data Backbone layer encompasses the elements necessary for the interactions with the tools connecting to the middleware, which through it should be able to acquire data and information from the lower-level and act based on it. This includes higher-level system descriptions and information pertaining to specific archetypes of tools (e.g. simulation, scheduling), as described in Subsections 5.2.1 through 5.2.3.

5.2.1. PMLSystem

The PMLSystem extends the PMLSubsystem concept in order to represent the whole system, thus standing at a higher level of abstraction. As such, it contains all the relevant system information in terms of topology, products and possible simulations that have been executed, as it can be seen in Figure 21.

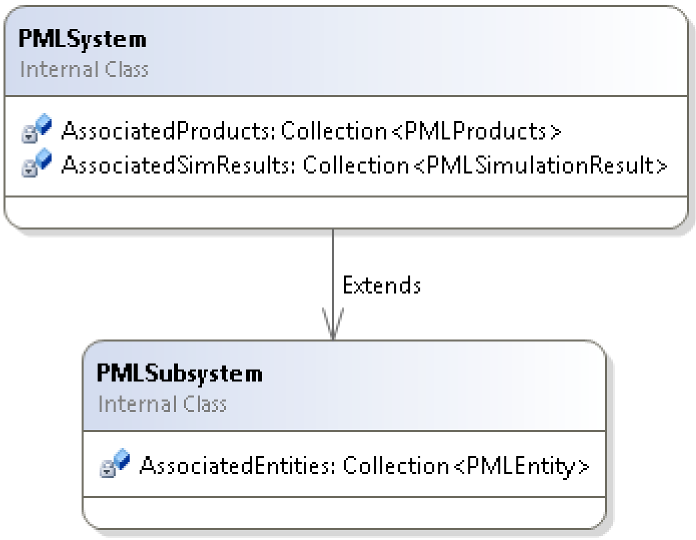

PMLSystem
Internal Class
AssociatedProducts: Collection<PMLProducts>
AssociatedSimResults: Collection<PMLSimulationResult>
Extends
PMLSubsystem
Internal Class
AssociatedEntities: Collection<PMLEntity>

Figure 21 - PMLSystem and PMLSubsystem

A *PMLSystem* has the following attributes:

- ***AssociatedProducts*** – A collection of elements of the *PMLProduct* class, representing the products associated to a given system.
- ***AssociatedSimResults*** – A collection of elements of the *PMLSimulationResult* class, containing the results from simulations related to the system.

5.2.2. PMLSimulationResult

The *PMLSimulationResult* element is used to represent a simulation output, namely in terms of the resulting Key Performance Indicators (KPI) for a given set of configurations and/or schedules, as seen in Figure 22.

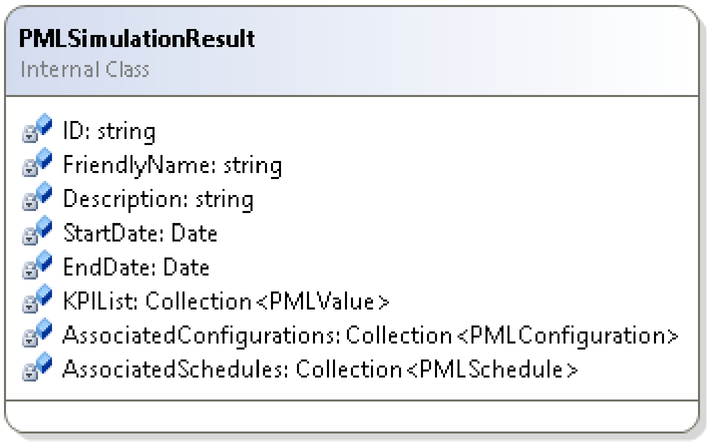

PMLSimulationResult
Internal Class
ID: string
FriendlyName: string
Description: string
StartDate: Date
EndDate: Date
KPIList: Collection<PMLValue>
AssociatedConfigurations: Collection<PMLConfiguration>
AssociatedSchedules: Collection<PMLSchedule>

Figure 22 – PMLSimulationResult

A *PMLSimulationResult* presents the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***StartDate*** – The timestamp associated to the instant the simulation began.
- ***EndDate*** – The timestamp associated to the instant the simulation terminated.
- ***KPIList*** – A collection of elements of the *PMLValue* class, representing the KPIs resulting from running the simulation.
- ***AssociatedConfigurations*** – A collection of elements of the *PMLConfiguration* class, indicating the system configurations associated with a given simulation task.
- ***AssociatedSchedules*** – A collection of elements of the *PMLValue* class, presenting the schedules associated with a given simulation task.

5.2.3. PMLSchedule and PMLOperationRequest

A *PMLSchedule* represents the allocation of the steps, from start to finish, that need to be executed in order for a certain product to be produced. This description entails not only the necessary operation tasks and their duration, represented by the *PMLOperationRequest* concept, but also the requirements in terms of equipment and personnel allocation. This description is depicted in Figure 23.

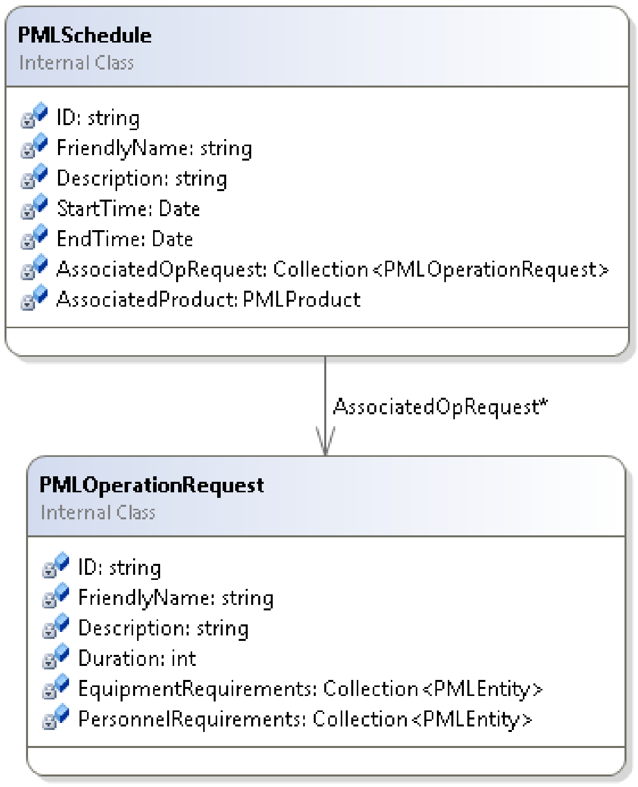


Figure 23 - PMLSchedule and PMLOperationRequest

A *PMLSchedule* encompasses the following attributes:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***StartTime*** – A timestamp indicating the instant production is scheduled to start.
- ***EndTime*** – A timestamp indicating the instant production is scheduled to end.
- ***AssociatedOpRequest*** – A collection of elements of the *PMLOperationRequest* class, indicating the scheduled operation tasks associated to a given schedule.
- ***AssociatedProduct*** – An element of the *PMLProduct* class, referencing the product associated with a given schedule.

The *PMLOperationRequests* consists in:

- ***ID*** – A string that serves as the unique identifier for this element.
- ***FriendlyName*** – A string that provides the readable name of the element.
- ***Description*** – A string containing a description of the element.
- ***Duration*** – The expected duration of a given operation task.
- ***EquipmentRequirements*** – A collection of elements of the *PMLEntity* class, detailing the equipment (e.g. station, resource) necessary to be allocated to a given scheduled operation.
- ***PersonnelRequirements*** – A collection of elements of the *PMLEntity* class, detailing the personnel necessary to be allocated to a given scheduled operation.

5.3. Standard Generic Interfaces

As previously mentioned in Section 4, one of the key aspects for tackling PERFoRM's interoperability challenges, mainly in regards to the seamless exchange of data between heterogeneous entities, is the adoption of standard interfaces. These act as the main drivers for pluggability and interoperability, enabling the interconnection of both software and hardware entities across the different manufacturing layers in a seamless and transparent fashion.

The interfaces should provide the devices, tools and applications with the means to fully expose and describe their services in a standardized way, compliant with the data model described in Section 5. This includes fully specifying the semantics and data flow involved in terms of inputs and outputs required to interact with these elements.

As such, full interoperability and harmonization of data at a system of systems level is achieved by coupling the standard interfaces with the data model for a common representation of data and system semantics. However, taking into account the integration of legacy devices and their own individual data models and semantic requirements, the addition of technology adaptors (see Section 7.2) is also required in order to enable the translation and mapping of legacy data into the common PERFoRM representation, allowing for these devices to be conferred additional intelligence and integrated into the cyber-physical paradigm.

As described in the PERFoRM architecture, two clear abstraction levels can be identified, namely the lower level, concerning the machinery and control systems, and the higher-level which includes the different tools that can be plugged to the middleware. Hence, two generic interfaces were defined, being detailed in Sections 5.3.1 and 5.3.2, respectively.

5.3.1. Machinery and Control Systems Interface

The machinery level interface is responsible for opening a communication channel between the middleware and the shop-floor in physical layer. This should include I/O manipulation, data extraction and the capacity to send different configurations to the resources. An overview of this interface can be seen in Figure 24.

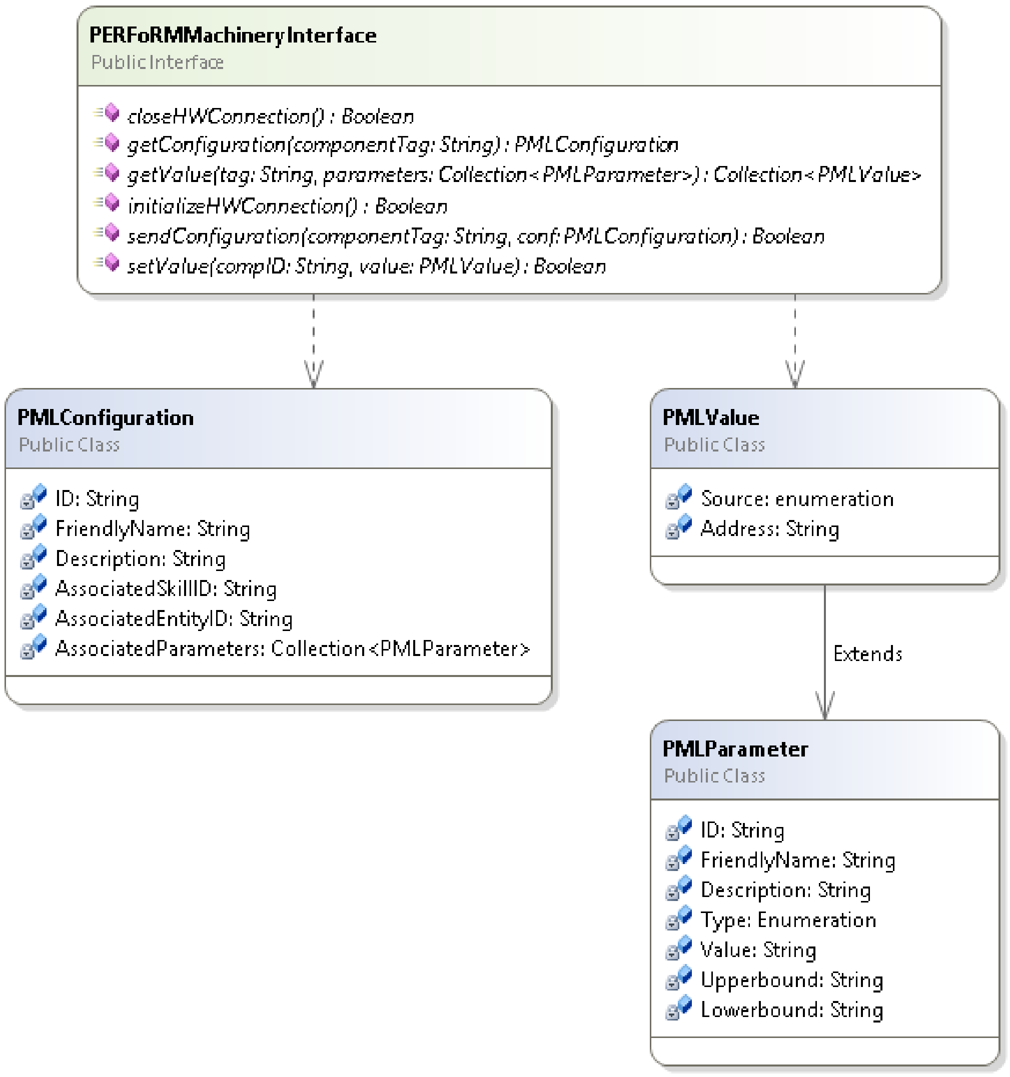

PERFoRMMachineryInterface
Public Interface
closeHWConnection() : Boolean
getConfiguration(componentTag: String) : PMLConfiguration
getValue(tag: String, parameters: Collection<PMLParameter>) : Collection<PMLValue>
initializeHWConnection() : Boolean
sendConfiguration(componentTag: String, conf: PMLConfiguration) : Boolean
setValue(compID: String, value: PMLValue) : Boolean
PMLConfiguration
Public Class
ID: String
FriendlyName: String
Description: String
AssociatedSkillID: String
AssociatedEntityID: String
AssociatedParameters: Collection<PMLParameter>
PMLValue
Public Class
Source: enumeration
Address: String
Extends
PMLParameter
Public Class
ID: String
FriendlyName: String
Description: String
Type: Enumeration
Value: String
Upperbound: String
Lowerbound: String

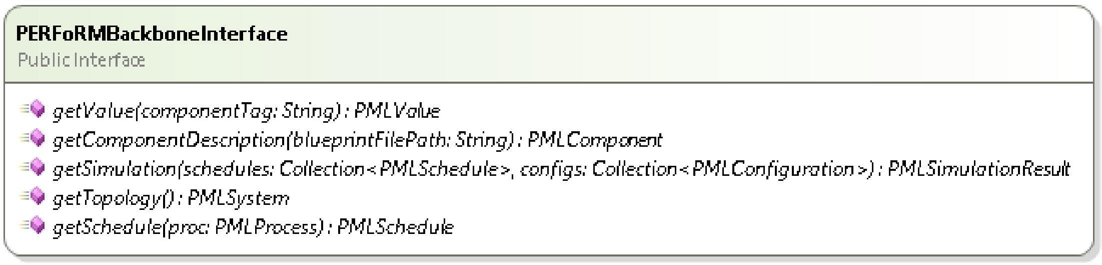
Figure 24 - Machinery and Control Systems Interface

The interface consists in the following methods:

- ***initializeHWConnection*** – Initializes the connection to the lower level. Returns the result of the operation as a Boolean value.
- ***closeHWConnection*** – Closes the connection to the lower level. Returns the result of the operation as a Boolean value.
- ***getValue*** – Retrieves a value or a collection of values from the machinery layer (e.g. sensor data). Receives as an input an id tag and a collection of parameters characterizing the request's specification, according to a semantic agreed beforehand (e.g. begin and end dates for historical values) . Returns a collection of elements of the *PMLValue* class.
- ***setValue*** – Sets a specific input of a resource to the desired value. Receives as an input the resource identifier as well as the value to be set. Returns the result of the operation as a Boolean value.
- ***getConfiguration*** – Returns the configuration associated with a given resource. Receives as an input the resource identifier. Returns a *PMLConfiguration*.
- ***sendConfiguration*** – Sends a new parameterized configuration to a resource. Receives as an input the target resource's identifier, as well as the desired configuration. Returns the result of the operation as a Boolean value.

5.3.2. Data Backbone Interface

The backbone level interface is responsible for exposing the functionalities of the tools connected to the middleware, as well as allowing these tools to communicate with the lower levels in order to acquire any information required to execute their respective tasks. An overview of this interface can be seen in Figure 25.


PERFoRMBackboneInterface
Public Interface

```
getValue(componentTag: String) : PMLValue
getComponentDescription(blueprintFilePath: String) : PMLComponent
getSimulation(schedules: Collection<PMLSchedule>, configs: Collection<PMLConfiguration>) : PMLSimulationResult
getTopology() : PMLSystem
getSchedule(proc: PMLProcess) : PMLSchedule
```

Figure 25 - Backbone Interface

The backbone interface consists in the following methods:

- ***getValue*** – Retrieves a required value from the system through the middleware. Receives as an input the identifier for the value to be retrieved. Returns a *PMLValue*.
- ***getComponentDescription*** – Returns the blueprint of a resource as a *PMLComponent* element (e.g. descriptions, capabilities, etc.) Receives as an input a string containing the path to the resource's blueprint file.

- ***getSimulation*** – Requests a simulation of the system to be performed using a given set of schedules and configurations. Receives as an input two collections, one *PMLSchedule* elements and the other of *PMLConfiguration* elements. Returns the simulation results as a *PMLSimulationResult object*.
- ***getTopology*** – Retrieves the current system state as a *PMLSystem* element.
- ***getSchedule*** – Requests a new production schedule for a given desired process. Receives as an input a *PMLProcess*. Returns a new schedule as a *PMLSchedule* element.

6. Data Model Application

6.1. IPB Workshop – Punching Cell

The first workshop organized in relation to the WP2 developments was planned in order to harmonize the efforts between this work package and WP3, with a focus on the integration of the data model's machinery level (T2.3), the connectors (T3.1) and the middleware (T2.4).

The workshop's test case focuses on a punching cell consisting in a conveyor belt and its DC motor (*M1*), a punching cylinder and respective DC motor (*M2*), two part presence sensors (*S1* and *S2*, phototransistors) and two switches (*A1* and *A2*), as illustrated in Figure 26.

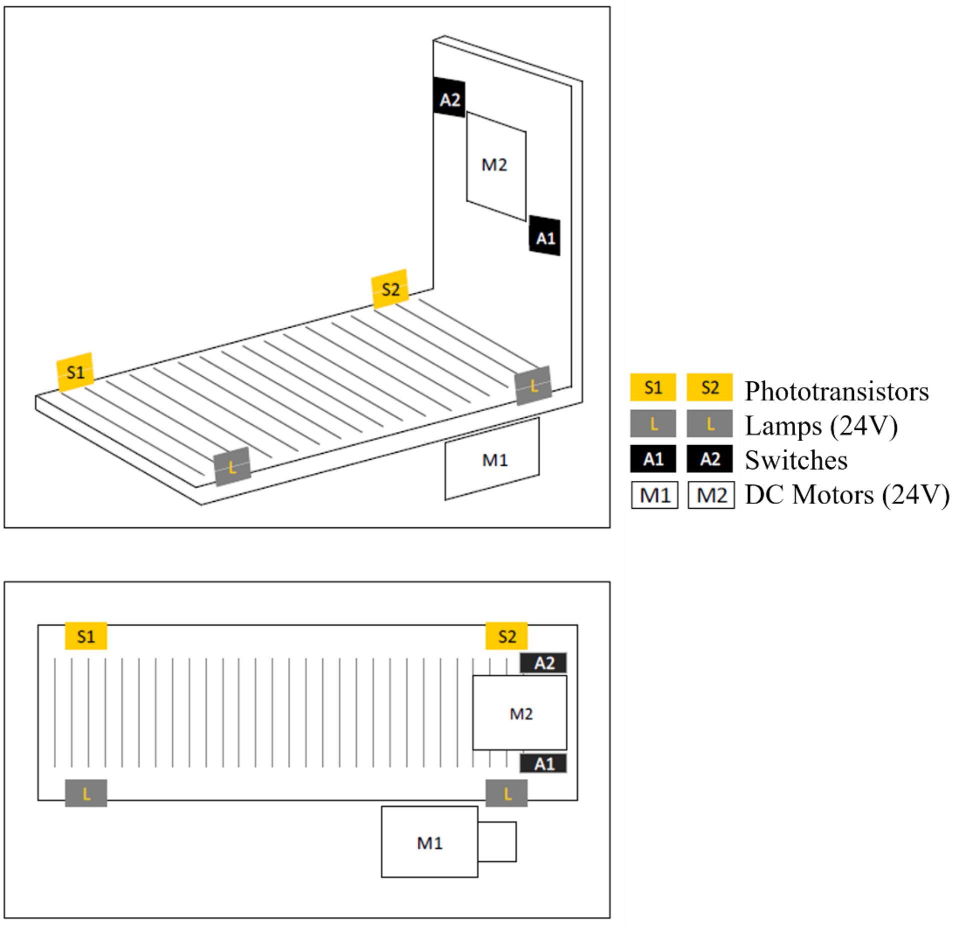


Figure 26 - Punching Cell Overview

The relevant data available to be extracted from the cell is described in Table 9.

Table 9 - Punching Cell Values

Entity	Value	Source
M1	M1F	PLC

Entity	Value	Source
M1	M1T	PLC
M2	M2D	PLC
M2	M2U	PLC
M1	S1	PLC
M1	S2	PLC
M2	A1	PLC
M2	A2	PLC
M1	Humidity	DB
M1	Temperature	DB
M1	HeatIndex	DB
M1	BatteryVoltage	DB
M1	Humidity	DB
M2	Temperature	DB
M2	Pressure	DB
Cell	InExecution	PLC
Cell	ProcessingTime	PLC

The punching process is relatively simple, consisting essentially in a part entering the cell at S1, being then moved to S2 in order for the motor M2 to initiate the punching of said part. Afterwards, the part is simple moved back along the conveyor (controlled by M1) to S1.

The process flow works as follows:

S1 (1) > M1F (1) > S1 (0) > S2 (1) > M1F (0) > M2D (1) > A2 (0) > A1 (1) > M2D (0) > M2U (1) > A1 (0) > A2 (1) > M2U (0) > M1T (1) > S2 (0) > S1 (1) > M1T (0)

While the PLC provides the low-level, sensor and I/O data regarding each of the components, as shown in Table 9, a database also provides some additional information for both motors, mainly regarding readings such as temperature, humidity, voltages and hydraulic pressure.

For the workshop's purposes, the granularity target was defined at the motor level, being clear that these two entities, M1 and M2, provide the four main driving skills for the cell, namely the conveyor control, M1F and M1T, and the punching control, M2U and M2D, respectively.

As such, taking into account the relevant values extracted from Table 9, as well as the overall topological organization of the cell, the following PERFoRMML model was obtained, as shown in Figure 27.

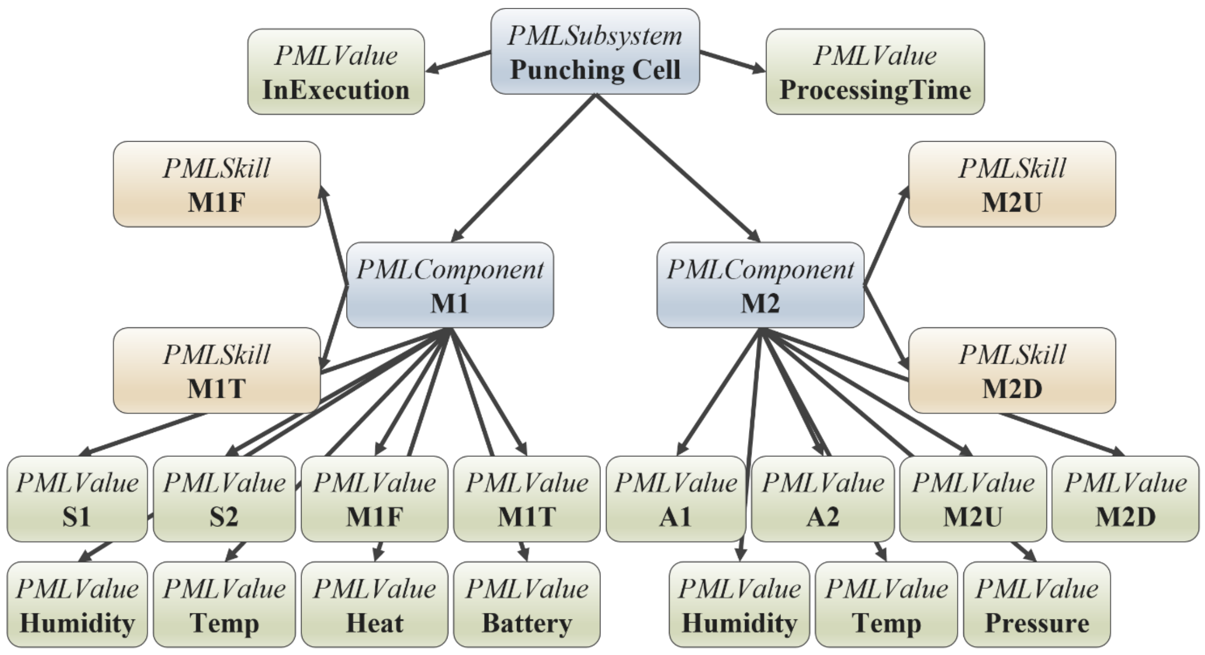


Figure 27 - Punching Cell Modelling

7. Interactions with other architectural elements

7.1. Middleware

One of the core elements of the PERFoRM architecture is the Middleware. It plays a key role for the horizontal and especially the vertical integration of production devices, as it acts as a platform where all components of the architecture can connect to in order to achieve multiple goals, such as:

- Establish a link between a client and a server application for data exchange
- Provide data translation functionalities from one data format into another
- Provide discovery mechanisms to dynamically discover functionalities provided in the system

How these functionalities are achieved internally is described in detail in D2.4. The Middleware is designed to have a scalable, distributed approach. It is not a single software solution, but a set of solutions which in combination are able to achieve the goals set above in a performant way.

Within the scope of PERFoRM, the Middleware will be especially important as a link between management systems, such as ERP, MES or Simulation systems and the various production devices and systems on the shop floor level. Because of this, a major task of the Middleware is the translation of data from one schema into another, e.g. translating a business level model into a machinery level model. For that reason, the results of D2.3 are a direct input for the Middleware implementation, as the translation of these models needs to be configured within the Middleware.

Additionally, the standardization of interfaces to access the Middleware is necessary. For example, the interfaces defined within this deliverable need to include standardized formats and protocols for accessing the Middleware's discovery mechanisms.

7.2. Technology adapters

Manufacturing companies are usually characterized by the use of legacy and heterogeneous systems for the management and the execution of their production process. At machinery level (L1 and L2 layers of ISA 95 standard) example of these systems are robots, CNC machines, Programmable Logic Controllers (PLCs) and Human Machine Interfaces (HMIs); at backbone level (L3 and L4 layers of the ISA 95 standard) examples of these systems are Enterprise Resource Planning (ERP), Manufacturing Execution System (MES) and Production Databases (DBs). The innovative architecture proposed in the PERFoRM project can be industrially accepted and really adopted only if the possibility to integrate the legacy systems is presented. For this reason, technology adapters are key elements to connect legacy systems to the PERFoRM middleware and to transform the legacy data model into the standard interface data model defined in Task 2.3 of the project.

As depicted in the following figure, three different kinds of adapters are considered in the WP3 and tackled in its tasks. These adapters respond to the different types of legacy systems which can be found in a production environment and are able to seamlessly connect these systems with the industrial middleware and the higher level of the enterprise network (ERP, MES, etc.).

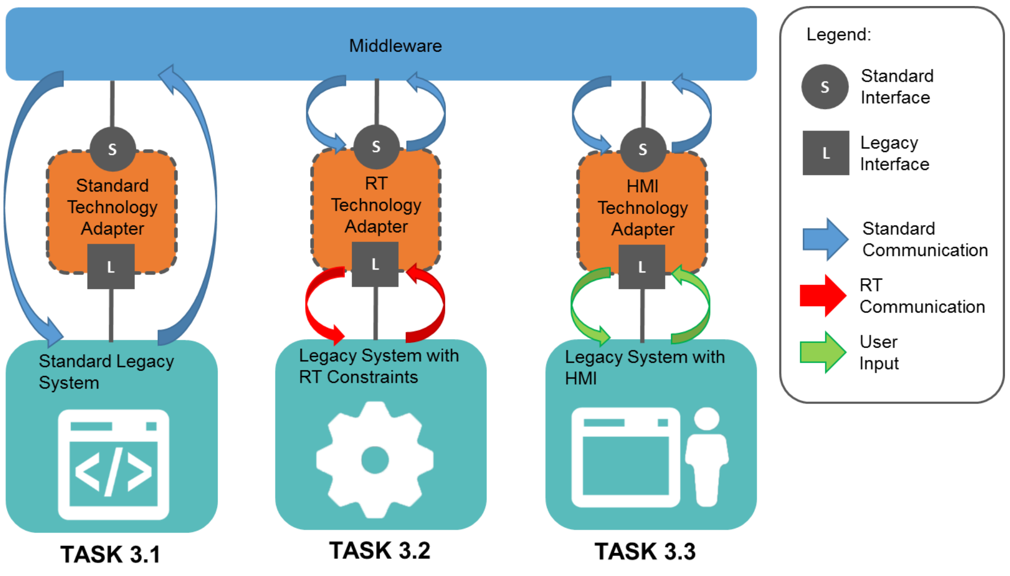


Figure 28 - Technology Adapters Types

Real-Time constraints are particularly important when considering CNC machines and robotic cells as they may need quick adjustments and corrections according to the data acquired from low-level sensors locally installed in the production resource (e.g. vibration analysis of spinning spindles). Currently, no hard real-time constraints were identified in the use cases. However, should they arise, this aspect will be tackled within Task 3.2, as depicted above. HMIs, instead, can be used not only for monitoring and controlling the production resource but also for capturing human expert knowledge and support following human activities from past experience (e.g. change over and ramp-up operations can be supported by policies derived from past cases).

Following the indications coming from the WP1, for each of the four use cases addressed in the PERFoRM project a list of the legacy systems that need to be connected with industrial middleware has been created and reported in Table 10.

Table 10 - Legacy Systems

Use Case	**Objective**	**Legacy Systems**
Siemens Compressors	Integration of a predictive maintenance system	➢ EPR System (SAP APO) ➢ BDE Data Logging System (Oracle DB) ➢ LHnet Ticketing System (SQL DB) ➢ CNC Machines (SINUMERIK 840D)

IFEVs Micro-Electrical Vehicles	Automation of the production line	➢ Welding Robotic Cells and Powertrain Testing Stations (Siemens PLC IM-151)
Whirlpool Microwave Ovens	➢ Implementation of a KPI real-time monitoring system ➢ Reconfiguration of the path of the robot for the leak test	➢ PERFoRM DB (SQL DB) ➢ PLM Repository (txt file) ➢ Leak Robot Station (UR10 Controller)
GKN Turbine Vanes	Construction of a reconfigurable robotic cell	➢ Robotic Cell PC/PLC ➢ Roughness Process (Mitutoyo SJ-210)

The integration of the hardware equipment and software applications listed above requires the use of proper technological adapters to transform the native data format into the data model defined by PERFoRM.

Moreover, the implementation of the adapters is strongly dependent of the selected technology for the industrial middleware (Task 2.4 of the project). For example, Siemens WinCC OA (Open Architecture) provides a direct PLC interface which greatly simplifies the implementation of adapters for such kind of hardware equipment. PLC integration is particularly important for the IFEVs and GKN use cases where PLCs are used to control the robotic cells. Another example of middleware technology considered in the PERFoRM project is the IBM Integration Bus. This solution offers, among other interesting features, a Database Input node that permits to retrieve updated data directly from a database: it creates a message flow that quickly reacts to changes to application data held in the database. Database connection and integration is particularly important for the Siemens and Whirlpool use cases where databases contain the information needed for feeding the predictive maintenance system (Siemens use case) and the KPI monitoring systems (Whirlpool use case).

8. Conclusion

This document encompassed the description of both the common data model for PERFoRM, as well as of the generic interfaces to be used as the main drivers for seamless interconnectivity and data exchange between the heterogeneous elements comprising the system.

To this end, the document starts by presenting an analysis of the generic use case requirements derived from WP1, moving to a further analysis of the specific requirements presented by each of the tools being developed in WP4.

Afterwards, an assessment of the current data exchange formats was presented, aiming to justify the usage of AutomationML as the foundation for the design of PERFoRMML, PERFoRM's common data model. With this, the design of PERFoRMML was fully described, encompassing two different layers of abstraction:

- *Machinery and Control Systems* – Entails all the elements necessary to model the system's topology, data types and interaction at the physical machinery level.
- *Data Backbone* – Encapsulates all the information required for the interactions with the tools plugged to the middleware, including higher-level system descriptions and information related to each of the specific archetypes of tools, such as definitions of simulations and schedules.

Furthermore, generic standard interfaces were defined, one for each abstraction level, fully exposing the functionalities of each of their respective elements.

Finally, to demonstrate the usage of the data model and its integration with the remaining architectural elements of the project, an application case was described and modelled for the WP2/WP3 integration workshop, thus successfully showcasing the generic applicability of PERFoRMML, ease of use and its capacity to meet the project's requirements.

References

[1] Perf. Project, *Deliverable D7.1 - Siemens Description and Requirements of Architectures for Retrofitting Production Equipment*. 2016.

[2] Perf. Project, *Deliverable D8.1 - Micro Electric Vehicles Description and Requirements of Architectures in View of Flexible Manufacturing*. 2016.

[3] Perf. Project, *Deliverable D9.1 - Description of Requirements and Architecture Design*. 2016.

[4] Perf. Project, *Deliverable D10.1 - Use Case goals/KPIs and Requirements Defined*. 2016.

[5] Perf. Project, "Deliverable D1.2 - Requirements for Innovative Production System Functional Requirement Analysis and Definition of Strategic Objectives and KPIs," 2016.

[6] Perf. Project, *Deliverable D2.1 - Guidelines for Seamless Integration of Humans as Flexibility Driver in Flexible Production Systems*. 2016.

[7] Perf. Project, *Deliverable D2.2 - Definition of the System Architecture*. 2016.

[8] B. Scholten, *The road to integration: A guide to applying the ISA-95 standard in manufacturing*. ISA, 2007.

[9] P. Leitão, V. Marik, and P. Vrba, "Past, present, and future of industrial agent applications," *Ind. Informatics, IEEE Trans.*, vol. 9, no. 4, pp. 2360–2372, 2013.

[10] W. L. B, A. Lobato-jimenez, and E. Axinia, "A Survey on Standards and Ontologies for Process Automation," vol. 9266, pp. 22–32, 2015.

[11] A. Rocha *et al.*, "An Agent Based Framework to Support Plug And Produce," *2015 IEEE 13th Int. Conf. Ind. Informatics*, pp. 1318–1323, 2015.

[12] G. Di Orio, A. Rocha, L. Ribeiro, and J. Barata, "The PRIME Semantic Language : Plug and Produce in Standard-based Manufacturing Production Systems," in *The International Conference on Flexible Automation and Intelligent Manufacturing 2015 (FAIM'15), At University of Wolverhampton, UK*, 2015, no. JUNE.

[13] I. E. Commission, "IEC 62264 Enterprise-control system integration Parts 1-5." .

[14] I. E. commission, "IEC 61512 Batch Control Parts 1-4." .

[15] I. O. for Standardization, "{ISO} 15926 Integration of lifecycle data for process plants including oil and gas production facilities - Part 1 - Overview and fundamental principles." 2003.

[16] I. E. Commission, "{IEC} 62424 Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools." 2008.

[17] I. E. commission, "{IEC} 62714 Engineering data exchange format for use in industrial automation systems engineering - Automation markup language - Parts 1 and 2." .

[18] M. Schleipen, R. Drath, and O. Sauer, "The system-independent data exchange format CAEX for supporting an automatic configuration of a production monitoring and control system," in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, 2008, pp. 1786–1791.

[19] S. Faltinski, O. Niggemann, N. Moriz, and A. Mankowski, "AutomationML: From data exchange to system planning and simulation," in *Industrial Technology (ICIT), 2012 IEEE*

International Conference on, 2012, pp. 378–383.

[20] Mtc. Institute, “MTConnect Standard Parts 1-4.” 2014.

[21] M. Parreira-Rocha, A. Grilo, and J. Jassbi, “Risk of employing an evolvable production system,” 2015.

Appendix

I. List of Interface Functions and Messages

Outbound	Inbound	Function Name	Function Description	Input Message	Output Message
T[5]	M	getValue	Gets value from M	{operationTag: string, samples: int, processingTime: int, energyComsuption: int}	{values: Collection<PMLValue>}
T	M	setValue	Sets value to M	{operationTag: String, value: PMLValue}	{response: Boolean}
T	T	getValue	Gets values from other T or DB	{componentTag: string, samples: int}	{values: Collection<PMLValue>}
T	T	getSimulation	Gets simulations, considering the provided scenarios	{scenarios: Collection<PMLConfguration>}	{simulations: Collection<PMLSimulationResult>}
T	D	initializeHWConnection	Initializes hardware connection	-	{response: Boolean}
T	D	closeHWConnection	Closes hardware connection	-	{response: Boolean}
T	D	getValue	Gets value from device, PLC, robot	*device*	{values: Collection<PMLValue>}
T	D	setValue	Sets value to device, PLC, robot	{componentTag: String, value: PMLValue}	{response: Boolean}

[5] Tool (T), Middleware (M), Device, PC, PLC, Sensor, etc. (D)

Outbound	**Inbound**	**Function Name**	**Function Description**	**Input Message**	**Output Message**
T	T	TBD	Sending data from analytic T to visualization T	{“name_station”: “int”, “name_component”: “int”, “state_variable: “int”}	{“name_station”: “int”, “name_component”: “int”, “state_variable: “int”, “reference_state”: “int”}
D	T	TBD	Sending data from SmartFactory Cube to visualization T	{“name_station”: “int”, “name_component”: “int”, “state_variable: “int”, “value”: “double”}	-
T	D	getStatus	Gets machine status at specified (historic) time	{“timestamp”: “time”, “Pidentifier” }	{“PIdentifier”: “string”, “value”: “string”, “type” : “string”}
T	T	GetSimulation	Start the simulation with a given list of possible scheduling plans and return a Ranking List including Scores / Final KPI´s for each Manufacturing schedule	{“schedulingplan”: “complextype”, “simulationEngineID” : “int” }	{“Score”: “complextype” }
D	T	Regression	An statistical analysis to model the relations between variables which helps for predicting failure	*Alarms id, Alarms description, alarm time; Maintenance time, maintenance reason*	*A failure probability for every three machine in Siemens*
T	T	getSimEnvStatus	Asks the SimEnv status	-	ENUM{"SIMULATING - BUSY" , "IDLE - READY" , "ERROR - UNAVAILABLE" , "…"}

Outbound	Inbound	Function Name	Function Description	Input Message	Output Message
	T	abortSimulation	Stops executing any simulation and returns to IDLE state	-	ENUM{"ABORT SUCCESSFUL" ,"ERROR - UNAVAILABLE", "ERROR - NOT SIMULATING" , …"}
T	T	startSimulation	Tells the Sim Env to obtain all necessary information to run a simulation, and then to execute;	ENUM{"SINGLE", "MONTE CARLO", "OPTIMIZATION", "…"}; *date and time at which to start and to stop simulation;* ID of the hierarchical elements; Reference to a (manually) preconfigured simulation task; list of the particular results requested, in a specified order ENUM{"COMBINED OEE", "TOTAL THROUGHPUT", "ENERGY CONSUMPTION", "DELIVERY OVERRUN", "ROBUSTNESS SCORE", "…"} or a PMLConfiguration; The ID of the services requesting the results of the simulation	ENUM{"REQUEST ACCEPTED","ERROR - UNAVAILABLE", "ERROR - BUSY", "…"}
T	T	returnSimulationResults	After the simulation has finished, returns the results to the service that called the simulation service	*the ID of the services where the results of the simulation should be sent; a list of the particular results requested, in the order requested by the startSimulation() request. All numerical results to be returned in text form.*	-
T	T	getAllEntityIDs	Asks the MW to return the list of all Machinery, workers, and other entities in the system	-	*list of IDs for all machines*

Outbound	**Inbound**	**Function Name**	**Function Description**	**Input Message**	**Output Message**
T	T	getFactoryTopology	Asks the connected entities for information about themselves	*The ID of the entity for which information is requested*	*Common name of the entity; entity kind ENUM{"PROCESSING", "ASSEMBLY", DISASSEMBLY", "RECEIVING", "SHIPPING", "…"}; entity skills; entity values and skills; entity location*
T	T	getProductOrders	Asks the MW to deliver the set of Product Orders	-	*the list of all ordered PMLProducts, along with their respective attributes*
T	T	getValue	Gets the value of a particular parameter of a particular entity for a particular time range	*The ID of the entity for which information is requested*	*requested value*
D	T	Observe	Decides the status of the machine based on the current sensors observation and history	{"Sensor 1 data": double", "Sensor 2 data": double", …,"Sensor N data": double",	{"Machine Status":'string',Value: 'Normal', 'Faulty','Failure}
T	T	Read_MS	Reads the current state of the machine	{"Machine Status":'string',Value: 'Normal', 'Faulty','Failure}	{'Fault Signature': 'array of boolean'}
T	T	Find_faulty_element	Predict the root causes of the fault	{'Fault Signature': 'array of boolean'}	{'elements (1-N)': 'string', Value:'Probability' }

Outbound	Inbound	Function Name	Function Description	Input Message	Output Message
D	T	TBD	Sends data from sensor to data/router/data analytic module	{“name": “int”, “value”: “double”, “timestamp”: “decimal(13,3)”}	{“name_station”: “int”, “name_component”: “int”, “state_variable: “int”}