Ecosystem for COllaborative Manufacturing PrOceSses – Intra- and Interfactory Integration and AutomaTION
(Grant Agreement No 723145)

D3.3 Digital Factory Model II

Date: 2018-10-31

Version 1.0

Published by the COMPOSITION Consortium

Dissemination Level: Public

Co-funded by the European Union’s Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement No 723145

Document control page

Document file:	D3.3 - Digital Factory Model II - v1.0
Document version:	1.0
Document owner:	CERTH
Work package:	WP3 – Manufacturing Modelling and Simulation
Task:	T3.2 – Integrated Digital Factory Models
Deliverable type:	R
Document status:	☒ Approved by the document owner for internal review ☒ Approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Alexandros Nizamis, Vagia Rousopoulou (CERTH)	2018-09-28	Table of Contents
0.2	Dimosthenis Ioannidis, Thanasis Vafeiadis (CERTH), Farshid Tavakolizadeh (FIT), Paolo Vergori (ISMB), Charisi Vasiliki (ATL)	2018-10-17	Chapter 4 updates
0.3	Alexandros Nizamis, Vagia Rousopoulou (CERTH)	2018-10-18	Chapter 6 updates related to DFM schema
0.4	Alexandros Nizamis (CERTH)	2018-10-21	Chapter 7 updates related to DFM API
0.5	Alexandros Nizamis (CERTH)	2018-10-23	Document ready for internal peer review
0.6	Alexandros Nizamis (CERTH), Charisi Vasiliki (ATL)	2018-10-30	Document corrections after peer review process
1.0	Alexandros Nizamis (CERTH)	2018-10-31	Ready for submission to EC

Internal review history:

Reviewed by	Date	Summary of comments
Nadir Raimondo, Paolo Vergori (ISMB)	29/10/2018	Contents are in place and methodology is sound. Chapter 4.2.6 needs to be aligned quality-wise. Minor comments in the text.
Nacho González (ATOS)	26/10/2018	A very high quality document. Addresses in a very comprehensive way the topics stated in the DoW and in the objectives of the document. Correctly formatted using the official COMPOSITION templates and quality guidelines. All elements well captioned. No grammar nor orthography errors

Index:

1 Executive Summary 4
2 Abbreviations and Acronyms 5
3 Introduction 6
3.1 Purpose, context and scope of this deliverable 6
3.2 Content and structure of this deliverable 6
4 Digital Factory Model in COMPOSITION Overall Architecture 7
4.1 Overview 7
4.2 Basic Interactions 8
4.2.1 Business Process Models 8
4.2.2 Intra-factory Interoperability 9
4.2.3 LinkSmart Service Catalog 9
4.2.4 Simulation and forecasting tool 10
4.2.5 Deep Learning Toolkit 10
4.2.6 Decision Support System 12
5 State of the Art Analysis of Existing Standards 16
5.1 Formats for data exchange 16
5.2 Business to Manufacturing Mark-up Language (B2MML) 16
5.3 MIMOSA 16
5.4 Green Building XML (gbXML) 17
5.5 Business Process Model and Notation (BPMN) 17
5.6 Open Geospatial Consortium (OGC) 17
5.6.1 Sensor Model Language (SensorML) 17
5.6.2 Observations and Measurement 17
5.6.3 OGC SensorThings 18
5.7 Extenxible3D 18
5.8 Predictive Model Markup Language (PMML) 18
6 Digital Factory Model Schema 19
6.1 Design Methodology 19
6.2 Specifications of Digital Factory Model Schema 20
6.2.1 Factory ID 21
6.2.2 Information Model 21
6.2.3 Events 28
7 Digital Factory Model API 31
7.1.1 Methodology and Implementation Technologies 31
7.1.2 Supported Interfaces 34
7.1.3 Deployment and Security 39
8. Conclusions and Future Work 41
9. List of Figures and Tables 42
9.2 Figures 42
9.3 Tables 42
10. References 43

1 Executive Summary

This deliverable presents the results of Task 3.2 Integrated Digital Factory Models and it is the second and final version of D3.2 Digital Factory Model I (M15). It aims to describe and analyse the implemented COMPOSITION Digital Factory Model (DFM). More precisely, in this report both the implemented DFM schema and the developed DFM Web API are described. The DFM design and the work has done in Task 3.2 is driven by COMPOSITION project use cases, requirements and WP3-Manufacturing Modelling and Simulation activities.

Information related to manufacturing systems, available factory processes, factory resources and real time events from the analytics tools should be available to many COMPOSITION components. Unfortunately, the previous mentioned information is acquired from various heterogeneous sources in the factory. In order to be understandable from all project's components they should be in a common format. A model can be used to capture and represent information from the factory in a common format available to all related components.

The Digital Factory Model are used as the aforementioned model. DFM will provide the digitalization of industrial aspects as a common model of information elements. The DFM schema is used as the common format for information exchange between the IIMS components.

During the design phase of COMPOSITION DFM, the knowledge gained at SatisFactory EU Project (SatisFactory, 2015) was adopted and extended. As both projects are related to manufacturing domain and the representation of manufacturing resources many standards were used at Common Interface Data Exchange Model from Satisfactory were used also at COMPOSITION's DFM. Besides that, other standards and structures were incorporated at DFM in order to cover this project's requirements.

This document provides a thorough analysis of the Digital Factory Model. Besides purpose, context, and scope the first part of this document is devoted to the, content and structure of this deliverable. The next parts describe both the well-known standards were analysed and some of them used in DFM's design and implementation. Besides the DFM schema and structure, a DFM API is also developed and presented in this document. It offers a wide variety of web services related to storing/retrieving data to/from a common database in a format validate against DFM schema.

This report describes the work has done from M3 (Task 3.2 starts) to M26 (date of this deliverable) and represents almost the final outcome of this task that ends at M28.

2 Abbreviations and Acronyms

Table 1: Abbreviations and acronyms are used in this deliverable

Acronym	Meaning
API	Application Programming Interface
BMS	Building Management System
B2MML	Business to Manufacturing Mark-up Language
BPMN	Business Process Model and Notation
DFM	Digital Factory Model
DLT	Deep Learning Toolkit
DSS	Decision Support System
gbXML	Green Building XML
JSON	JavaScript Object Notation
IIMS	Integrated Information Management System
IoT	Internet of Things
ISO	International Organization for Standardization
LA	Learning Agent
MQTT	Message Queue Telemetry Transport
OGC	Open Geospatial Consortium
OGI	Oil and Gas Interoperability
PMML	Predictive Model Markup Language
SensorML	Sensor Model Language
SFT	Simulation and Forecasting Tool
SWE	Sensor Web Enablement
WP	Working Package
X3D	eXtensible 3D
XML	eXtensible Markup Language
XMI	XML Metadata Interchange
XSD	XML Schema Definition

3 Introduction

3.1 Purpose, context and scope of this deliverable

The purpose of Task 3.2 Integrated Digital Factory Models and its corresponding deliverables is to establish a common methodology, and respective notations, for modelling processes. In this task Digital Factory Models instances will be created for both BSL and KLE as these are the industrial partners of the project. The processes, the resources and the events related to the project's use cases will be modelled in a common format available to other IIMS components. The scope of this deliverable is to describe the work that has been done for Task 3.2 and to present the current version of DFM schema. It further describes the DFM API which offers services for storing/retrieving data to/from a common database.

This report is the second version of D3.2 Digital Factory Model I. In this second version the approach that was followed was to update the first version and keep all the information from first version that were needed in order to create a coherent document that describes Task 3.2 from its beginning. Besides the information coming from the first version, this version presents the updates and extensions in Task 3.2. The update of the DFM schema in parts of assets and sensors schemas and the creation of a Resource Catalog for the rest of IIMS components is presented in this version. Moreover, the replacement of the XML events schema with a JSON one is presented. All these schema updates lead to web services updates and extensions which are also drawn in this document. Furthermore, the deployment of the component based on project's architecture and the alignment with the security framework have been implemented and documented in this report.

As this deliverable comes on M26 just two months before the end of the task (M28), both the DFM schema and API versions are near to the final ones. Maybe some small modifications will be done within the next two months until the end of the task. Also until the project completion if it is necessary for the overall COMPOSITION project some modification will be done as some components which interact with DFM will be delivered later.

3.2 Content and structure of this deliverable

In this deliverable the COMPOSITION's Digital Factory Model is presented. A DFM API and its supported services are described too. In order to properly describe the specifications of these components we decided to include the following basic sections in this report:

Section 4 describes the integration of the DFM component with the overall COMPOSITION architecture and its interactions with other COMPOSITION components. Special attention is given to the integration of BPMN diagrams from Task 3.1-Process Modelling and Monitoring Framework, and to the interactions with the Simulation and forecasting tool, the Decision Support System and the Deep learning toolkit.

Section 5 includes a brief state-of-the-art analysis of the existing standards related to industry domain and common data definition.

Section 6 presents and analyses the implemented DFM schema. This section specifies the basic elements of DFM and the corresponding XSD and JSON schemas.

Section 7 is about the DFM API and its services. The architecture, the specifications and current supported interfaces of API are presented. Actually, this section and Section 6 describe the basic components and results of this deliverable.

Section 8 is the conclusions section which sums up this deliverable's outcomes.

4 Digital Factory Model in COMPOSITION Overall Architecture

4.1 Overview

The Digital Factory Model is a core component of COMPOSITION Intra-factory system. The DFM enables the digitalization of industrial aspects. Both static and dynamic data which are provided from different system's parts in a heterogeneous format are described in a common format using DFM schema. This means that all the data are modelled and provided with the same format to all related components. By using the DFM API all the data are stored in a common database. Of course, the stored data in the database is validated against DFM format. The next figure presents the place of the DFM component in the COMPOSITION overall architecture based on D2.4-The COMPOSITION architecture specification II (COMPOSITION D2.4, 2018).

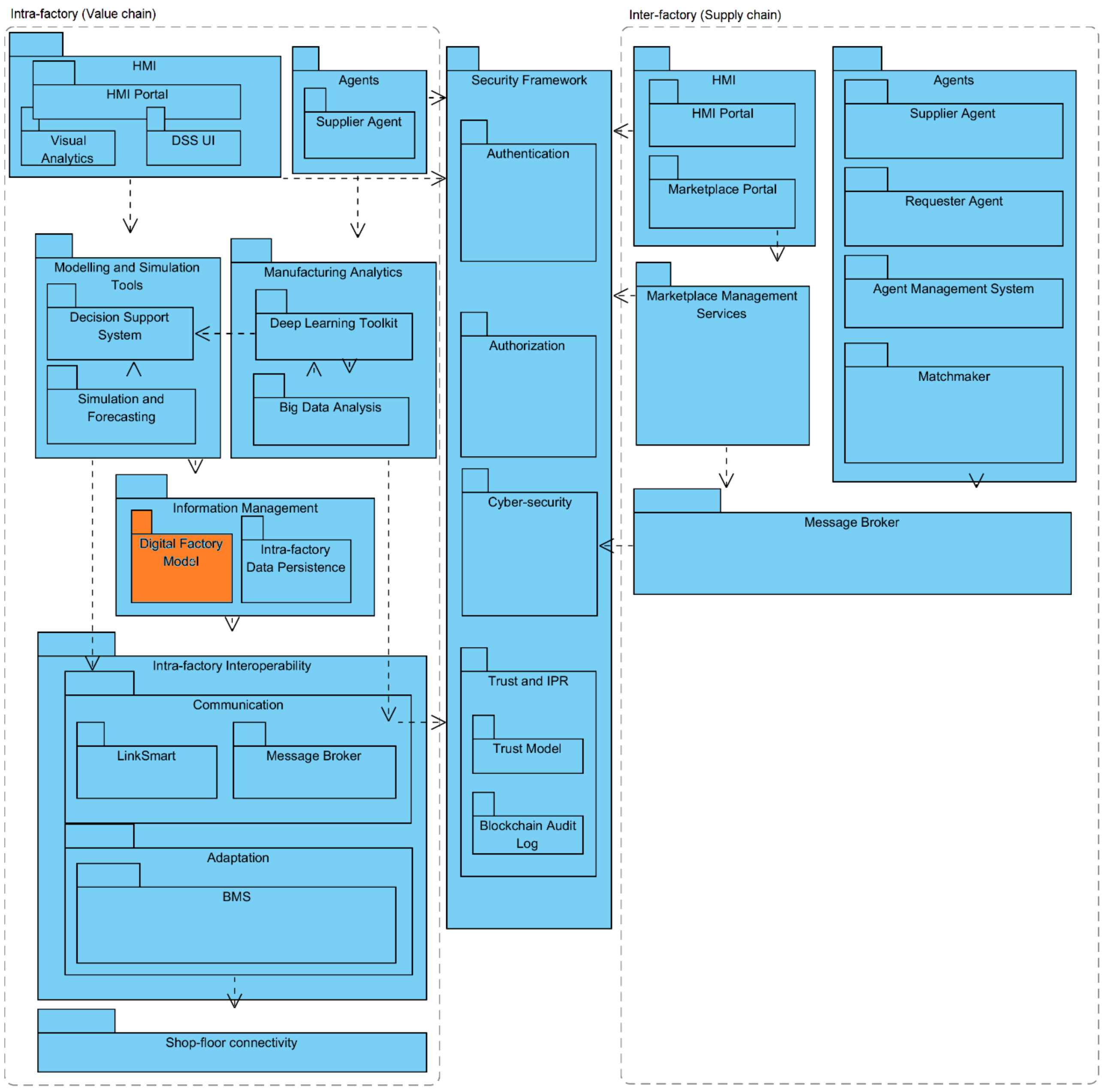


Figure 1: COMPOSITION Functional view

4.2 Basic Interactions

In order to explain better the position and the interactions of DFM, we offer a brief analysis of the components which are related to DFM with special focus given in their connections with it. We describe only components which will use DFM in cases such as the data format validation or the communication with the data store and not the intermediate components which just enable communication or ensure security such as the Event Broker and Security Framework as their role is more generic and related to all components. Their role will be described better and in more details in these components corresponding deliverables.

4.2.1 Business Process Models

In COMPOSITION, BPMN was chosen as the standard to model business process as well as manufacture process. Business Process Model and Notation (BPMN) is a standard for business process modelling that provides a graphical notation for specifying business processes in a Business Process Diagram (BPD). BPMN is based on a flowcharting technique. The objective of BPMN is to support business process management, for both technical and business users, by providing a notation that is intuitive to business users, yet able to represent complex process semantics.

There are several advantages of using BPMN to model business and manufacture processes:

- Standardization: BPMN provides not only standard for graphical notation, but also standard for representation in XMI. That means BPMN is not only easy to be interpreted consistently by people in different domains, but also easy to be interpreted by different software programs, as long as those programs follow the XMI standard. This feature makes BPMN a favourable choice in describing processes involving cross-domain knowledge.
- Simplicity and flexibility: BPMN offers very simple and intuitive notations for describing business processes. Even people who have limited knowledge in ICT can easily model and interpret BPMN. This makes BPMN very appealing to both business users and technical users. Despite its simplicity, BPMN also offers high flexibility which makes it suitable to model both simple and complex processes.
- Rapid development and integration: There are already many tools and libraries which make it easy to work with BPMN. These tools offer functionalities such as modelling Business Process Diagrams (BPD), executing BPMN logic and visualizing BPMN dynamically. With these tools, a real life process could be turned into BPMN and further integrated into software application logic in a very rapid and simple manner.

In the context of COMPOSITION, the BPMN technology will be exploited in the Process Modelling and Monitoring Framework (T3.1). The framework on one hand will monitor and process data coming from sensor networks, through the respective automation and control system (e.g. MES, SCADA), and other information management or computer aided tools. On the other hand, it will enrich data by annotating them with the active status of the process. To achieve this goal, relating these data to the specific steps or events in the BPMN systematically is necessary. For this purpose, a Deployment Model is needed. A Deployment Model describes the available machines, devices and sensors in a manufacture as well as the mapping of each of these available resources to a specific IoT channel, such as a MQTT topic. Each event in the BPMN will be configured to listen to a specific IoT channel specified in Deployment Model. This way, a real life sensor signal can be mapped to trigger a specific event in BPMN, making synchronization between the real process and the BPMN process possible. The BPMN process will then be deployed on the Process Modelling and Monitoring Framework, which runs on top of a BPMN Engine for BPMN logic execution. The Process Modelling and Monitoring Framework can then annotate data according to the current active status of the BPMN process.

As a result, the Process Modelling and Monitoring Framework are integrated with the DFM in a mutual way: on one hand, it complements a DFM instance by providing the BPMN for the process modelling. On the other hand, it relies on the DFM to provide the Deployment Model of available resources to correlate real-time data to the BPMN process.

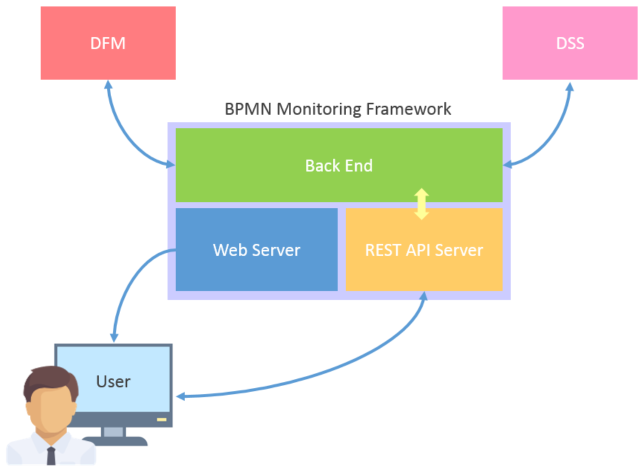


Figure 2: Structure of the BPMN Monitoring Framework Based on D3.1

4.2.2 Intra-factory Interoperability

LinkSmart Middleware

As depicted in the figure 1 the DFM component communicates with Intra-factory Interoperability layer and LinkSmart middleware. The LinkSmart will offer a Device Connector (Gateway) which will act as a gateway between the BMS's low level output and the COMPOSITION TCP/IP network. However, it was defined by the consortium that the BMS component includes similar functionality and the LinkSmart/Device Connector will not be used. So, the DFM will be connected only with the BMS component which will provide the low level data to COMPOSITION TCP/IP network. Besides that, two LinkSmart components: 1) LinkSmart Service Catalog with talks to all other services and 2) LinkSmart Learning Service which is related to the Big Data Analysis component will be used based on the updated system's architecture. The added LinkSmart Service Catalog is related with the DFM component in the way it is described in the next section.

Building Management System

The Building Management System or BMS is key component in the intra-factory interoperability layer. This component provides a model for interconnecting the COMPOSITION ecosystem within the shop floor acting as a translation layer. It provides connectivity from sensors to the IIMS components. Within this component, information is pervasively collected from any connected systems in order to support the management operators in making decisions and to take direct control for automation tasks. Since the BMS enables the connection with all the major automation standards, it will act as a bridge between the cyber-physical systems (sensors, gateways, etc.) and the other IIMS components.

The BMS component will offer services for storing and reading sensors data coming from shop-floors. All these data will be described in the format that will be defined by DFM schema. Furthermore, the IIMS components that are going to read data from BMS should receive the datastreams related to a factory resource from DFM in order to create the MQTT topics.

4.2.3 LinkSmart Service Catalog

The LinkSmart Service Catalog is an application offering service registration and discovery in IoT environments. A service is a network-connected application (message broker, database, web service, etc.) that provides functionalities to other components via designated protocols. Service Catalog offers a RESTful API which can be considered as the entry point for applications and other components of a COMPOSITION system. An entry in the catalog shall contain necessary information about a service such that other application can consume its resources. In the current design of the system, an entry describing Digital Factory Model must contain the unique ID, URL, and public key of a DFM service. This information is needed so that other components of the system can contact and also verify messages issued by this service.

We use zeroconf/DNS-SD to advertise the endpoint of the Service Catalog, such that applications and services in a local network can discover it without manual configuration. In a dockerized environment, the discovery of Service Catalog is possible through local DNS servers.

4.2.4 Simulation and forecasting tool

The Simulation and Forecasting Tool component is part of the high-level platform of COMPOSITION, the Integrated Information Management System (IIMS). Simulation and forecasting tool's main purpose is to simulate process models and provide forecasts of events whose actuals outcomes have not yet been observed.

In order to be able to provide predictions, the Simulation and forecasting tool should have access in both static and dynamic data. Many of static data are provided by the DFM component. The data related to actors and assets from the project's use cases are stored as DFM instances and SFT can query them as the rest of IIMS components. Furthermore, the real time data from sensory infrastructure are described in a common format based on DFM schema and the SFT is able to parse them from BMS in a common way. The Simulation and forecasting tool is able to get sensors data from BMS (using MQTT topics) by using the datastreams that are stored in the Resource Catalogue (it will be presented at chapter 6) for a factory stored in a DFM instance.

Moreover, the predictions coming from SFT are stored to the DFM instances as observations of upcoming events and be fed to DSS in order to enhance and automate the decision making in the factory. The prediction from SFT are stored using the DFM API services and the DSS is able to read them by using the services of the API as well.

4.2.5 Deep Learning Toolkit

The Deep Learning Toolkit is a component that founds its location in the intra-factory scenario and is a standalone component. Its functioning is be described in details in D5.4, but we can sum it up here for the sake of providing a broader background to this document.

The component is contained within a Docker image that runs in a perpetual activity at the shop floor level. As a component, it has many declinations based on its deployment, so for simplicity we will analyse the deployment concerning the use case BSL-2, related to the predictive maintenance scenario. In this use case, the Deep Learning Toolkit is expected to receive its inputs from a heterogeneous sensors array deployed in the shop floor, mitigated by the Building Management System, distributed by the Intra-factory Interoperability Layer and mediated by the Big Data Analytics. After all these steps, the final results should be a data array conformed to the historical dataset formed in the training phase.

In fact, the Deep Learning toolkit is composed by different phases that take place both offline and online. The only phase that is concerning this document is the Live phase named Continuous Learning, in which data formats are important and expected to be exchanged among other components. As literature matter of fact, the Artificial Neural Networks forming the Deep Learning Toolkit need to abide to the data consistency law for both historical and live data. Being these phases that threat the historical dataset an offline phase, it is then a logical deductible consequence that the live datasets must have the very same fields. The expected output is a dichotomy based scalar value that predicts whether there is going to be an allegedly event of breakdown possible failure, in the next time slot evaluated by the prevision timeframe.

The historical data provided by the Boston Scientific end user have been clustered in the Data Formatting and Pre-processing phase of the Deep Learning Toolkit component. Is not in scope of this document to analyse the technical choices that have been made on this data analytics step, but we can provide a small summary of them, by saying that the output of this phase is an unlimited data stream composed by a fixed number of features. These features are related to the machinery involved in the predictive maintenance scenario (BSL-2).

During the first iteration of the COMPOSITON project, they were representative of measured temperatures, expected temperatures and output powers relative to the Brady oven. Being these measures coming from sensors-based detections, they have been organized in data triplets related to measurement areas, resulting in sixty independent fields. In this iteration the OGC SensorThings was used by the Deep Learning Toolkit to homogenize its data to the projects' data formats receiving and providing information and previsions in the form of OGC Observation.

The standard defines three mandatory fields that need to be filled in, in order to be compliant. In the followings they are listed, alongside a brief description extracted from the OGC standards. (OGC standards, 2017)

Table 2: Observation's suggested fields for Deep Learning Toolkit

phenomenonTime	mandatory	Time(Interval) String (ISO 8601)
resultTime	mandatory	Time(Interval) String (ISO 8601)
result	mandatory	Any (depends on the observationType defined in the associated Datastream)

In the case of the Deep Learning Toolkit, the *result* field provided a more than a suitable candidate to host the component periodical previsions dispatched in retain mode on the intra-factory MQTT broker in reaction to incoming batches of live sensors readings.

In addition, two optional fields have been evaluated in order to enrich the information qualitative provisioning to third-party components. The two fields are represented in the following form the very same table grabbed from the standard and summarized in the followings.

Table 3: Observation's optional fields for Deep Learning Toolkit

resultQuality	optional	DQ_Element
validTime	optional	Time Interval String (ISO 8601)

The *resultQuality* field perfectly aligns with the abstract concept of accuracy information, dispatched with every partial result outputted by an Artificial Neural Network. In fact, every metric that is used in the Deep Learning Toolkit are a balanced measurement of errors and variations, aimed at providing a quantitative estimation of the divergence from the average at each of the performed steps. Each step is named epoch and provides its own metrics. The *resultQuality* field can be used to provide a comprehensive representation of the aggregated statistical goodness of these metrics.

The *validTime* optional filed is represented in the standard by a Time Interval String, and has been evaluated to define a time to live expectancy validity of the Deep Learning Toolkit prevision itself. In fact, like network packages that travel across communication networks, previsions that travel across the shop floor layer can be marked with this validity time, based on the quality of the results being therefore dispatchable to tier-one, tier-two and tier-n components with a decreasing time to live expectancy. In particular, this means that short term previsions might be dispatched to nearby components only and long lasting previsions could marked as suitable to be read by human eyes.

In order to provide a data readings array to the Deep Learning Toolkit, instead of a less scalable single reading observation for each of the aggregated sensors, several measurements that happen at the very same time has been aggregated. This is made possible, by defining data arrays instead of scalar and string values in the result field. A very simple example of an aggregated reading is provided:

```
{
        "result": [[120,140,80,...],[121,149,80,…],…]
        "phenomenonTime": "2017-01-01T00:00:00.000Z",
        “resultTime”: "2017-01-01T01:00:00.000Z",
}
```

The OGC format was also used to inform the Deep Learning Toolkit of any kind of warning or failure coming from the measured machinery, improving the Artificial Neural Network model. A simple example of these kinds of readings could be:

```
{

        "result": “0”,  “0” = ok, “1” = warning, “2” = error
        "phenomenonTime": "2017-01-01T00:00:00.000Z",
        “resultTime”: "2017-01-01T01:00:00.000Z"

}
```

During the second iteration of the project, the model input dataset has been updated by integrating newly features that further developments have made available. These changes have been to a large extent originated from by the switch from Brady to Rhythmia oven for the BSL-2 predictive maintenance scenario.

The base data described above has been adapted by filtering logs events provided by the oven (e.g. temperature warning, deviation alarm, etc.) and acoustic information registered nearby the oven fan. On the other hand, the original measured temperature value is no longer available on the Rhythmia oven so the sensors-based detections as decreased from sixty to forty values.

In order to be used in a supervised machine learning framework, these information has to be organized as a time series. Each element of these series is organized in a table as follow:

Table 4: Sample features

0	1	…	39	40	41	…	232	233	234	235	236	…	240	241

The leftmost part of the table (green, columns from 0 to 39) contains the values sampled from different temperature and output power sensors, measured at the same time. The central part of the table (red, columns from 40 to 235) contains the mapping of reflow events. The rightmost part of the table (blue, columns from 236 to 240) contains the acoustic data sensors readings in decibel. The last column (241) contains a label to identify the oven status to train the Artificial Neural Network.

During the first iteration of the component, information regarding each raw data stream from the shop floor (logs, sensors and acoustic data) were distributed through the Intra-Factory Management System. The Big Data Analytics module, now deployed as Learning Agent is responsible for the operational procedures involving the aggregation of the aforementioned data streams, in order to create a suitable mapping of this information to be interpreted by the Deep Learning Toolkit. The Deep Learning Toolkit returns previsions in reaction to incoming live samples, as expected.

Due to the peculiar nature of the exchanged data between the two components (rate, scope, usage, privacy, etc.), a one-to-one local communication strategy between the two modules has been introduced, in order to allow resources management optimization and logically separate local traffic from the intra-factory MQTT broker. In particular, the Big Data Analytics Complex-Event Machine Learning (CEML, 2018) framework has provided the communication functionalities needed to combine Complex-Event Processing (CEP) and Machine Learning (ML) applied to the IoT.

Based on the above assumptions, the choice of creating a separate network with a private connection between the two containers has been made, linking them in a standalone manner. This means that the final configuration of the IIMF framework allows communications to the Deep Learning Toolkit Python backend through the Pyro remote method invocation functionalities. Pyro automatically serialize the data exchanged, reducing the effort and resources necessary for the OGC encapsulation. All the security issues derived by exposing a remote object interface through Pyro have been resolved through the trusted isolated network. Results and previsions from the Deep Learning Toolkit are therefore provided only to the Learning Agent module that is therefore in charge to dispatching all the information to the designed containers autonomously.

Whereas all the above is true in the intra-factory scenario, in the inter-factory a similar approach has been made for communicating with the Agent, but is not in scope of this deliverable to dive into details that will be made properly available in D5.4 due at M30.

4.2.6 Decision Support System

Decision Support System is an integrated system developed for the intra factory scenarios. It is a component of the COMPOSITION ecosystem, but it also is a standalone system, which can be used in collaboration with CMMS software. DSS is thoroughly described at D3.5, and a small abstract is given here to support the background of the deliverable.

The main purpose of the collaboration between the DSS and the DFM the use of the DFM data schema by the DSS, providing uniform data from different sources. Various data inputs are formatted according to the schema, before processing. Uniformity helps DSS to treat all incoming data the same way, which leading to quicker computation and better decision results. The Digital Factory Model provides an ontology with all necessary entities in a common format. The Digital Factory Model can describe all the information related to a factory such as buildings, assets, actors, processes and live events.

Due to the nature of the decision – making process, the need to explicitly model the context of a decision support system output and the DFM schema can be reused for the output data. DSS and its feedback loop should also provide and communicate the data based on the DFM data schema. Compliance to the schema allows data consistency and ability for reuse.

Each DFM instance represents a different shop floor. IIMS components can build DFM instances or read data from these instances by using the DFM API services which is based on RESTful services and HTTP protocol. DSS uses the DFM API services to retrieve all the static information about the shop floor, built upon well – known standards. Information about actors, assets and equipment are described using B2MML standard.

Moreover, the BPMN diagrams from Task 3.1 are available to the DSS via the DFM API. The DSS can get the corresponding to a shop floor instance, BPMN diagrams by calling the appropriate service for Business Processes. The Business Process List element of the DFM was covered by OMG's BPMN XML package. This package provides schemas which offer all the necessary means for the representation of a BPMN diagram in a DFM instance.

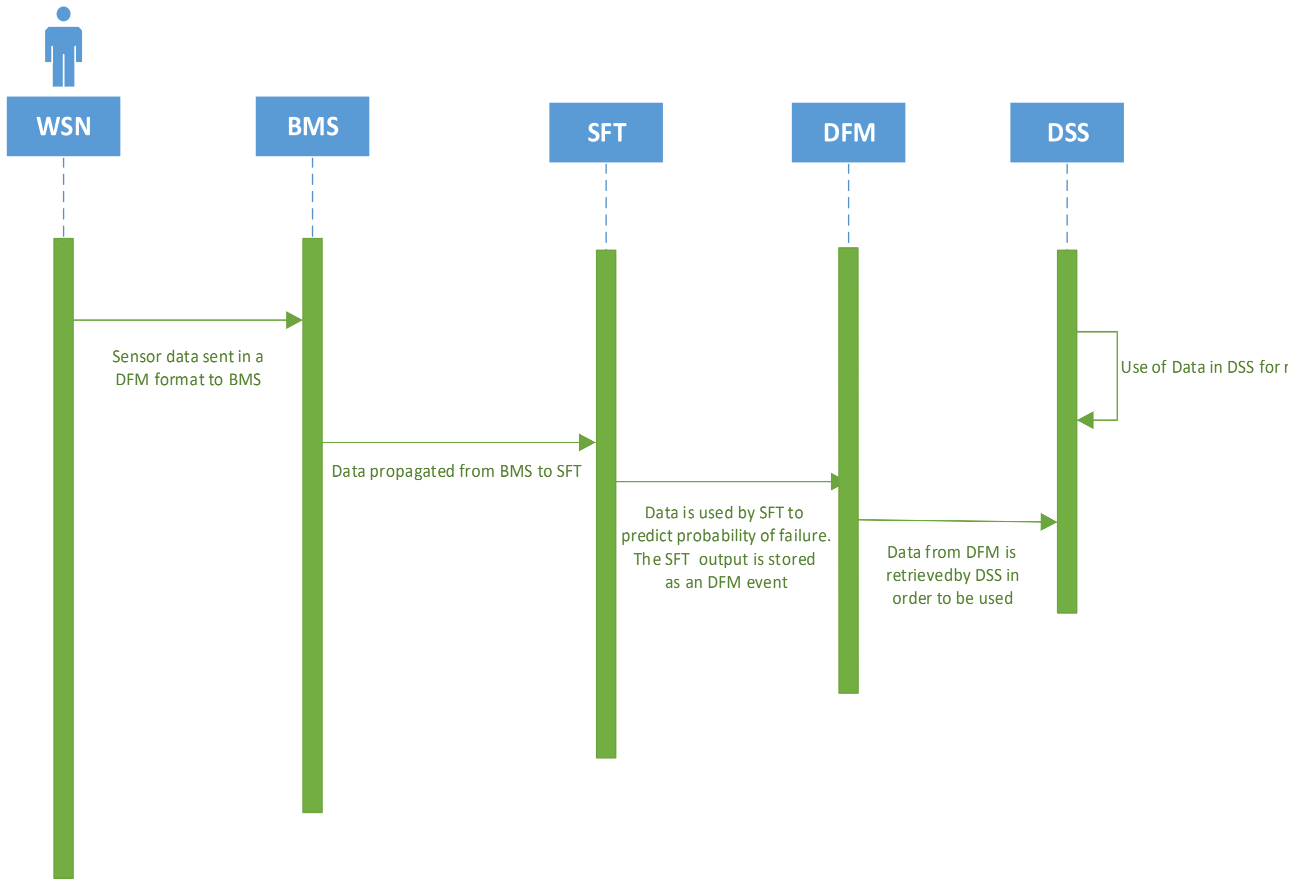

WSN
BMS
SFT
DFM
DSS
Sensor data sent in a DFM format to BMS
Data propagated from BMS to SFT
Data is used by SFT to predict probability of failure. The SFT output is stored as an DFM event
Data from DFM is retrievedby DSS in order to be used
Use of Data in DSS for r

Figure 3: Sequence Diagram for the data route in with DFM and SFT COMPOSITION Components

The sequence diagram which provides the route the data has to follow in order to reach DSS from SFT and DFM components. The basic logic behind the sequence diagram is that the data is gathered from WSN and is sent to DFM. If data is compliant with the DFM model, it can be processed from all the rest COMPOSITION components. This data is propagated to SFT, where they are transformed from sensorial data to meaningful data (e.g. probability of failure of the BOSSI polishing machine on KLEEMANN shop floor). The transformed data is sent to DSS through the exchange protocols, previously discussed. DSS uses and transforms the data in the rule engine, according to internal processes and creates rules and KPIs based on it.

DSS uses DFM for its procedures. For each shop floor, a FactoryID is appointed by the DFM, as well as both DFM's main classes are used to describe the DSS entities. Information Model is used to describe actors (personnel), tasks and assets on the DSS (the schema and classes are described in more details at chapter 6 of the document). For example, the personnel in KLEEMANN's instance is based on the Actor class of the DFM. Below, the DFM schema for the class and the actual entity of the personnel is shown.

There is an Actor ID which is hidden in the view, which corresponds with the DFM schema. The name and surname are sub-classes of the Person class, while the Skill corresponds with the Description. In the Personnel Information class, actor's phone and email can be located. Although not all sub-classes and elements are used for the personnel in KLEEMANN's example, the main structure is based on the DFM schema. The same logic is followed for BSL shop floor and for the rest of the classes of the Information Model, such as assets and tasks. Ease of communication between the two components is based on the existence of the common class.

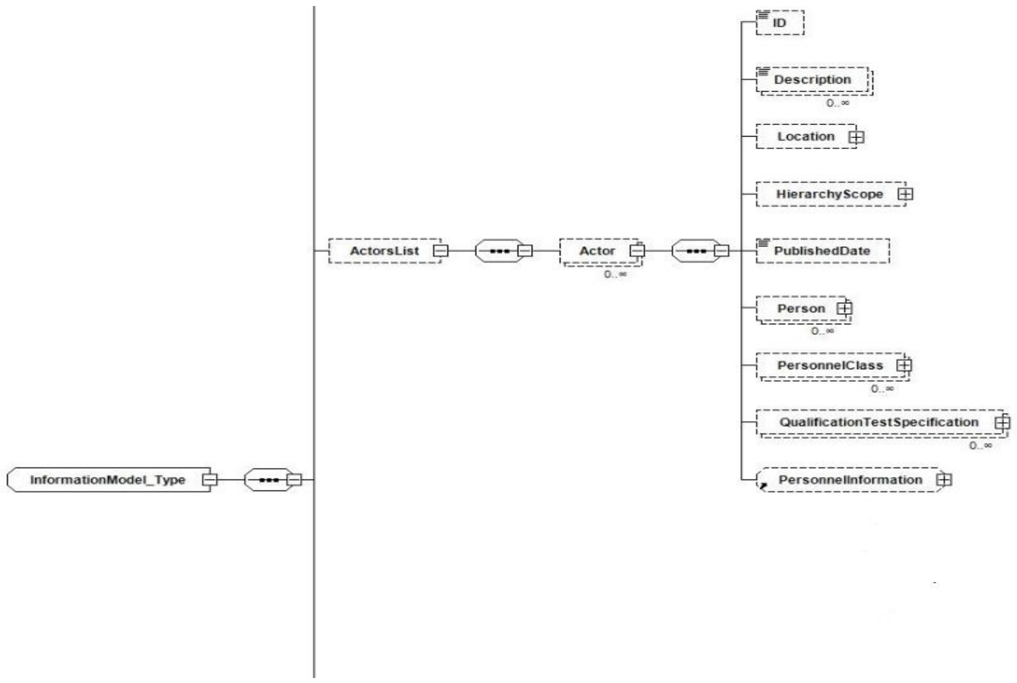
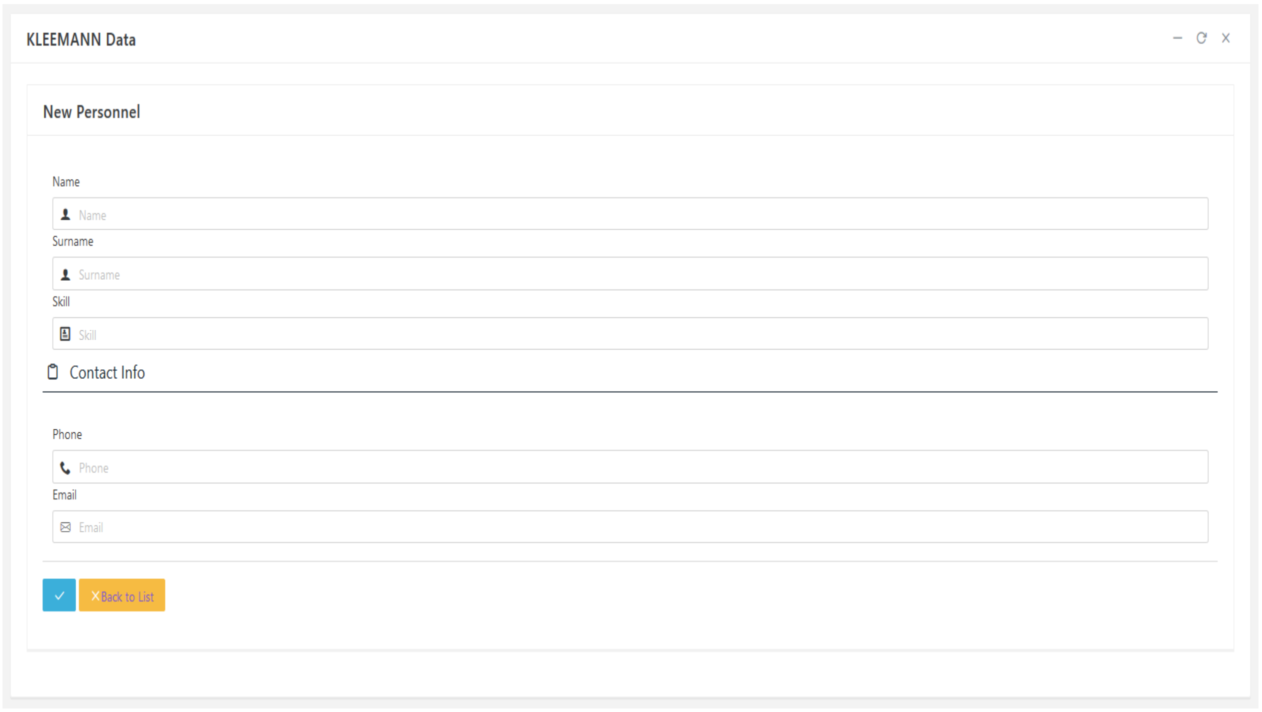


Figure 4: DFM Schema for Actor class


KLEEMANN Data

New Personnel

Name

Name

Surname

Surname

Skill

Skill

Contact Info

Phone

Phone

Email

Email

Figure 5: Personnel card corresponding to the Actor class of the DFM schema

Accessing data from the DSS, is based on the Events Type class. Incoming data is mapped as events on the DFM. Most of it contains IDs, types, timestamps, sources, locations, measurements, values etc. Not all Events Type sub-classes are used, but all incoming data complies with the schema. Agreement on the DFM schema is prerequisite for data conformance and persistence.

5 State of the Art Analysis of Existing Standards

During the first stages of the Task 3.2 a thorough analysis of the existing well-known standards related to manufacturing, sensors, buildings and process modelling has been conducted. This chapter describes in short the results of this analysis by presenting the standards which selected as the best candidates to be used in DFM schema's implementation. Many of them finally were selected for the DFM design. Before introducing the state of the art review, worthy of mention are languages and formats used by existing standards of Digital Factory modelling.

5.1 Formats for data exchange

Extensible Markup Language or **XML** is an adjustable text format that defines a set of rules for encoding documents that can be read both by human and machines. XML enables the exchange of a wide variety of richly structure data across the Web. XML is a recommendation from the World Wide Web Consortium (W3C). This is similar to Hypertext Markup Language (HTML). XML contains mark-up symbols to describe page or file contents. XML is extensible and self-descriptive.

XML Schema is commonly known as XML Schema Definition (XSD). It is a World Wide Web Consortium (W3C) recommendation used to describe and validate the structure and the content of XML data. XML Schema is meant to play a basic role in XML processing, especially in Web services where it serves as one of the axioms that higher levels of abstraction are built upon. It is extensible, new elements can be derived from existing elements. XML Schema does not require intermediate processing by a parser. It supports data types as well as default values. XML Schema supports namespaces, so that different XML vocabularies can be included within a document.

JavaScript Object Notation or **JSON** is a syntax for storing and exchanging data. It is a text format that is completely language independent. JSON resembles JavaScript object literal syntax. It can be used independently from JavaScript and many programming environments feature the ability to read (parse) and generate JSON. The two main structures of JSON are a collection of name/value pairs and an ordered list of values.

5.2 Business to Manufacturing Mark-up Language (B2MML)

Business to Manufacturing Mark-up Language or B2MML (B2MML, 2017) is an XML implementation of the ANSI/ISA-95, Enterprise-Control System Integration. It belongs to family of standards (ISA-95), known as IEC/ISO 62264. B2MML consists of a set of XML schemas. They are written using the W3C's XML Schema language (XSD) that implement the data models in the ISA-95 standard. B2MML enables the integration of business systems, such as enterprise resource planning (ERP) and supply chain management systems with manufacturing systems, such as control systems and manufacturing execution systems (MES). B2MML extends XML to set up a grammar, a message structure that contributes in manufacturing and business systems communication.

5.3 MIMOSA

MIMOSA (MIMOSA, 2017) is working on the adoption of open information standards for Operations and Maintenance in manufacturing, fleet, and facility environments. MIMOSA is a not-for-profit trade association focuses on enabling industry solutions leveraging supplier neutral, open standards, to establish an interoperable industrial ecosystem for Commercial Off-The-Shelf (COTS) solutions components provided by major industry suppliers. In order to achieve this, MIMOSA has promoted the development of the Oil and Gas Interoperability (OGI) Solutions Process™. OGI Ecosystem™ is a true supplier neutral solutions environment enabling a radical change towards a solutions reserving lower cost, faster implementations and improved quality. The basis of OGI Ecosystem is instituted by the following elements: the OGI Pilot™, the OGI Solutions Architecture™ and the ISO OGI Technical Specification.

OGI Solutions Process leverages existing standards, use cases specifications and methods in order to assist in solving business problems. The premise of MIMOSA is that major productivity gains critical physical infrastructure design, build, operate and maintain depend on transitioning to an interoperable, componentized architecture with shared supplier-neutral industry information models, information and utility services.

MIMOSA focuses on physical asset (plants, platforms and facilities) life-cycle management and infrastructure Operations and Maintenance. It develops and publishes industry-driven standards in alignment with ISO. MIMOSA association works closely with formal standards bodies to contribute in the development of international standards reflecting industry requirements.

5.4 Green Building XML (gbXML)

Green Building XML or gbXML (GbXML, 2017) is an industry supporting schema that allows various 3D building information models (BIM) and engineering analysis software to share information with each other. GbXML benefit the building industry providing solutions for the certification, design, maintenance, operation, and recycling of building information models. GbXML eliminates the need to re-enter the same information over and over again into multiple analysis tools. It uses XML so that major industry applications can import and export project information and data no matter the device, the vendor or the software platform.

5.5 Business Process Model and Notation (BPMN)

Business Process Model and Notation or BPMN (BPMN, 2017) is a standard for business process modelling. This standard aims provide businesses with the capability of understanding their internal business procedures. It offers a graphical notation and will offer to organizations the capability to communicate these procedures in a standard manner. A business processes is specified in a Business Process Diagram (BPD), based on a flowcharting technique very similar to activity diagrams from Unified Modelling Language (UML). The objective of BPMN is to support business process management, for both technical users and business users. It is achieved by providing a notation that is intuitive to business users. A mapping between the graphics of the notation and the underlying constructs of execution languages, particularly Business Process Execution Language (BPEL) is provided by BPMN specification.

A standard notation readily understandable by all business stakeholders is the focus of BPMN. The stakeholders can be the business analysts who create the processes, the technical staff responsible for implementing them, and their business managers as well. Consequently, BPMN is a common language, bridging the communication gap between business process design and implementation.

5.6 Open Geospatial Consortium (OGC)

The Open Geospatial Consortium or OGC (OGC, 2017) is an international industry consortium engaged with providing quality open standards for global geospatial community. Different types of organizations such as government agencies, universities and business entities participate in OGC consensus process. It aims to develop publicly available interface standards. OGC Standards support interoperable solutions that "geo-enable" the Web, wireless and location-based services and mainstream IT, and used in various domains such as Environment, Health, Agriculture, Meteorology and Sustainable Development.

5.6.1 Sensor Model Language (SensorML)

Sensor Model Language or SensorML (SensorML, 2017) is an OGC standard. This standard provides standard models and an XML encoding for describing process of measurement by sensors and instructions for deriving higher-level information from observations. A wide range of sensors can be described by SensorML.

A process described using SensorML is discoverable and executable. The definition of methods, parameters, inputs and outputs and the provision of metadata are main characteristics of these processes.

In order to provide a robust way of defining processes and processing components, SensorML includes sensors, actuators and computational processes applied pre- and post- measurement. The main intention is to achieve interoperability at syntactic and semantic level, so that sensors and processes can be embedded on different machines, used on compound workflows and be shared among web nodes.

5.6.2 Observations and Measurement

Observations and Measurements (O&M, 2018) is an international OGC Standard. This standard is an XML implementation for the OGC and ISO Observations and Measurements (O&M) conceptual model and it includes a schema for Sampling Features. The origins of the O&M are in the Sensor Web Enablement (SWE) initiative of the OGC. O&M provides a system-independent, Internet-enabled ways of data exchange between different parts of sensor networks and other systems using the captured sensor information. XML schemas for observations and features involved in sampling when making observations are defined by OGC O&M standard. These enables the exchange of information describing observation acts and their results using common document models. Different technical and scientific communities are able to use these models.

5.6.3 OGC SensorThings

SensorThings (SensorThings, 2017) is another OGC standard. It offers an open, geospatial enabled, unified way to interconnect Internet of Things (IoT) devices, data and applications over the Web. SensorThings is a platform-independent, non-proprietary and perpetual royalty-free standard. It builds on open standards, such as the Web protocols and the OGC Sensor Web Enablement (SWE) standards, including the ISO/OGC Observation and Measurement data model.

SensorThings includes two basic functionalities and each function is handled by a profile, the Sensing and the Tasking Profile. The Sensing Profile provides a standard way to manage observations and metadata from diverse IoT sensor systems. There are similarities with the OGC Sensor Observation Service. Tasking Profile provides a standard way for parameterizing IoT devices. It provides functions similar to the OGC Sensor Planning Service. The main difference between the SensorThings API and OGC Sensor Observation Service and Sensor Planning Service is that the SensorThings API is designed specifically for the resource-constrained IoT devices and the Web developers.

5.7 Extenxible3D

Extensible3D or X3D (X3D, 2017) is an ISO ratified and royalty-free open standards file format and runtime architecture for 3D scenes and objects representation and communication. ISO X3D standard is a successor to Virtual Reality Modelling Language (VRML). X3D provides a system with an open architecture, to store, retrieve and reproduce real time 3D scenes, covering various domains and user scenarios.

X3D acts as a central hub that can route 3D model information between diverse 3D applications. It provides open and non-proprietary tools to read and write geometric data and metadata. X3D makes X3D players available over all platforms in order to visualize data.

5.8 Predictive Model Markup Language (PMML)

Predictive Model Markup Language or PMML (PMML, 2017) is a standard used to describe predictive solutions. The Data Mining Group (DMG), a consortium of commercial and open-source data mining companies is responsible for the development of PMML. It is a leading standard for statistical and data mining models. It supports sharing predictive analytic models among different applications.

A model expressed in PMML can be trained in one system and be deployed to another. PMML is the standard that the foremost data mining tools can import and export. PMML offers interoperability between different statistical and data mining tools. By using PMML they are able to communicate in the same language. In this way, the need for custom coding is avoided and a predictive solution can be easily moved among different applications and tools. The companies are able to use first-rate tools to build the best possible solutions as PMML works as the common denominator.

6 Digital Factory Model Schema

At this section we describe the specifications of COMPOSITION's Digital Factory Model. Besides the detailed presentation of DFM's specifications, we provide an analysis of the methodology was followed during the design process.

6.1 Design Methodology

The Digital Factory Model of COMPOSITION was designed with the aim to able to:

- Describe in a common format, the data coming from heterogeneous resources with heterogeneous formats
- Define this common format which will be accepted by the project's components

The methodological approach during the design phase of DFM was driven by the two main aforementioned goals. Besides that, there were more factors which lead the design process. They are related to the analysis of existing standards, the analysis of project's requirements and use cases, and the analysis of the COMPOSITION architecture. Based on these factors, the design process followed the next set of steps:

1. *The analysis of the existing standards*. Actually, this was the first phase of the Task 3.2 and it was the initial step of the design process. During the initial stage of the Task 3.2 a thorough analysis of the relevant approaches and standards has been conducted. The basic existing standards for manufacturing related data representation were analysed. Furthermore, related tasks from other EU projects were studied. The analysis of the existing standards is presented in the previous chapter of this report
2. *The analysis of the requirements*. This was the second step of the DFM's design process. The requirements about modelling as they are reported at D2.2 Initial requirements specification (COMPOSITION D2.2, 2017) and updated at D2.5 Lessons Learned and Updated Requirements Report I (COMPOSITION D2.5, 2018) were analysed. The main requirements related to modelling tasks and DFM as well are presented to the below table.

Table 5: Main modelling requirements

Requirement Number	Title	Short Description
COM-14	A common methodology and notation for modelling shall be established	Modelling of processes and stakeholders is needed by most of the forecasting, simulation and decision support features of COMPOSITION. So, a common methodology for modelling should be established
COM-15	The processes and stakeholders of the pilots shall be modelled	All processes and stakeholders of the pilots, which are relevant for the forecasting, simulation and decision support features of COMPOSITION, are modelled
COM-67	Business processes must be described using the BPMN standard	Business processes must be defined in machine-readable format that supports also easy visualization
COM-69	COMPOSITION DFM has to be multi-scaled	The COMPOSITION DFM has to be multi-scaled in order to cover the following levels: machine-level, end-user-level, and process-level
COM-149	COMPOSITION sensors' data should be described using common formats	The real time data should be exchanged in a common format in order to be available for all related components Description of real time data in a common format using a standards and well known formats such as JSON or XML

3. *The analysis of the use cases*. A thorough analysis of the use cases provided us with some more concrete requirements for DFM development.
4. *The analysis of the COMPOSITION architecture and the identification of relevant components*. In this step the main activities, were the definition of the role of DFM in project's architecture and the definition of data exchange formats that the architecture supports. The DFM selected as the component which describes both the static and the dynamic data in a common format available to the related IIMS components. Every component will store or retrieve data based on this format. Both XML and JSON formats are supported by the projects architecture.
5. *The evaluation of the existing standards*. In this step, we selected the most suitable standards for the COMPOSITION purposes. The evaluation and the selection of the standards based on the results of the analyses from the previous steps. Finally the selected standards were: B2MML, BPMN, gbXML, x3d and OGC for Observation and Measurements.
6. *The implementation of the DFM*. This is the last step in which the DFM was implemented as a compilation of both XSD and JSON schemas. The selected standards from step 5 were imported to the DFM schema and were used for the schema's development.

Both XML and JSON formats were selected as the data types for static and dynamic data descriptions respectively. The XML schema is an industry supported standard for data structure specification. Well-known standards such as B2MML and gbXML which fit perfect with the project's needs are implemented in XML format. Furthermore, for the BPMN diagrams defined that is easier to be exported and handled in XML format. For the dynamic data descriptions the OGC for Observation and Measurements was selected which is offered in both XML and JSON format. The JSON format was adopted in current version of DFM in order to be compatible with OGC SensorThings standard that will be used to manage observations and metadata from diverse IoT sensor systems from the shop-floor. This standard includes the OGC Observation and Measurement data model in JSON format only.

The figure below presents a high level structure of the overall implemented DFM schema. The XSD Diagram tool (XSD Diagram, 2017) is used for the figure's exportation. However, the Events part are in JSON format.

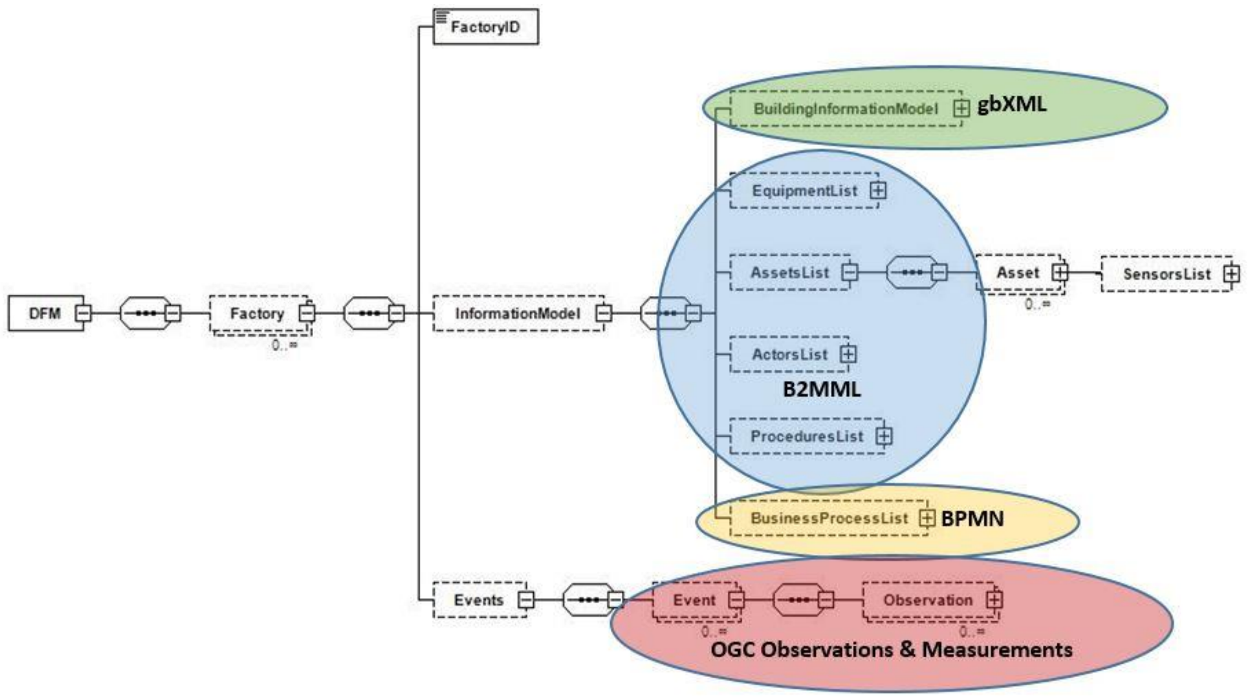


Figure 6: High level structure of DFM

6.2 Specifications of Digital Factory Model Schema

The COMPOSITION Digital Factory Model is able to describe all the information related to a factory such as buildings, assets, actors, processes and sensors' measurements and analytic tools' predictions. Every specific factory is represented as a DFM instance. Every instance has a unique factory ID and contains representations of both static and dynamic data of the corresponding factory. As depicted in the figure 6 the DFM structure has three root components:

Factory ID: represents the unique ID of the factory

Information Model: contains all the static information of the factory

Events: contains all the dynamic information about the factory

6.2.1 Factory ID

The Factory ID is one of the three root elements as mentioned before. However this element is just a String type element without any other sub-components. It is a very important element for the DFM structure as it describes each factory in a unique way. Every factory instance is connected to its own Factory ID and both the dynamic and the static data for this factory instance are stored or retrieved based on this ID. To sum up the Factory ID was adopted in order to distinguish data from different factories and to correlate every set of data with the factory they belong.

The other two root elements of the DFM, contain many sub-components, thus they will be presented in details.

6.2.2 Information Model

The Information Model of the DFM is a complex type element which contains all the static information of the factory. As depicted in the figure 6 the Information Model is comprised by a number of components. More precisely, the representation of a shop-floor in the COMPOSITION contains the following components:

- Building Information Model
- Equipment List
- Assets List
- Actors List
- Procedures List
- Business Process List

The above components are complex type elements which are based on well-known standards and schemas and they are able to provide all the necessary means for the factory's static information descriptions.

6.2.2.1 Building Information Model

The Building Information Model contains all the information related to the geometry of buildings. They are static information about walls, windows, streets, zones etc. The building information of a factory have been selected to be modelled based on COMPOSITION's requirement for shop-floor level modelling and use cases such as UC-BSL-5 Equipment Monitoring and Line Visualisation, and UC-KLE-3 Scrap metal and recyclable waste transportation (from bins to container). For these use cases, information about buildings' views is required in order to describe the location of different assets inside the factory. Besides these use cases in which is clear that building information should be modelled, in almost all intra-factory use cases the building information modelling is important in order to correlate the modelled sensors, assets and actors with the factory areas.

The Building Information Model of the COMPOSITION's DFM is a complex type element which is covered by two well-known standards, gbXML and x3d.

The gbXML stands for Green Building XML is the core standard which is used for building information's modelling in DFM. It is an open, well-known schema which is easily portable. It is supported by a wide range of design tools and it provides to DFM schema all the needed means for factory's spaces description and modelling.

The x3d schema is also imported and used for building information modelling. It is used for 3D modelling and geometry definitions. The x3d schema provides coordination's description and it is compatible with gbXML. It is used to support the gbXML schema's geometry definitions.

The next figure presents a high level view of the DFM's Building Information Model structure and its adopted schemas.

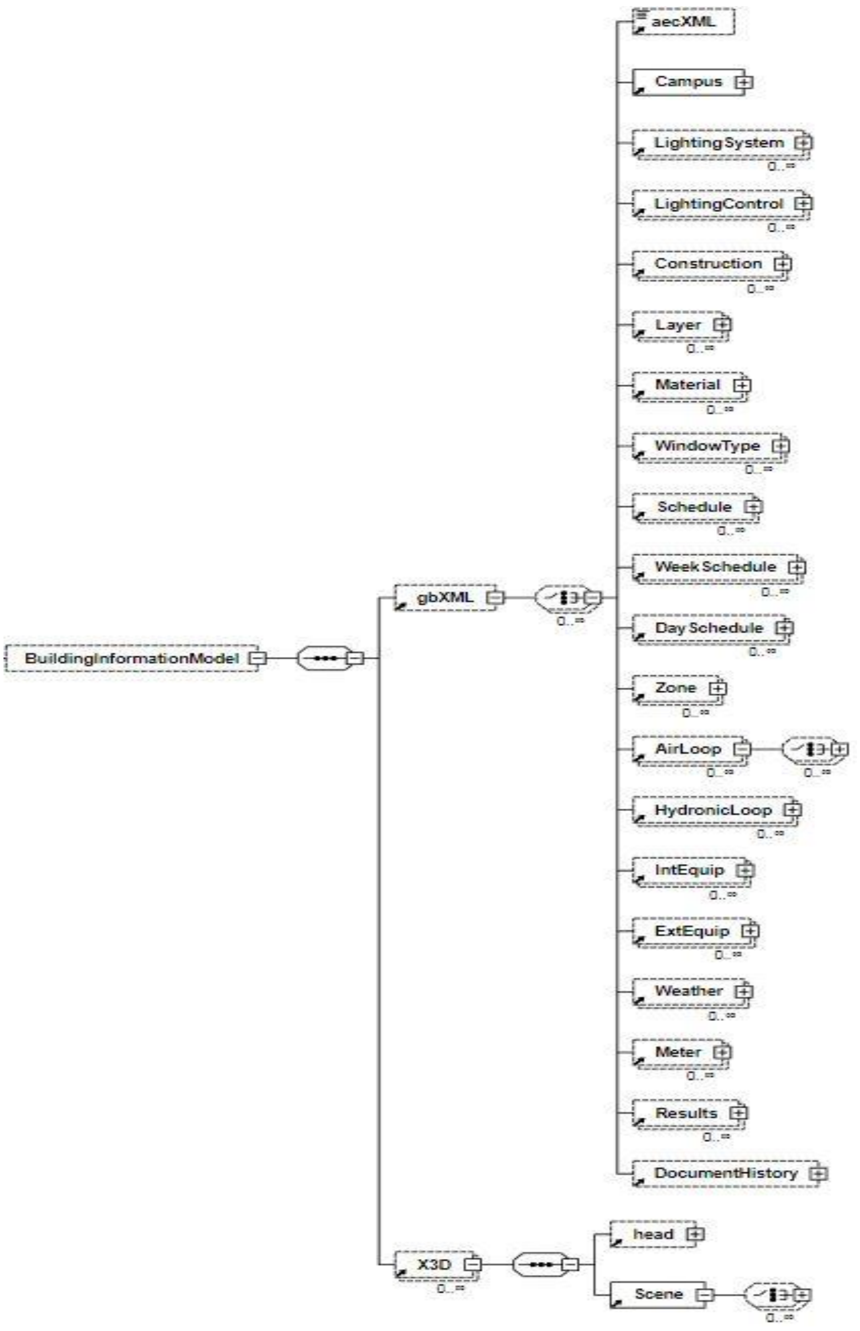


Figure 7: Building Information Model schema

6.2.2.2 Equipment List

The Equipment List offers the data types for the description of manufacturing equipment in the factory. In a DFM instance this list contains the installed equipment located in a specific space in the shop-floor. In order to cover COMPOSITION requirement for resources modelling and use case scenarios related to resources monitoring and management, the B2MML standard was adopted for the equipment modelling. The information from Equipment List will be used from IIMS components such as Decision Support System and Monitoring framework. As the B2MML is a well-known standard which is used by many integrated systems in manufacturing environments, it was an ideal candidate to cover the needs of COMPOSITION's DFM in the part of resources' description. It was selected over MIMOSA standard because B2MML offers a more flexible structure and addresses better the project requirements related to manufacturing resources descriptions.

The next figure presents a high level view of the structure of DFM's Equipment List based on B2MML schema:

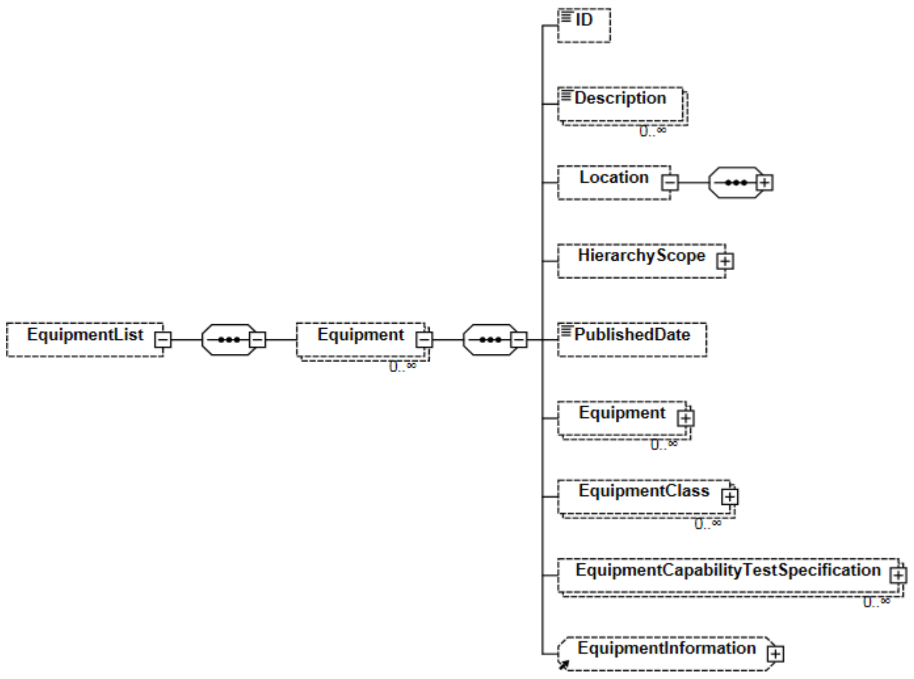


Figure 8: Equipment List schema

6.2.2.3 Assets List

The Assets List aims to the description of the assets associated with the manufacturing domain. In a DFM instance this list contains the assets which belong to a factory and they are located in a specific space in the shop-floor. As in the case of the Equipment List, in order to cover the COMPOSITION requirement for resources modelling and use case scenarios related to resources monitoring and management, the B2MML standard was adopted for the assets modelling. The B2MML is selected in this case as well, as it is a well-known standard which is used by many integrated systems in manufacturing environments.

Resource Catalogue

The B2MML package related to the assets has been updated for the COMPOSITION purposes. Every factory asset, for example a machine has installed built-in sensors or newly deployed sensors in order to cover the project use cases' requirements. A sensors list has been created and added on the assets as the modelling of sensors is mandatory. In the first version of this document the DFM schema contained a sensors list connected to a shop-floor in general and not in an asset. This was considered as a wrong approach and it is replaced by the presented approach in this document. Now every sensor is connected to a factory asset. Sensors related to the pilots such as vibration sensors installed in machines, light barriers, built in sensors in machines and bin's fill level monitoring sensors are modelled. The Assets List equipped with Sensors List is going to be used as a kind of Resource Catalogue for the rest of IIMS components. As soon as a sensor deployed in COMPOSITION (before or after the system is put in use), information on how to identify data (OGC ST Observations) coming from this sensor are added to the DFM and distributed to other parts of the system. The BMS will connect to a sensor. The BMS assigns a datastream id to the data from the sensor and publishes this to COMPOSITION as an OGC ST observation. Other components can query the DFM to find out which datastream id is used for the information they are interested in and subscribe to live data or query for historical data. These ids will be the sensors id coming from Sensors List that belongs to a factory asset (machine or a bin etc.).

A schema for sensors' description similar to one that was used at SatisFactory and Adapt4EE (Adapt4EE, 2011) EU projects was also adopted in order to cover the COMPOSITION's needs. This schema offers a wide variety of datatypes for the representation of the installed sensors such as ID, position, unit of measurement, min/max measurements and description. It is selected over other standards related to sensors modelling e.g. SensorML as it was considered that this schema offers all the necessary means for a sensors list's description without adding a complex structure in the DFM as the use of other ready standards would do. This schema structure is similar with B2MML structure and so it is easy to be validated by the DFM API without the concern of a new and complex schema's validation. Moreover, it will be easier for other components and end users to use this format because they are familiar with the Equipment List's structure for example, rather than to try understand and use a new and more complex format.

Furthermore, the described assets schema that contains the above described sensors list can be easily used by project components which are familiar with OGC SensorThings.

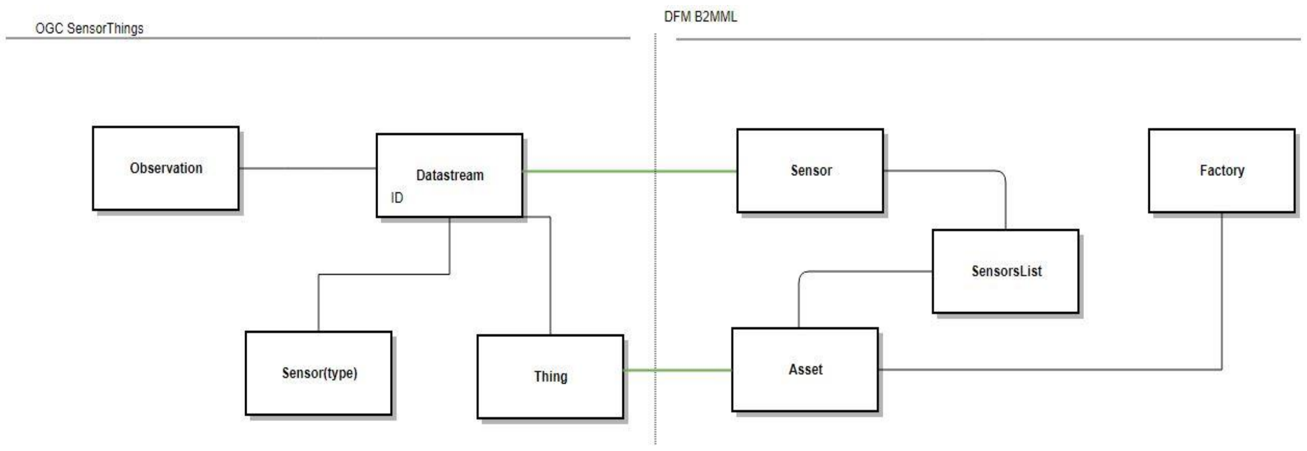


Figure 9 DFM Assets and OGC SensorThings Mapping

A DFM’s Asset class is equivalent with the Thing class of OGC SensorThings. A Sensor class of DFM is mapped to a SensorThings Datastream. For example, a machine in the production line of a factory is considered as an asset/thing. For this machine are available some sensors/datastream. A component such as the Learning Agent of COMPOSITION which uses SensorThings is able to find available datastreams in a machine of digital factory instance of COMPOSITION by using the aforementioned mapping between the two data models. The automatic mapping is enabled by DFM API services which are described in the next chapter of this report. The schema of DFM Asset List with a structure familiar to B2MML is presented in the following figure:

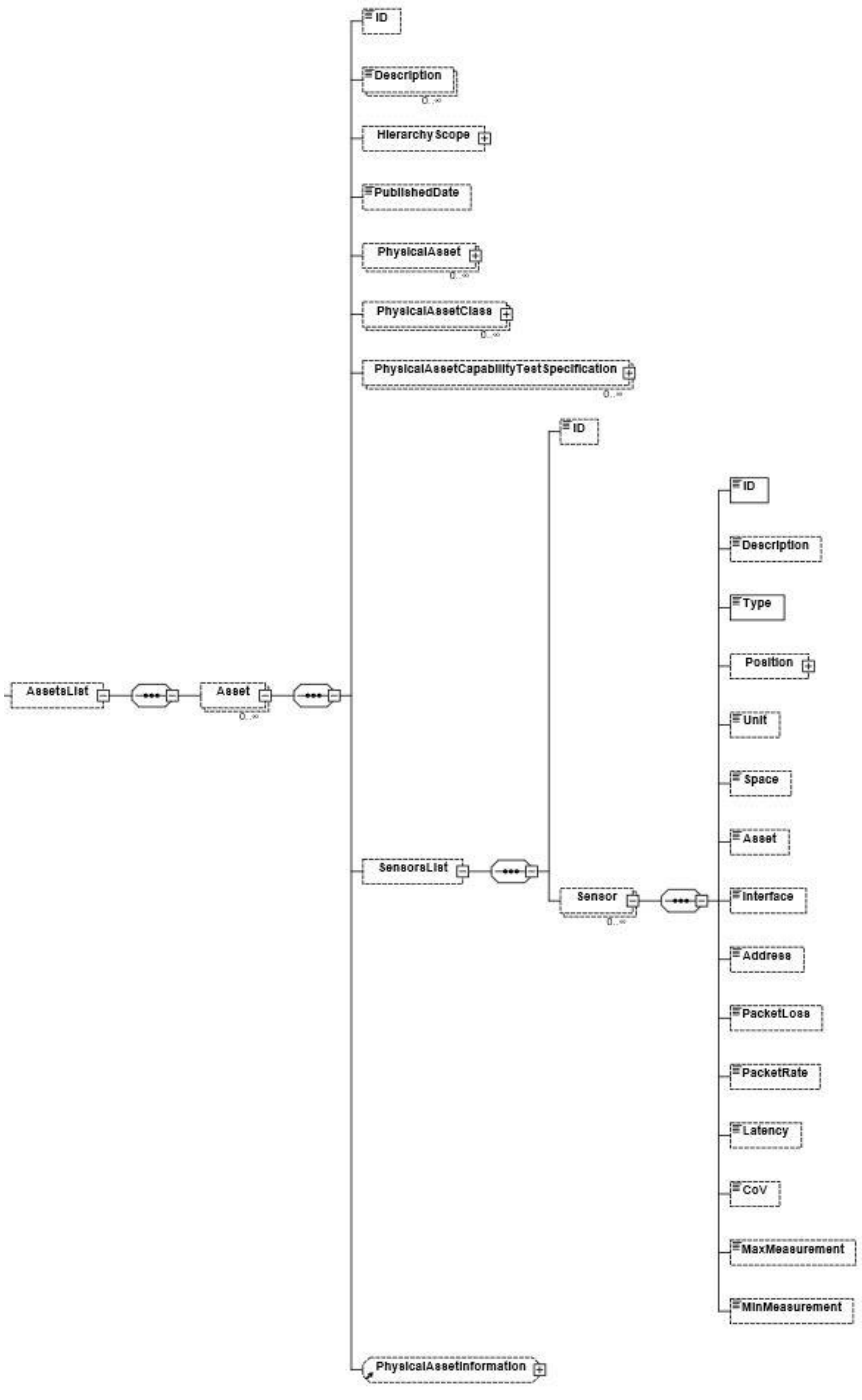


Figure 10: Assets List schema

6.2.2.4 Actors List

The DFM's Actors List contains information regarding to actors who participate in the use cases of the project. The representation of actors is vital for the COMPOSITION use cases such as UC-KLE-1 and UC-BSL-2 which are related to predictive maintenance. In these use cases the actors should be informed by Decision Support System in order to take action.

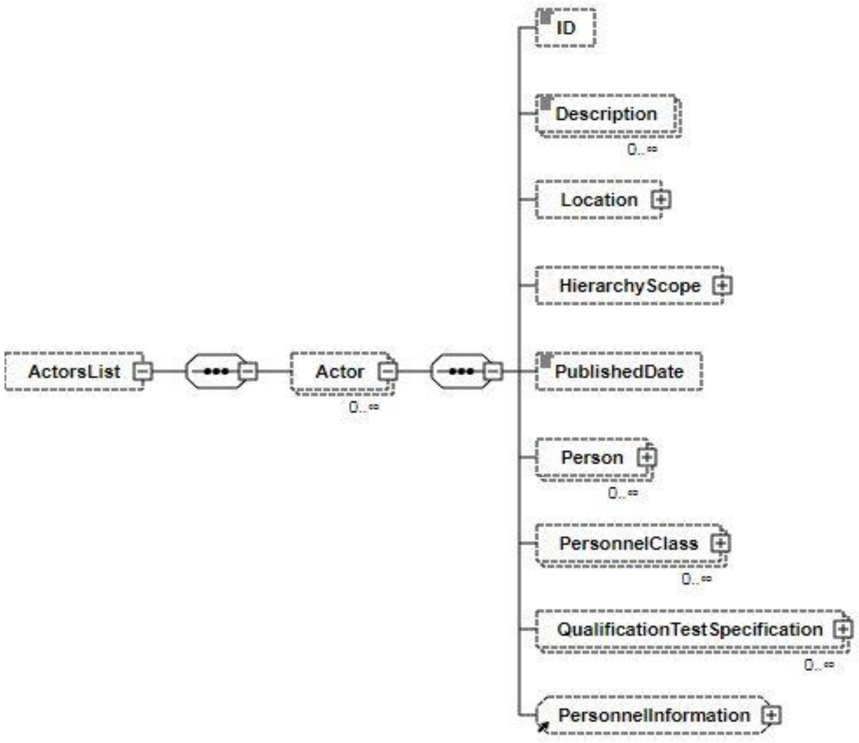
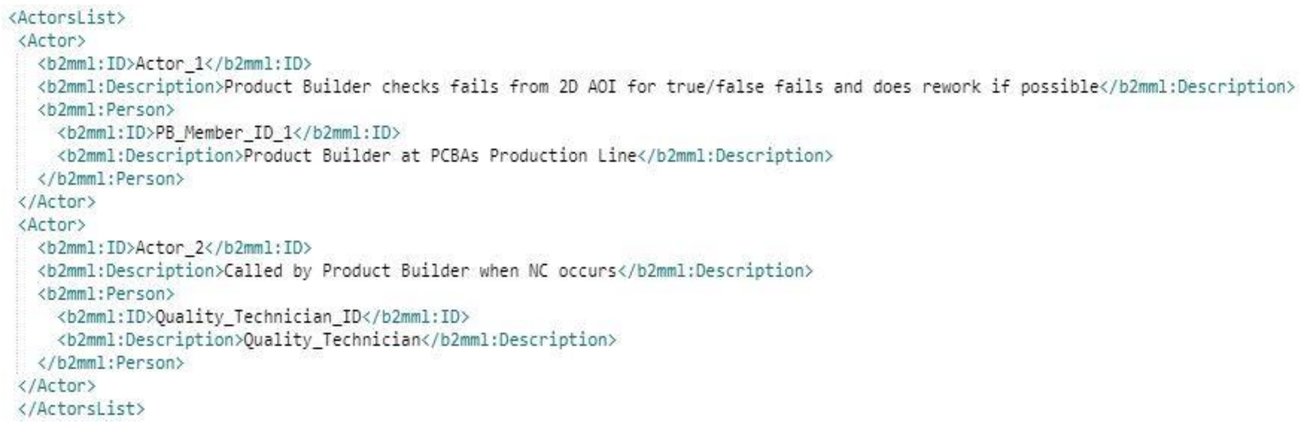


Figure 11: Actors List schema

As depicted in the previous figure the Actors List is also covered by B2MML schema and it has a similar structure with the aforementioned Assets and Equipment Lists. The B2MML schema provides data types for the description of an actor such as name, ID, job description, location in the factory and personnel class. These types for staff's description were evaluated as sufficient for the COMPOSITION's needs.

In following figure an XML example for some actors' representation is presented. Indicatively we chose to present the XML syntax of the Actors List. However, all the other lists based on B2MML schema have similar syntax.



```
<ActorsList>
 <Actor>
   <b2mml:ID>Actor_1</b2mml:ID>
   <b2mml:Description>Product Builder checks fails from 2D AOI for true/false fails and does rework if possible</b2mml:Description>
   <b2mml:Person>
     <b2mml:ID>PB_Member_ID_1</b2mml:ID>
     <b2mml:Description>Product Builder at PCBAs Production Line</b2mml:Description>
   </b2mml:Person>
 </Actor>
 <Actor>
   <b2mml:ID>Actor_2</b2mml:ID>
   <b2mml:Description>Called by Product Builder when NC occurs</b2mml:Description>
   <b2mml:Person>
     <b2mml:ID>Quality_Technician_ID</b2mml:ID>
     <b2mml:Description>Quality_Technician</b2mml:Description>
   </b2mml:Person>
 </Actor>
 </ActorsList>
```

Figure 12: Actors List XML Representation

6.2.2.5 Procedures List

The Procedures List of a DFM instance contains all the information related to the procedures or activities taking place during a use case. This list will be in conjunction with the Actors List or Assets List. A procedure will represent the activity or the procedure that an actor or an asset such as a machine executes. The Procedures List structure contributes to the conceptual connection between factory resources.

Also in this case, the Procedures List element covered by B2MML standard as has mentioned before, it is a well-known standard which is used by many integrated systems in manufacturing environments and is able to cover the COMPOSITION's IIMS needs.

The structure of Procedures List schema is presented to figure 13. As depicted in the figure the structure of this schema is similar to the rest schemas were provided by B2MML standard.

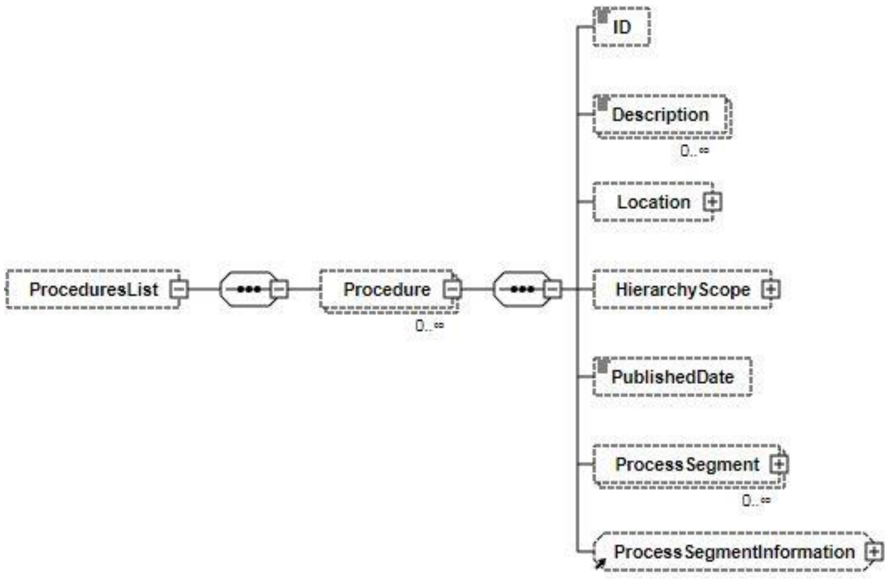


Figure 13: Procedures List schema

6.2.2.6 Business Process List

The Business Process List of DFM contains all the BPMN diagrams related to factory processes. All the processes that will have finally been selected to be modelled as BPMN diagrams will be imported to the corresponding DFM instance. The BPMN diagrams will be designed at Task 3.1 Process Modelling and Monitoring Framework. These diagrams will be exported to XML format which is valid with the DFM schema and they will be imported to the Business Process List of the DFM instances.

The Business Process List element of the DFM was covered by OMG's BPMN XML package. This package provides schemas which offer all the necessary means for the representation of a BPMN diagram in a DFM instance.

In a BPMN diagram all the processes are described as tasks. The factory's assets, actors, equipment and other resources participate in tasks/process. In the DFM instances these resources are correlated with task/processes using some common IDs.

The next figure presents a high level view of the structure of DFM's Business Process List based on BPMN schema:

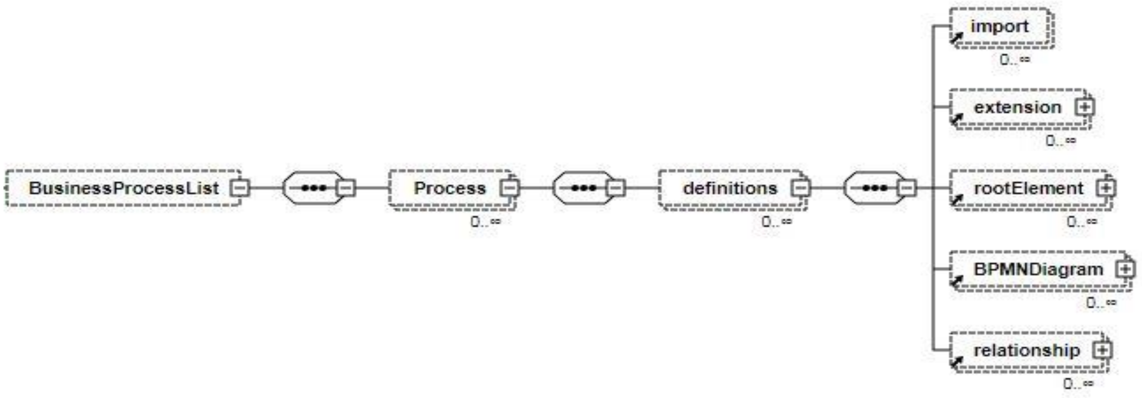


Figure 14: Business Process List schema

6.2.3 Events

The Events element is the third root element of COMPOSITION DFM schema's structure. An Event is a complex type element which contains dynamic information related to a shop-floor. Generally, the DFM schema consists the basic schema for the description of live data coming from factory (sensors data, prediction tools data etc.) that is followed by the related components. In a DFM 'live' instance the Events structure contains dynamic virtual data such as real time predictions/events coming from IIMS analytics tools. However, the sensors data are described by the same format but they are stored to the BMS as they are considered real data of the factory/building and not digital one as the predictions.

In order to cover the DFM's requirements for events modelling, the OGC standard for Observation and Measurement was adopted. More specifically, this standard defines JSON schemas for observations and measurements for features involved during the observations. It provides document models for the exchange of information describing observation acts and their results. This standard offers structure for the description of measurements such as result, related observations, time, observed property, unit of measurement etc. This format match pretty well with the needs of predictions' modelling. Moreover the OGC standards will also be used by IIMS to manage observations and metadata from the IoT sensor systems from the shop-floor. Thus, the use of the same standard supports the use of the same concepts and definitions among the IIMS components and offers interoperability between different components such as DFM, BMS, DLT, SFT, DSS etc.

The OGC O&M schema includes multiple JSON schemas (nested JSON schemas). This enable the re-use of same context as it is defined once. Moreover, it enables the objects definition comes from different documents. The OGC O&M JSON schema is packaged in 11 JSON schemas that define generic datatypes and objects, and indicate the different documents' dependencies as well. These packaged schemas documents allow to define objects such as observations, sampling features, common types, geometry objects, time series data and metadata etc.

In the following figures are depicted the OGC O&M JSON schema and its internal dependencies, and the basic Observation type as they are documented from (O&M, 2018)

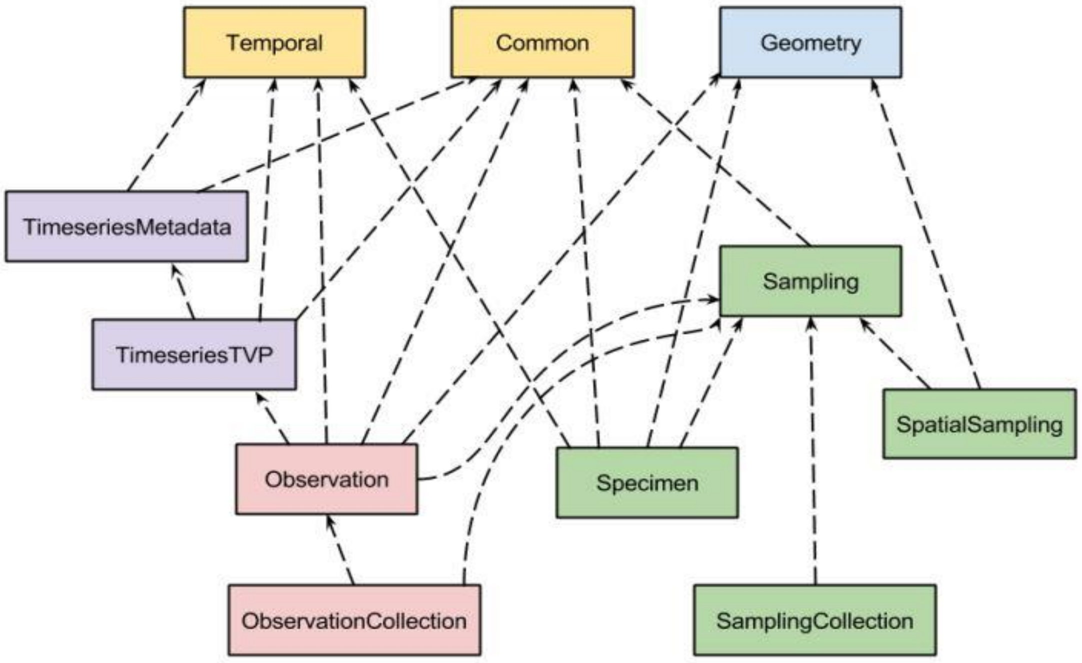


Figure 15: OGC Observation & Measurements JSON schemas' package (O&M, 2018)

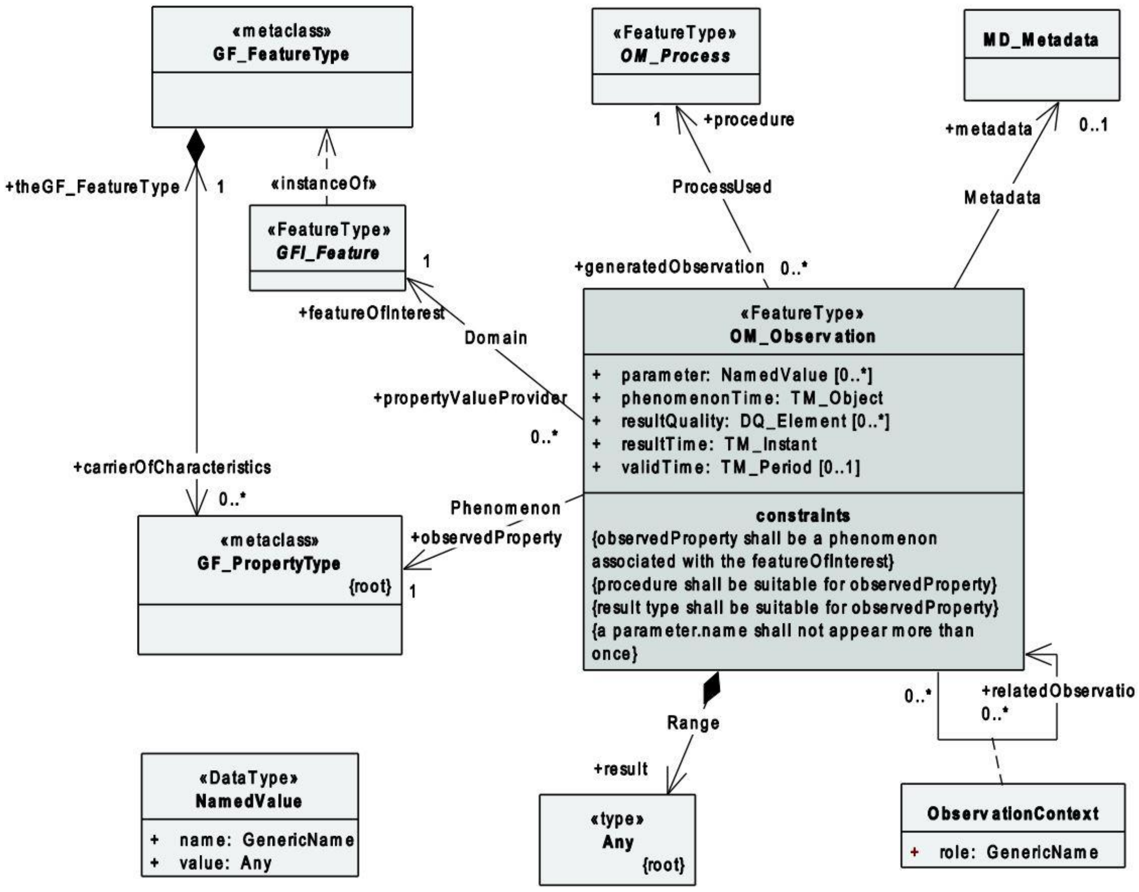
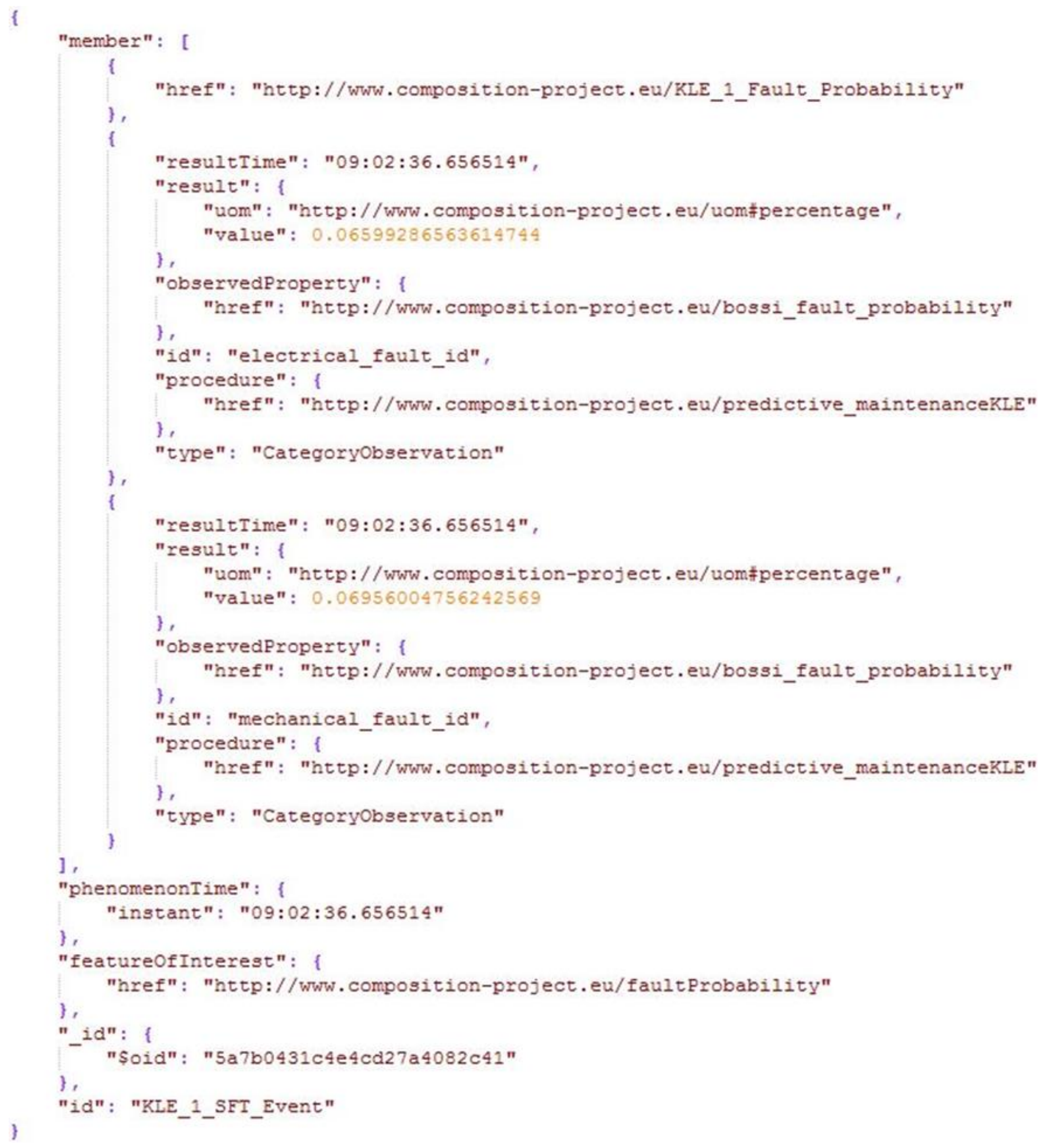


Figure 16: Basic Observation Type (O&M, 2018)

An example of DFM Event (OGC Observation & Measurements) representation related to a prediction of failure of the Bossi machine coming from SFT for UC-KLE-1 is depicted in the following figure. Also, a simplify example of a DLT prediction in OGC O&M format is described at Chapter 4 of this document.



```
{
    "member": [
        {
            "href": "http://www.composition-project.eu/KLE_1_Fault_Probability"
        },
        {
            "resultTime": "09:02:36.656514",
            "result": {
                "uom": "http://www.composition-project.eu/uom#percentage",
                "value": 0.06599286563614744
            },
            "observedProperty": {
                "href": "http://www.composition-project.eu/bossi_fault_probability"
            },
            "id": "electrical_fault_id",
            "procedure": {
                "href": "http://www.composition-project.eu/predictive_maintenanceKLE"
            },
            "type": "CategoryObservation"
        },
        {
            "resultTime": "09:02:36.656514",
            "result": {
                "uom": "http://www.composition-project.eu/uom#percentage",
                "value": 0.06956004756242569
            },
            "observedProperty": {
                "href": "http://www.composition-project.eu/bossi_fault_probability"
            },
            "id": "mechanical_fault_id",
            "procedure": {
                "href": "http://www.composition-project.eu/predictive_maintenanceKLE"
            },
            "type": "CategoryObservation"
        }
    ],
    "phenomenonTime": {
        "instant": "09:02:36.656514"
    },
    "featureOfInterest": {
        "href": "http://www.composition-project.eu/faultProbability"
    },
    "_id": {
        "$oid": "5a7b0431c4e4cd27a4082c41"
    },
    "id": "KLE_1_SFT_Event"
}
```

Figure 17: DFM Event - SFT Probabilities of Fault for KLE-1

7 Digital Factory Model API

As described in the executive summary and introductory sections besides the Digital Factory Model schema, a DFM API has been implemented and will be presented in this report. This API provides a wide set of interfaces/services. The IIMS components are able to access and manipulate data from the DFM 'live' instances, which represent a model of factory, using the services of the DFM API. In this section some key components of DFM API's implementation and its supported interfaces are presented.

7.1.1 Methodology and Implementation Technologies

The DFM API is designed for the purposes of the COMPOSITION project. It is the component which enables the access of some IIMS components into the common database of DFM instances. Based on section 4 of this report, the DSS, the Learning Agent, the Simulation tool and the Deep learning toolkit components should be able to manage the data which are stored in a common database. Moreover all the data exchanges should be in a format valid with DFM schema. So, the DFM API component is implemented to cover these needs and to offer the expected functionality.

The methodology was followed during the development process is described by the following steps:

1. The analysis of the requirements, the system's architecture and the project's use cases
2. The analysis of the available technologies and tools
3. The selection of technologies and tools based on requirements and needs of the project
4. The implementation based on selected technologies and using the selected tools
5. Testing and quality check of the implemented DFM API
6. The use of COMPOSITION Security Framework in order to secure DFM API's end points
7. The Deployment of DFM API as a Docker image in COMPOSITION production server

Requirements

The development of DFM API was driven by COMPOSITION requirements. The requirements based on D2.2 Initial requirements specification (COMPOSITION D2.2, 2017) and updated at D2.5 Lessons Learned and Updated Requirements Report I (COMPOSITION D2.5, 2018). Thus, based on the architecture definition and the project's use cases, the following main requirements were set for the DFM API implementation:

Table 6: Basic Requirement of DFM API's design

Requirement Number	Title	Short Description
COM-76	Monitoring framework and DSS shall be able to display production line assets and equipment as they represented in DFM	Mechanical equipment and other assets are important parts of the production line and they are represented at DFM. A thorough simulation of the production line should be able to involve them.
COM-149	COMPOSITION sensors' data should be described using common formats	The real time data should be exchanged in a common format in order to be available for all related components
COM-152	IIMS components should be able to store/retrieve data to/from DFM instances	A DFM API should be created in order to connect IIMS components with the data were stored in the common database
COM-153	DFM API should provide web services for data storing, retrieval or deletion	This requirement defines the DFM API's main functionality. The implemented API should offer services for data storing, retrieval or deletion
COM-154	DFM API should ensure that the data exchange will be based on the DFM schema	All the data exchanges is based on a common format (DFM schema) and validated against this format

Technologies and Tools

The technologies which are used for DFM API's development are indicated by two basic factors:

- Address the requirements were described above
- Use open and free technologies and tools as the project mention to do in DoA

The main selected technologies and tools are the following:

Java was selected as the implementation language. It is a general purpose, object oriented programming language. Java is one of the most popular programming languages in use, especially for client server web applications.

Web Services as defined by World Wide Web Consortium is a system designed to support interoperable Machine to Machine interaction over a network. Web services are server applications which can process and exchange data. They are selected as a perfect match to represent the required services.

REST or Representational State Transfer was selected as the architectural style of web services. REST offers better performance, modifiability and scalability to enable web services to work better on the Web. The REST architecture style is a client/server architecture where clients and servers exchange representations of resources by using a standardized interface and protocol. Resources are accessed using Uniform Resource Identifiers (URIs) which are the typical links on the Web.

HTTP stands for HyperText Transfer Protocol and was the selected protocol to be used by the RESTful API. This application protocol is used to link pages of hypertext and it is a way to transfer files. HTTP is the foundation of data communication for the Web. HTTP protocol is a supported protocol by COMPOSITION system architecture.

NoSQL databases provide mechanisms for retrieval and storage of data that is modelled in means different than the tabular relations used in relational databases. Compare to relational databases, the NoSQL databases are more scalable and provide better performance, and they address several issues that the relational databases are not designed to address. The basic types of NoSQL databases are the document databases, graph stores, key-value stores and wide-column stores.

MongoDB is a free and open-source cross-platform database. It is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use. MongoDB is a document database as it stores the data in flexible JSON-like documents. As it is a document database addresses perfectly the storing requirements of IIMS as it needs XML and JSON documents storing. It is a NoSQL database. MongoDB is simple for developers to learn and use it.

XML or eXtensible Markup Language is a mark-up language and it was designed to store and transport data. The XML language is described in more details at chapter 5 of this report.

JSON or JavaScript Object Notation was the selected syntax format for exchanging messages. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers. These properties make JSON an ideal format for data-exchange.

NetBeans IDE is a well-known Integrated Development Environment. It is a free and open source tool. NetBeans IDE provides a user friendly workspace and tools for easy and quick development of desktop, mobile and web applications. Almost all the main and wide-used programming languages such as Java, C/C++, PHP and JavaScript are supported for application development by the NetBeans.

Glassfish Server is an open-source application server for Java web applications' deployment. The Glassfish Server project started by Sun Microsystems for the Java EE platform and now it is supported by Oracle Corporation. The Glassfish is free software that allows developers to create enterprise applications that are portable, scalable, and easily integrated. It is easily integrated with the selected NetBeans IDE and offers all the required functionalities for the DFM API deployment.

Implementation

The DFM API was implemented as a Java web application using NetBeans IDE and the aforementioned technologies. The DFM API is offered through Restful web services. Its main functionality is to receive HTTP requests from IIMS components. Based on requests, the DFM API stores or retrieves data from MongoDB store which represents the database store component. After that, DFM API returns an HTTP response to the requested IIMS component. The response contains the requested resource from the MongoDB in the cases the requests are related to data retrieval. In the cases that the requests are about data storage or deletion, the response is just a simple message for successful operation. The data exchange formats are both XML and JSON. The exchange formats should be valid with the DFM schema as it defined in the previous chapter of this document.

The next figure presents a high level overview of the implemented DFM API's architecture:

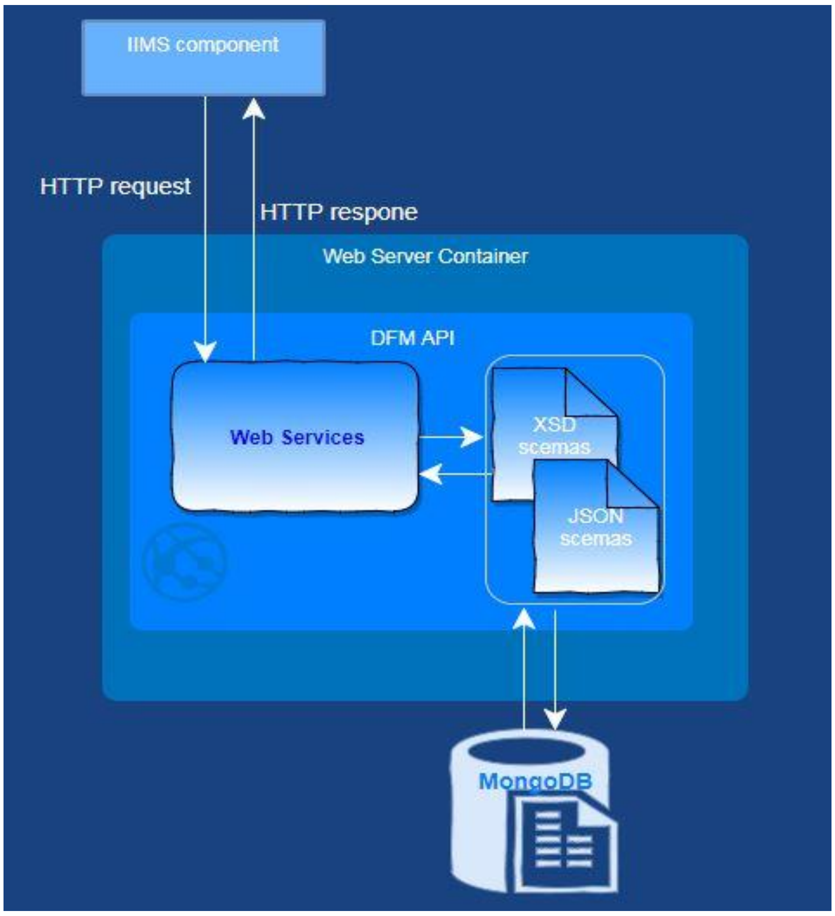


Figure 18: High level overview of DFM API architecture

Testing

After the development of DFM API, it was deployed in a Glassfish server container. Then all the available web services which will be presented in details at the following section were called using Postman Rest Client (Postman, 2018). All the calls results were evaluated and any problems related to supported functionalities were fixed.

Integration tests with other tools coming from WP3 such as the Monitoring Framework (from Task 3.1), the Simulation and Forecasting Toolkit (from Task 3.3) and the Decision Support System (from Task 3.4) have been performed in order to evaluate connectivity issues and the overall behavior and performance of the DFM API.

DFM Instances

DFM Instance related to BSL's PCBAs production line and KLEEMANN production line have been created. They are implemented using the DFM API's services. Data related to actors and assets were collected based on the pilot partner's descriptions and modelled in a format validate with DFM schema. Then these data are posted to MongoDB using the DFM API and the corresponding web services. Furthermore, a BPMN diagram about PCBAs production line was created in Task 3.1 and extracted in an XML format valid with DFM schema. This diagram was posted to MongoDB by using the DFM API's services as well. Moreover, all datastreams coming of the two factories sensors were modeled and added to the DFM's Assets List which is the Resource Catalogue of the IIMS. These DFM Instance are continuously updated with more information from different IIMS components via DFM API services.

7.1.2 Supported Interfaces

This section presents the catalogue of supported interfaces of the DFM API. Each interface is a service that an IIMS component is able to call in order to receive or store data to a common database. We describe all the provided web services of the latest version of the DFM API by presenting the type of the request, the type of the URL parameters or the data from the body of a request and the type of the response's data. The interfaces descriptions are presented separately for each DFM schema's component based on the descriptions of chapter 6.

7.1.2.1 Information Model interfaces

Overall Static Information

- String/XML *getFactoryInformation* (String factoryID)

  This function retrieves from the database all the static information of a factory based on its ID. Actually lists of actors, assets, equipment, procedures, sensors, business processes and buildings are returned. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is the URL parameter.

This is the only one overall supported interface from DFM API. All the other interfaces are component specific and they offer functionalities for data storage, retrieval or removal.

Building Information

- String/XML *setBuildingInformation* (String buildingXML)

  It sends an XML file compatible with the DFM's schema structure (actually gbXML) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the corresponding building information (gbXML) of the factory is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the building information is the body of the request. It is stored as a String (buildingXML).

- String/XML *getBuildingInformation*(String factoryID)

  It retrieves from the database all the buildings information stored in a DFM instance based on the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is the URL parameter.

- String/XML *getBuildingByID* (String factoryID, String buildingID)

  It retrieves from the database the information of a specific building stored in a DFM instance based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID and buildingID are URL parameters.

- StringXML *deleteBuilding* (String factoryID, String buildingID)

  It deletes the information of a building based on its own ID and the factory ID where the building belongs. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and buildingID are URL parameters.

- String/XML *deleteBuildingInformation* (String factoryID)

  It deletes the buildings information model stored in the DFM related to the factory ID. A text message is returned.

  It is called by a corresponding GET request. The factoryID is URL parameter.

Equipment List

- String/XML *setEquipmentList* (String equipmentXML)

It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about factory's equipment is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

It is called by a corresponding POST request. The XML file which contains the complete list of equipment information of the factory is the body of the request. It is stored as a String (equipmentXML).

- String/XML *setEquipment* (String equipmentXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the corresponding equipment information is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the equipment information is the body of the request.

- String/XML *getEquipmentList*(String factoryID)

  It retrieves from the database all the equipment information stored in a DFM instance based on the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *getEquipmentByID* (String factoryID, String equipmentID)

  It retrieves from the database the information of a specific equipment element related to a DFM instance based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID, equipmentID are URL parameters.

- String/XML *deleteEquipmentList* (String factoryID)

  It deletes based on the factory ID, the complete list with the equipment information of a factory which is stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *deleteEquipment* (String factoryID, String equipmentID)

  It deletes the information from the database of a specific equipment element based on its own ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and equipmentID are URL parameters.

Assets List

- String/XML *setAssetsList* (String assetXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about the factory's assets is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the complete list of the assets of the factory is the body of the request.

- String/XML *setAsset* (String assetXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the corresponding asset information is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

It is called by a corresponding POST request. The XML file which contains the information of a factory's asset is the body of the request.

- String/XML *getAssetsList*(String factoryID)

  It retrieves from the database all the assets information stored in a DFM instance based on the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is a URL parameter.

- String/XML *getAssetByID* (String factoryID, String assetID)

  It retrieves from the database the information of a specific asset related to a DFM instance based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID, assetID are URL parameters.

- String/XML *deleteAssetsList* (String factoryID)

  It deletes based on the factory ID, the complete list of the assets of a factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *deleteAsset*(String factoryID, String assetID)

  It deletes the information from the database of a specific asset based on its own ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and assetID are URL parameters.

Assets List and Services related to Resource Catalogue

- String/XML *setSensorsList* (String sensorXML)

  It sends an XML file compatible with the DFM's schema structure for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about the factory's sensors is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the complete list of the factory's sensors is the body of the request.

- String/XML *setSensor* (String sensorXML)

  It sends an XML file compatible with the DFM's schema structure for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the sensor's information which is described in the XML file is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the information of a factory's sensor is the body of the request.

- String/JSON *getAssetDatastream* (String factoryID, String assetID)

  It retrieves all the datastreams/sensors that are connected in a factory's asset. The DFM API read the factory ID and the corresponding asset ID and returns the sensors list of this asset.

- String/JSON *deleteSensorsList* (String factoryID, String assetID)

  It deletes based on the factory ID and asset ID, the complete list of the sensors/datastreams of an asset in the factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and assetID are URL parameters.

- String/JSON *deleteSensor*(String factoryID, String assetID , String sensorID)

It deletes the information from the database related to a specific sensor of an asset, based on asset ID, sensor ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

It is called by a corresponding GET request. The factoryID, assetID and sensorID are URL parameters.

Actors List

- String/XML *setActorsList* (String actorXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about the factory's actors is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the complete list of the actors of the factory is the body of the request.

- String/XML *setActor* (String actorXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the actor information which is described in the XML file is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the information of a factory's actor is the body of the request.

- String/XML *getActorsList*(String factoryID)

  It retrieves from the database all the actors information stored in a DFM instance based on the factory ID.The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *getActorByID* (String factoryID, String actorID)

  It retrieves from the database the information of a specific actor related to a DFM instance, based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID and actorID are URL parameters.

- String/XML *deleteActorsList* (String factoryID)

  It deletes based on the factory ID, the complete list of the actors of a factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *deleteActor*(String factoryID, String actorID)

  It deletes the information from the database related to a specific actor, based on its own ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and actorID are URL parameters.

Procedures List

- String/XML *setProceduresList* (String procedureXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about the factory's procedures is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

It is called by a corresponding POST request. The XML file which contains the complete list of the procedures of the factory is the body of the request.

- String/XML *setProcedure* (String procedureXML)

  It sends an XML file compatible with the DFM's schema structure (actually B2MML structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file. The procedure's information which is described in the XML file, is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the information of a factory's procedure is the body of the request.

- String/XML *getProceduresList*(String factoryID)

  It retrieves from the database all the procedures' information stored in a DFM instance based on the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *getProcedureByID* (String factoryID, String procedureID)

  It retrieves from the database the information of a specific procedure related to a DFM instance, based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID and procedureID are URL parameters.

- String/XML *deleteProceduresList* (String factoryID)

  It deletes based on the factory ID, the complete list of the procedures of a factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *deleteProcedure*(String factoryID, String procedureID)

  It deletes the information from the database related to a specific procedure, based on its own ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and procedureID are URL parameters.

Business Processes List

- String/XML *setBusinessProcessList* (String bpmnXML)

  It sends an XML file compatible with the DFM's schema structure (actually BPMNL structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the total information about the factory's business processes (BPMN diagrams) is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the complete list of the business processes diagrams of the factory is the body of the request.

- String/XML *setBusinessProcess* (String bpmnXML)

  It sends an XML file compatible with the DFM's schema structure (actually BPMN structure) for storing to the MongoDB. The DFM API reads the factory ID from the posted XML file, and the business process information (BPMN diagram) which is described in the XML file is stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The XML file which contains the information of a factory's BPMN diagram is the body of the request.

- String/XML *getBusinessProcessList*(String factoryID)

  It retrieves from the database all the BPMN diagram's information stored in a DFM instance based on the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *getBusinessProcessByID* (String factoryID, String bpmnID)

  It retrieves from the database the information of a specific business process diagram related to a DFM instance, based on its ID and the factory ID. The returned information will be an XML file in text form.

  It is called by a corresponding GET request. The factoryID and bpmnID are URL parameters.

- String/XML *deleteBusinessProcessList* (String factoryID)

  It deletes based on the factory ID, the complete list of the business processes of a factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID is URL parameter.

- String/XML *deleteBusinessProcess*(String factoryID, String bpmnID)

  It deletes the information from the database related to a specific business process diagram, based on its own ID and the corresponding factory ID. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and bpmnID are URL parameters.

7.1.2.2 Events interfaces

Besides the interfaces related to the static information of a factory the DFM API provides a list of web services related to events which represent the dynamic data/analytics tools predictions of a factory. The following interfaces are related to dynamic data from DFM instances:

- String/JSON *setObservation (*String factoryID, String observationJSON*)*

  It sends a JSON file compatible with the DFM's schema structure (OGC standard for Observations and Measurements is used in this case) for storing to the MongoDB. The DFM API reads the factory ID, and information about the factory's event/observation are stored or updated (if exists) in the corresponding DFM instance. A message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding POST request. The JSON file which contains the observation details is the body of the request.

  String/JSON getObservationByID (String factoryID, String eventID)

  It retrieves from the database an observation stored in a DFM instance based on the factory ID and event ID.

  It is called by a corresponding GET request. The factoryID and eventID are URL parameters.

- String/JSON *deleteObservation* (String factoryID, String observationID)

  It deletes based on the factory ID and observation ID, the observation/event of a factory which are stored in a DFM instance. A text message is returned. This message informs the component which sends the request if the operation was successful or not.

  It is called by a corresponding GET request. The factoryID and observationID are URL parameters.

7.1.3 Deployment and Security

7.1.3.1 Deployment – Docker Container

The DFM API component is a web-based component deployed in a Docker container. A Docker image has been created for the DFM API and a corresponding Docker container. The Docker container of the DFM API is deployed at COMPOSITION intra-factory production server.

The DFM API was developed as a Java EE application and was packaged as a .war file. The Docker image of the DFM API was created by using the Docker official image of the Glassfish Server with the addition of the DFM API's .war file. The application servers with stateless applications such as Glassfish is easy to be dockerized and scale easier as each new instance can receive requests without any synchronization of state. As the DFM API uses MongoDB for its database, a Docker image for this database has been created as well. The Docker official image of the MongoDB is used as it is. The DFM API container is connected to MongoDB container in order to connect the API's services and the database.

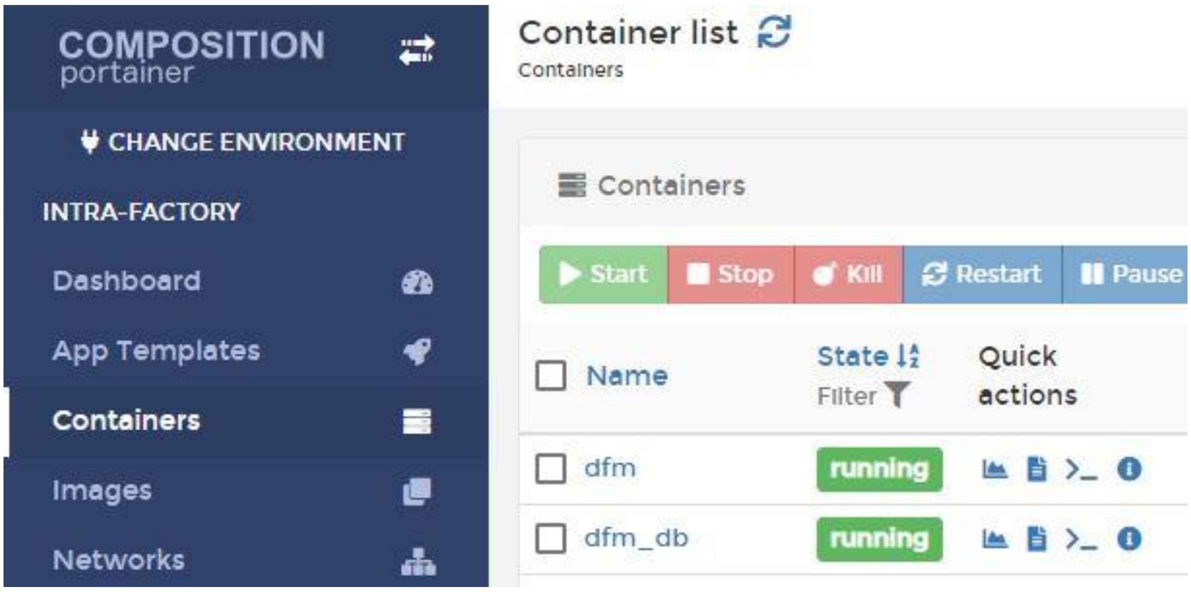


Figure 19: DFM Docker container at COMPOSITION Production Server

7.1.3.2 Security

All COMPOSITION components which expose RESTful APIs over the internet must enforce authentication using OpenID Connect. The LinkSmart® Border Gateway (BGW)[1] can secure these APIs such as DFM API by providing an overlay on top of all RESTful APIs, passing only authenticated and authorized requests to them.

A Basic Auth authentication will be used in order to secure the DFM API's end points. For the COMPOSITION purposes:

- User provides username/password in the REST request
- BGW intercepts the request and negotiates with an OpenID Connect server for a token
- If authenticated, BGW forwards the request to API and caches the token for upcoming requests until it expires

[1] https://docs.linksmart.eu/display/BGW

8. Conclusions and Future Work

In conclusion, this deliverable represents the current status of Task 3.2 Integrated Digital Factory Models of WP3 and describes the effort spent from M3 to M26 in this task. This status is almost the final one of the task as it will be completed by M28. More precisely, this report documents the COMPOSITION's DFM schema and DFM API.

A Digital Factory Model schema has been implemented and presented after a thorough analysis of standards related to manufacturing modelling and modelling formats. Well-known and widely used standards such as B2MML, x3d, gbXML and OGC for Observation and Measurements were adopted and used in DFM schema's implementation. All the DFM schema's components are described using both XML and JSON formats which offers a high level of simplicity, extensibility, interoperability and openness.

A Digital Factory Model API has been developed after an analysis of available technologies and tools, and consideration of the project's requirements and architecture. The DFM API is offered through Restful web services. By using the DFM API and its supported interfaces/services, the IIMS components are able to store, retrieve or delete data from DFM instances. The data exchange format should be valid against the DFM schema. The DFM API is deployed in a Docker container in order to interact with the rest of IIMS components.

The future steps of the previous iteration of this document (D3.2) have been implemented. The DFM schema has been updated in order to cover the demands for events representation in JSON format, the DFM API services modified and extended, the DFM instances are continuously populated with new data, the API is deployed in a Docker container and the API's endpoints were secured.

The outcome of this deliverable mainly affects the WP3 and its components such as the Monitoring Framework, the Simulation and forecasting tool and the Decision Support System. Besides this, the deliverable is also connected with WP5 and components such as the Deep learning toolkit and the LinkSmart.

Finally, as it is perceived, the results of Task 3.2 is presented in this deliverable. The DFM schema and the DFM API components have already been implemented and presented. However, the work has been done may be further extended in order to cover possible needs by the end of the project (M36). Furthermore, the DFM schema/model will be public available in order to boost the research in the fields of manufacturing modeling and simulation.

9. List of Figures and Tables

9.2 Figures

Figure 1: COMPOSITION Functional view 7
Figure 2: Structure of the BPMN Monitoring Framework Based on D3.1 9
Figure 3: Sequence Diagram for the data route in with DFM and SFT COMPOSITION Components 13
Figure 4: DFM Schema for Actor class 14
Figure 5: Personnel card corresponding to the Actor class of the DFM schema 14
Figure 6: High level structure of DFM 20
Figure 7: Building Information Model schema 22
Figure 8: Equipment List schema 23
Figure 9 DFM Assets and OGC SensorThings Mapping 24
Figure 10: Assets List schema 25
Figure 11: Actors List schema 26
Figure 12: Actors List XML Representation 26
Figure 13: Procedures List schema 27
Figure 14: Business Process List schema 28
Figure 15: OGC Observation & Measurements JSON schemas' package (O&M, 2018) 29
Figure 16: Basic Observation Type (O&M, 2018) 29
Figure 17: DFM Event - SFT Probabilities of Fault for KLE-1 30
Figure 18: High level overview of DFM API architecture 33
Figure 19: DFM Docker container at COMPOSITION Production Server 40

9.3 Tables

Table 1: Abbreviations and acronyms are used in this deliverable 5
Table 2: Observation's suggested fields for Deep Learning Toolkit 11
Table 3: Observation's optional fields for Deep Learning Toolkit 11
Table 4: Sample features 12
Table 5: Main modelling requirements 19
Table 6: Basic Requirement of DFM API's design 31

10. References

(COMPOSITION, 2016)	GRANT AGREEMENT 723145 — COMPOSITION: Annex 1 Research and inovation action
(SatisFactory, 2015)	SatisFactory EU Project. http://www.satisfactory-project.eu/satisfactory/
(COMPOSITION D2.4, 2018)	COMPOSITION EU Project: D2.4 The COMPOSITION architecture specification II
(COMPOSITION D2.2, 2017)	COMPOSITION EU Project: D2.2 Initial Requirements Specification. http://www.composition-project.eu/downloads/D2.2%20Initial%20Requirements%20Specification_V1.0.pdf
(COMPOSITION D2.5, 2018)	D2.5 Lessons Learned and Updated Requirements Report I
(COMPOSITION D3.2, 2017)	COMPOSITION Digital Factory Model I. https://www.composition-project.eu/about-composition/deliverables/
(B2MML, 2017)	Business To Manufacturing Markup Language: http://www.mesa.org/en/B2MML.asp
(GbXML, 2017)	Green Building XML: http://www.gbxml.org/
(MIMOSA, 2017)	MIMOSA - An Operations and Maintenance Information Open System Alliance: http://www.mimosa.org/
(BPMN, 2017)	Object Management Group Business Process Model and Notation: http://www.bpmn.org/
(OGC, 2017)	Open Geospatial Consortium: http://www.opengeospatial.org/
(OGC standards, 2017)	Open Geospatial Consortium: http://www.opengeospatial.org/docs/is
(SensorML, 2017)	Open Geospatial Consortium - SensorML: http://www.opengeospatial.org/standards/sensorml
(O&M, 2018)	Open Geospatial Consortium Observations and Measurements: http://www.opengeospatial.org/standards/om
(SensorThings, 2017)	Open Geospatial Consortium - SensorThings: http://www.opengeospatial.org/standards/sensorthings
(X3D, 2017)	Web 3D Consortium – x3d: http://www.web3d.org/x3d/what-x3d
(PMML, 2017)	Data Mining Group - Predictive Model Markup Language: http://dmg.org/pmml/v4-1/GeneralStructure.html
(XSD Diagram, 2017)	XSD Diagram Tool: http://regis.cosnier.free.fr/?page=XSDDiagram
(Adapt4EE, 2011)	Adapt4EE EU project: http://www.adapt4ee.eu/adapt4ee/index.html
(Postman, 2018)	Postman web site: https://www.getpostman.com/