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

D3.2 Digital Factory Model I

Date: 2017-11-29

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:	D3.2 - Digital Factory Model I
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	Nikolaos Kaklanis, Alexandros Nizamis(CERTH)	2017-06-27	Initial Document Structure and Inputs
0.2	Alexandros Nizamis, Dimosthenis Ioannidis (CERTH)	2017-09-20	Content to Sections 3 and 4 about Introductory sections and Overall COMPOSITION Architecture is added
0.3	Vagia Rousopoulou (CERTH)	2017-10-15	Content to Section 5 - State of the art analysis is added
0.4	Alexandros Nizamis, Vagia Rousopoulou (CERTH)	2017-11-11	Content to Section 6 - DFM Schema and Section 7 – DFM API is added
0.5	Farshid Tavakolizadeh, Junhong Liang(FIT) Paolo Vergori (ISMB) Ziazios Constantinos (ATL)	2017-11-17	Content to Section 4 related to LinkSmart, BPMNs, Deep Learning Toolkit and Decision Support System is added. General recommendations about the document
0.6	Alexandros Nizamis, Dimosthenis Ioannidis (CERTH)	2017-11-20	Content to conclusions and future work sections. Finalization for internal review
1.0	Alexandros Nizamis, Vagia Rousopoulou (CERTH)	2017-11-29	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Paolo Vergori (ISMB)	2017-11-23	Contents and methodologies are sound and promising. In the next iteration is expected an alignment of the DFM with JSON formats used in the intra-factory scenario
Rodrigo Diaz Rodriguez (ATOS)	2017-11-27	The document is in good shape, well-written and structured. In some sections a more detailed description is needed.

Index:

1 Executive Summary

This deliverable presents the first results of Task 3.2 Integrated Digital Factory Models. It aims to describe and analyse the current status of COMPOSITION Digital Factory Model (DFM). More precisely, in this report both the implemented DFM schema and the developed current version of the 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 shop floor 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 will be used as the aforementioned model. DFM will provide the digitalization of industrial aspects as a common model of information elements. The DFM will be used as the common format for information exchange between COMPOSITION 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 that analysed and some of them used in DFM's design and the specifications of current version of DFM. Besides the DFM schema and structure, a DFM API is also presented. 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 M15 (date of this deliverable). The final outcome of the Task 3.2 and the final versions of both DFM schema and DFM API will be presented on M26 when the Task 3.2 ends. All the updates until M26 will be reported at D3.3 Digital Factory Model II.

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
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
MQTT	Message Queue Telemetry Transport
OGC	Open Geospatial Consortium
OGI	Oil and Gas Interoperability
PMML	Predictive Model Markup Language
SensorML	Sensor Model Language
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 current version of an API which offers services for storing/retrieving data to/from a common database.

As this deliverable is followed by a part two on M26 and the project is even not in its half, both the DFM schema and API versions are not the final ones. Maybe some of the selected standards will be replaced by others if this is required by project's needs. Moreover, the catalogue of the provided web services will be extended. Finally, the instances of both KLE and BSL factories will be complete as soon as more data from pilot partners will be available and the required sensors will be installed as well.

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 schemas.

Section 7 is about the current version of DFM API. 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 outlines the next steps of Task 3.2 which will be presented at the task's end and deliverable D3.3 Digital Factory Model II in M26.

Section 9 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 finally 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 available data are stored in a database from Data Persistence component which is the data store of the COMPOSITION Intra-factory system. 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 D2.3-The COMPOSITION architecture specification I (COMPOSITION D2.3, 2017).

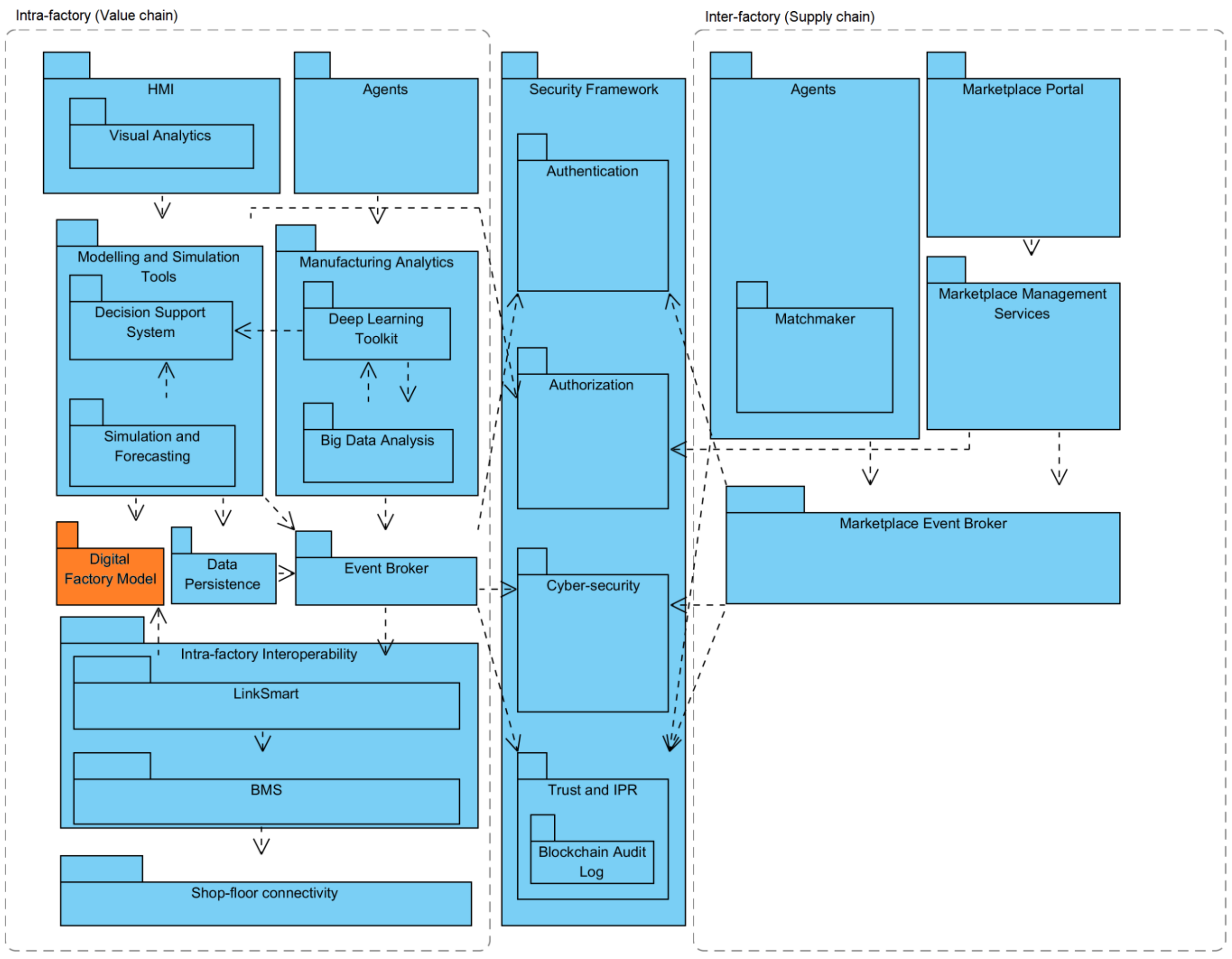


Figure 1: COMPOSITION Functional view

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 the DFM. 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 the DFM 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.

4.2.2 Intra-factory Interoperability

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, recently (M14) 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. All the updates will be presented in the corresponding components' deliverables and in the D2.4-The COMPOSITION architecture specification II. 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 such as the DFM component.

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. These 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 in the Data Persistence component. Furthermore, the real time data from sensory infrastructure are stored in the corresponding DFM instances. The Simulation and forecasting tool is able to get this data in a common format using the DFM API's services.

4.2.5 Decision Support System

Decision Support System integrates Digital Factory Models with sensor data, and other information and knowledge about the products, manufacturing, planning, simulation, communication and controls at all levels of planning and manufacturing. To design such system must be defined using specific data and algorithms. DSS architecture relates to the use of common data for all systems at different levels to allow working with the virtual models of factory.

Digital Factory Models define the necessary entities and data format that decision support system uses in order to process and associate data and for producing meaningful information for visualization performance, notifications and decision support.

There is a need to explicitly model the context of a decision support system, for example to determine how much the evidence from one decision case can be trusted in another, similar context.

Also, another is the visualization of performance that can offer helpful insights into the behaviour of the overall manufacturing process.

Also major challenge is to use data from the DFM related to the heterogeneity of tools and methods used for different activities, including simulations that could be done by different actors involved in the collaborative work.

4.2.6 Deep Learning Toolkit

The Deep Learning Toolkit will be placed within the intra-factory scenario and will be a standalone component. Its functioning will be described in details in D5.3, 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-1, related to the predictive maintenance scenario. In this use case the Deep Learning Toolkit is expected to receive its inputs from a heterogeneous sensor 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 explained in literature, the Artificial Neural Networks that form 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 historical data provided by the Boston Scientific end user have been clusterized 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 (BLS-1) and are representative of measured temperatures, expected temperatures and output powers. Being these measures coming from sensors-based detections, they have been organized in data triplets related to measurement areas, resulting in sixty one independent fields.

Based on the above assumption, we are going to define data types that need to represent univocally encapsulated sensors arrays for sixty one clusterized linear independent heterogeneous features. At project level it has been promoted the OGC SensorThings as a standard for providing a comprehensive representation of sensors readings from the shop floor layer. As a newly developed component deployed at the same level, ISMB would really like to homogenize its data choices to the projects' data formats. At the time this document is being written, the Deep Learning Toolkit is in its draft form and a wide survey about data format possibilities is being conducted. If the homogenization of data formats and standards is going to take place, the Deep Learning Toolkit will receive and provide data and previsions in the form of an 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

result	mandatory	Any (depends on the observationType defined in the associated Datastream)
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 is a more than a suitable candidate to host its output. In fact, the component provides periodical previsions that are dispatched in retain mode, each and every time there is a new knowledge available from the Artificial Neural Networks embedded models in reaction to incoming batches of live sensors readings. The expected output could be a dichotomy based scalar value that predicts whether or not there is going to be an allegedly event of breakdown possible failure, in the next time slot evaluated by the prevision timeframe.

The standard provides two more sensible fields that are optional, but the Deep Learning Toolkit team is evaluating if and where they could find a reasonable place in the data format, 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 going to be used in the final form of the Deep Learning Toolkit will be 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. These metrics will be then aggregated and summarized, taking into consideration the previous values coming from the Artificial Neural Network. The idea is to use the *resultQuality* field 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 will be eventually used by the Deep Learning Toolkit to define a time to live expectancy validity of the 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.

An issue that the task 5.2 team is currently addressing is the lack of *Datastream* that is mandatory in the API description. When this issue will be sorted out by the Data Factory Model, it will be possible for this component as well to identify the *FeatureOfInterest* field array, in order to further enhance the output and input layer format.

An interesting extension that could be taken into consideration in order to provide a data readings array, instead of a less scalable single reading observation for each of the aggregated sensors, is the MultiDatastream. It allows grouping several measurements that happen at the very same time, like in our use case. This is made possible, by defining data arrays instead of scalar and string values in the data fields. A very simple example of an aggregated reading is provided:

```
{

        "result": [120,140,80,...]
        "phenomenonTime": "2017-01-01T00:00:00.000Z",
        “resultTime”: "2017-01-01T01:00:00.000Z",

}
```

The Deep Learning Toolkit needs to be informed by the shop floor components, namely the Big Data Analytics in the form of its Learning Agents, of any kind of warning of 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"

}
```

This information could also be used by the Deep Learning Toolkit for providing a broader range of possibilities coming from the Artificial Neural Network, without the needing of mandatory mediating its classification results in binary constrains formats. Below a simple representation of this multi-variant data output is represented.

```
    {
            "result": "0",
            "phenomenonTime": "2017-01-01T00:00:00.000Z",
            “resultTime”: "2017-01-01T01:00:00.000Z",
            “validTime”: “20080915T155300/20101113T000000”,
             “resultQuality”: ”0.36”,
            "FeatureOfInterest": {
                    "description": "Deep Learning Toolkit BLS Brady Predictive Maintenance",
                    "name": "DLT BLS Brady Prediction",
                    "encodingType": "application/vnd.geo+json",
                    -"feature": {
                            -"accuracy": [
                                    -114.133,
                                    51.08
                            ],
                    "type": "Point"
                    }
            }
}
```

Of course at this point in time the encoding type is not provided by the reference itself, so it has not been possible to find a suitable corresponding data format. In this example the geo-coordinates have been used to represent data accuracy non aggregated metrics, like minimum mean error and average maximum error. In the final form, the *feature* type could be replaced or disrupted completely during the integration phase.

It is then demanded to the next iteration of this document to complete the description in the real world scenario when the lab Docker server will be populated with all the Docker containers and all component images will be exchanging real world data.

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 formal recommendation from the World Wide Web Consortium (W3C), and it is similar to Hypertext Markup Language (HTML). Both XML and HTML contain 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, although 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. In most languages, this is realized as an array, vector, list, or sequence.

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, family of standards (ISA-95), known internationally as IEC/ISO 62264. B2MML consists of a set of XML schemas written using the World Wide Web Consortium's XML Schema language (XSD) that implement the data models in the ISA-95 standard. B2MML helps integrate 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 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 design, certification, operation, maintenance, 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 the standard schema language, XML so that major industry applications can import and export project information no matter the vendor, device, or software platform.

5.4 MIMOSA

MIMOSA (MIMOSA, 2017) is a not-for-profit trade association dedicated to developing and encouraging the adoption of open information standards for Operations and Maintenance in manufacturing, fleet, and facility environments. MIMOSA 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. It has a rich history of developing industry standards which are driven by industry requirements. MIMOSA works closely with formal standards bodies to help develop international standards reflecting industry requirements.

5.5 Business Process Model and Notation (BPMN)

Business Process Model and Notation or BPMN (BPMN, 2017) is a standard for business process modelling that will provide businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standard manner. The graphical notation specifies business processes 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, by providing a notation that is intuitive to business users, yet able to represent complex process semantics. The BPMN specification also provides a mapping between the graphics of the notation and the underlying constructs of execution languages, particularly Business Process Execution Language (BPEL).

The primary goal of BPMN is to provide a standard notation readily understandable by all business stakeholders. These include the business analysts who create and refine the processes, the technical developers responsible for implementing them, and the business managers who monitor and manage them. Consequently, BPMN serves as a common language, bridging the communication gap that frequently occurs between business process design and implementation. Since 2014, BPMN has been complemented by a new standard for building decision models, the Decision Model and Notation standard.

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. Multiple companies, government agencies and universities participate in OGC consensus process 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 approved Open Geospatial Consortium standard. SensorML provides standard models and an XML encoding for describing any process, including the process of measurement by sensors and instructions for deriving higher-level information from observations. SensorML can be used to describe a wide range of sensors, including both dynamic and stationary platforms and both in-situ and remote sensors.

Processes described in SensorML are discoverable and executable. All processes define their inputs, outputs, parameters, and method, as well as provide relevant metadata. SensorML models detectors and sensors as processes that convert real phenomena to data.

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, 2017) is an international OGC Standard. This standard specifies an XML implementation for the OGC and ISO Observations and Measurements (O&M) conceptual model, including a schema for Sampling Features. The origins of the O&M are in the Sensor Web Enablement (SWE) initiative of the OGC. Together with other SWE framework open standards like SensorML and Sensor Observation Service (SOS), 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. O&M standard defines XML schemas for observations and features involved in sampling when making observations. These provide document models for the exchange of information describing observation acts and their results, both within and between different scientific and technical communities.

5.6.3 SensorThings

SensorThings (SensorThings, 2017) is an OGC standard provides an open, geospatial enabled, unified way to interconnect Internet of Things (IoT) devices, data and applications over the Web. SensorThings is a non-proprietary, platform-independent, and perpetual royalty-free standard. It builds on a rich set of proven-working and widely-adopted 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. It provides functions similar to the OGC Sensor Observation Service. Tasking Profile provides a standard way for parameterizing - also called tasking - of task-able IoT devices, such as sensors or actuators. 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 the de facto standard to represent predictive solutions. PMML is developed by the Data Mining Group (DMG), a consortium of commercial and open-source data mining companies. 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 allows for different statistical and data mining tools to speak the same language. In this way, a predictive solution can be easily moved among different tools and applications without the need for custom coding. PMML works as the common denominator, since whenever the solution is built, it is immediately represented as a PMML file. This allows companies to use first-rate tools to build the best possible solutions.

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) were analysed. The main requirements related to modelling tasks and DFM as well are presented to the below table.

Table 4: 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

3. *The analysis of the use cases*. As the requirements were in an initial state and the project was in an early phase we further analysed all the use cases which were related to modelling tasks. As the requirements were in abstract level of details, the thorough analysis of the use cases provided us with some more concrete requirements.
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. The XML was selected as the language for the description of data.

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 an XSD schema. The selected standards from step 5 were imported to the DFM schema and were used for the schema's development.

The XML format was selected as the data type for both dynamic and static data descriptions. It is a supported format from COMPOSITION system architecture. Moreover, 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 XML format was adopted in current version of DFM in order to be in a common format with the static data representations. Nevertheless, during technical discussions among the consortium the OGC SensorThings standard 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. This fact will lead to an unnecessary transformation from JSON to XML for a huge number of cases. So, in order to avoid this transformation the replacement of XML format from JSON format for dynamic data description only will be performed. As the replacement was decided just the last month before this report and it is still an ongoing process, the XML format for the dynamic data representation is presented here as it reflects the work have done until this report. Moreover the OGC Observation and Measurement will not be replaced as a standard but only its format will be changed. All the changes will be reflected in the second and final part of this deliverable at M26.

The figure below presents a high level structure of the implemented DFM schema. The XSD Diagram tool (XSD Diagram, 2017) is used for the figure's exportation. It is a free Xml Schema Definition diagram viewer and it is used in this report for XSD diagrams' representations.

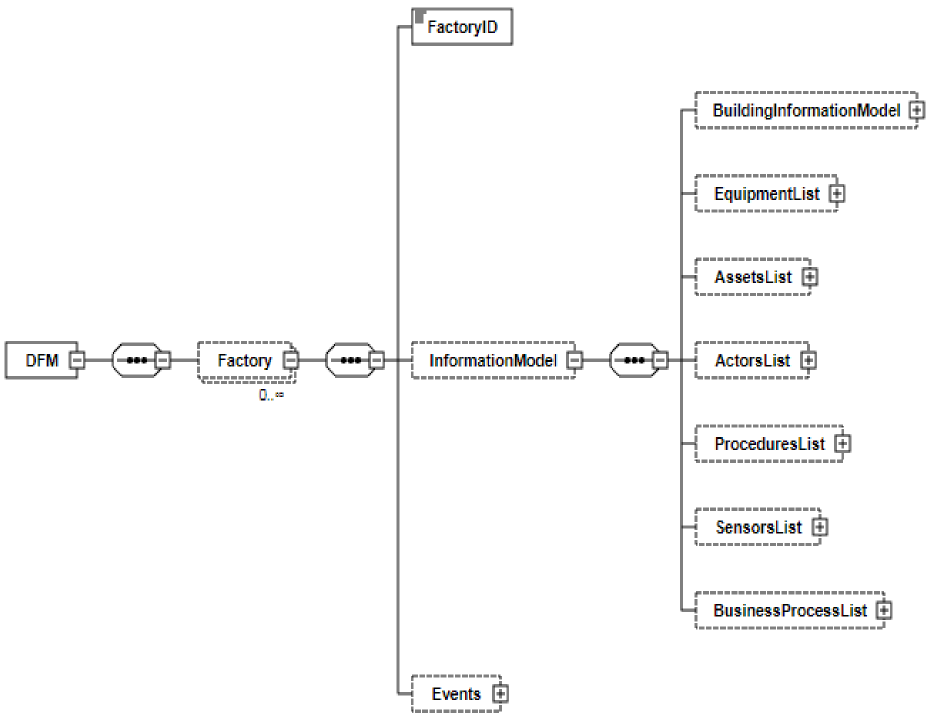


Figure 2: High level structure of DFM

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. 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 2 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 2 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
- Sensors 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, UC-BSL-7 Automatic long term tracking of high value material 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.

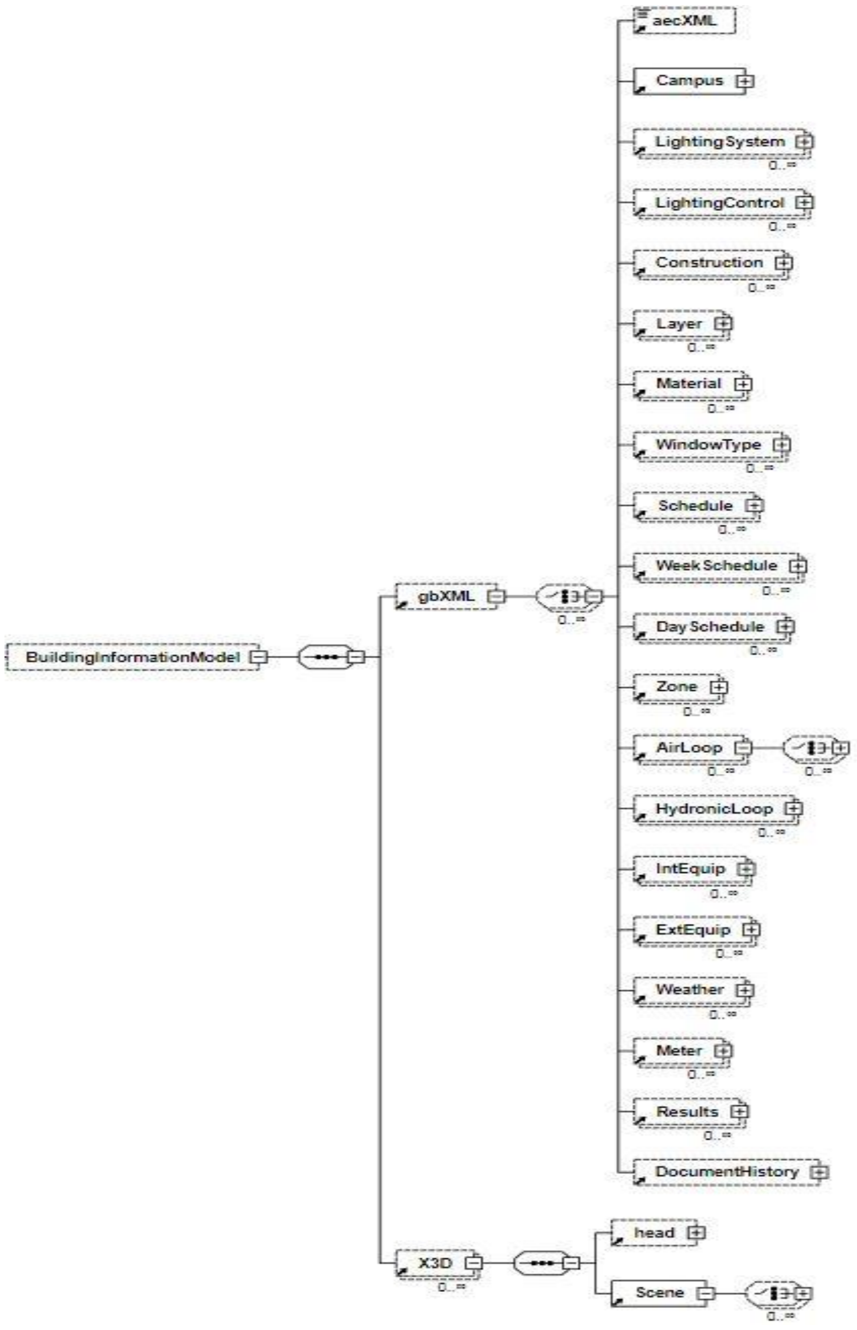


Figure 3: Building Information Model schema

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:

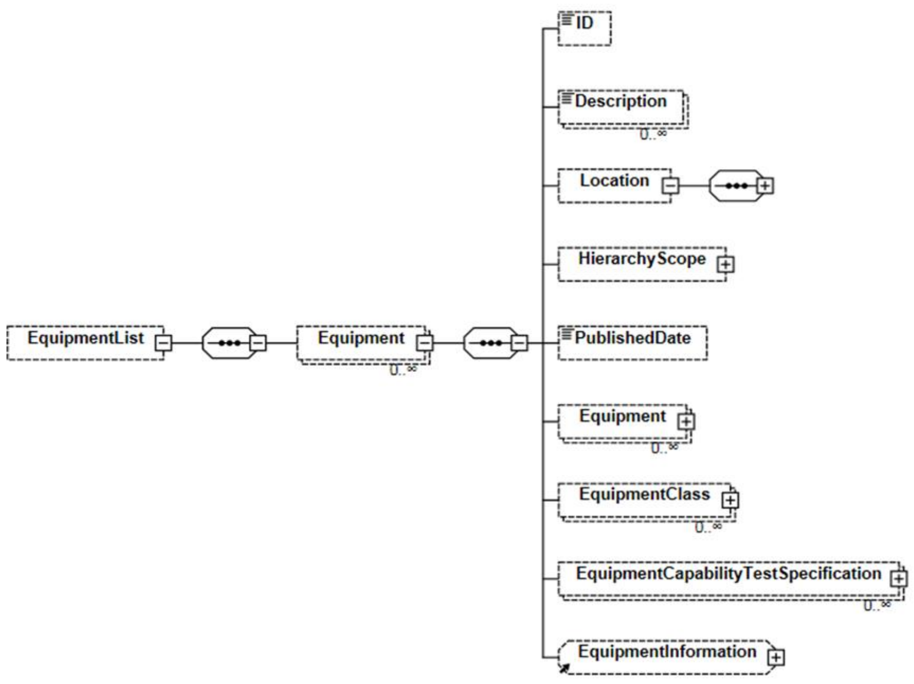


Figure 4: Equipment List schema

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.

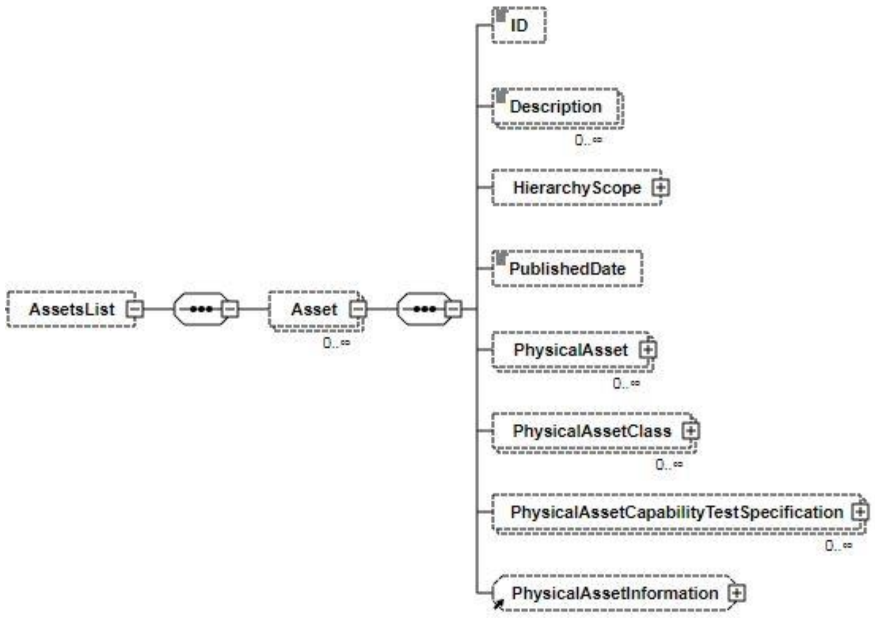
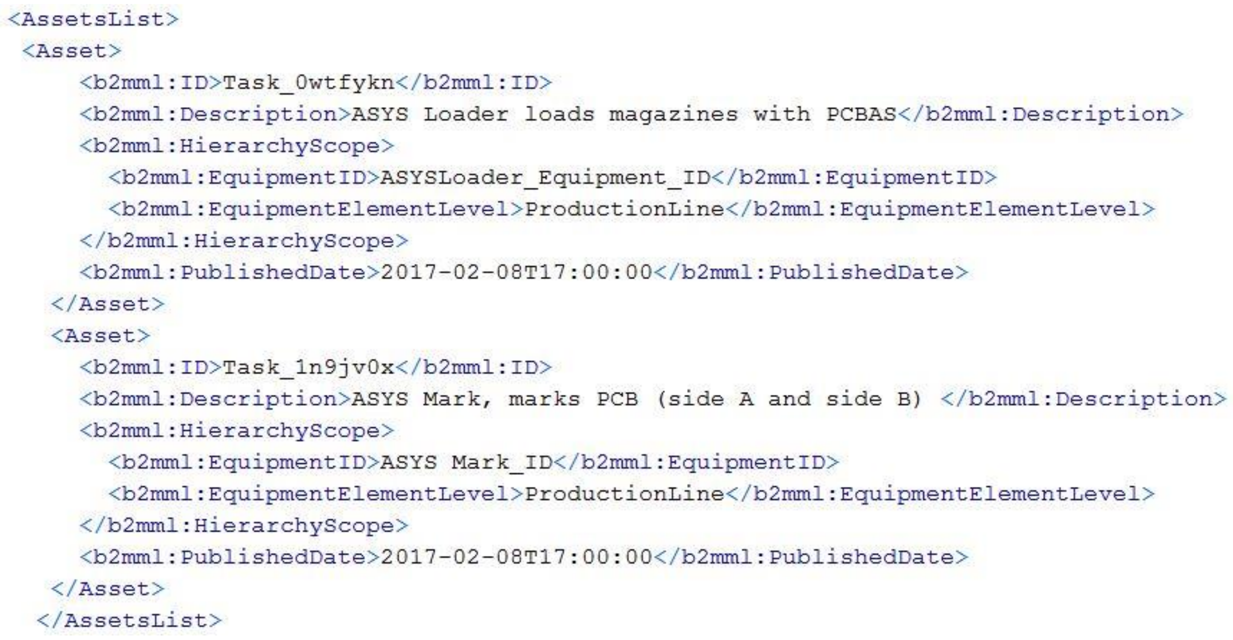


Figure 5: Assets List schema

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



```
<AssetsList>
 <Asset>
     <b2mml:ID>Task_0wtfykn</b2mml:ID>
     <b2mml:Description>ASYS Loader loads magazines with PCBAS</b2mml:Description>
     <b2mml:HierarchyScope>
       <b2mml:EquipmentID>ASYSLoader_Equipment_ID</b2mml:EquipmentID>
       <b2mml:EquipmentElementLevel>ProductionLine</b2mml:EquipmentElementLevel>
     </b2mml:HierarchyScope>
     <b2mml:PublishedDate>2017-02-08T17:00:00</b2mml:PublishedDate>
   </Asset>
   <Asset>
     <b2mml:ID>Task_1n9jv0x</b2mml:ID>
     <b2mml:Description>ASYS Mark, marks PCB (side A and side B) </b2mml:Description>
     <b2mml:HierarchyScope>
       <b2mml:EquipmentID>ASYS Mark_ID</b2mml:EquipmentID>
       <b2mml:EquipmentElementLevel>ProductionLine</b2mml:EquipmentElementLevel>
     </b2mml:HierarchyScope>
     <b2mml:PublishedDate>2017-02-08T17:00:00</b2mml:PublishedDate>
   </Asset>
  </AssetsList>
```

Figure 6: Example of Assets List's XML syntax

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.

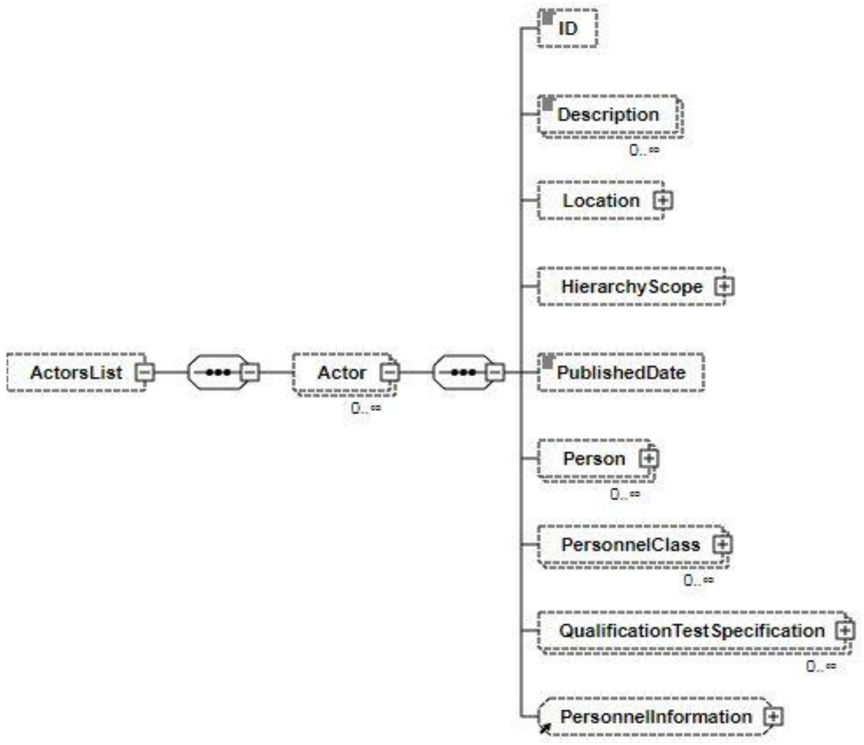


Figure 7: Actors List schema

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.

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 8. As depicted in the figure the structure of this schema is similar to the rest schemas were provided by B2MML standard.

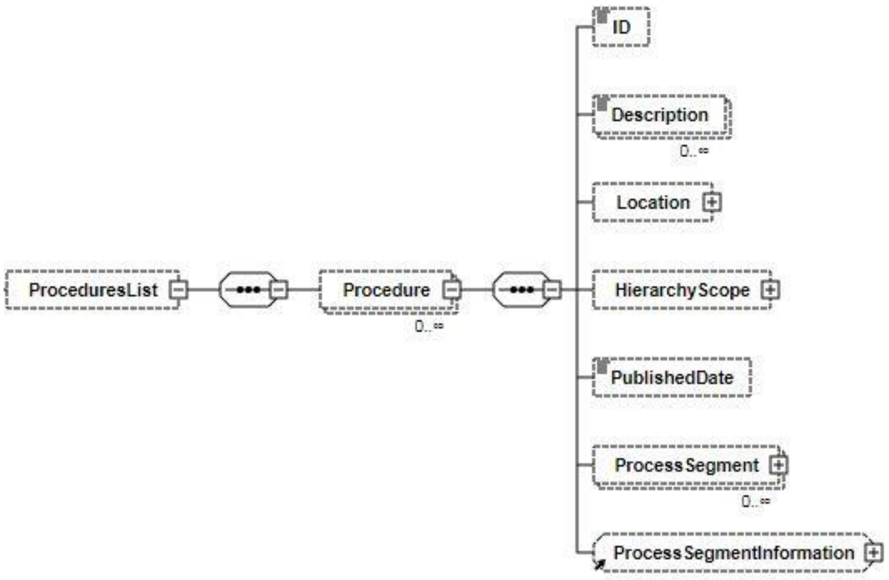


Figure 8: Procedures List schema

6.2.2.6 Sensors List

The Sensors List of the DFM aims to describe all the sensors installed in the factory and they are related with the COMPOSITION's use cases. Sensors will be used almost in all intra-factory use cases of this project. This fact indicates that the modelling of sensors is mandatory. 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 will be modelled.

The Equipment List or Assets List could be used for the description of the installed sensors as the B2MML standard does not contain a structure dedicated to sensors. However, a more dedicated solution for sensors' description was selected.

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.

The next figure presents the schema structure of the DFM's Sensors List.

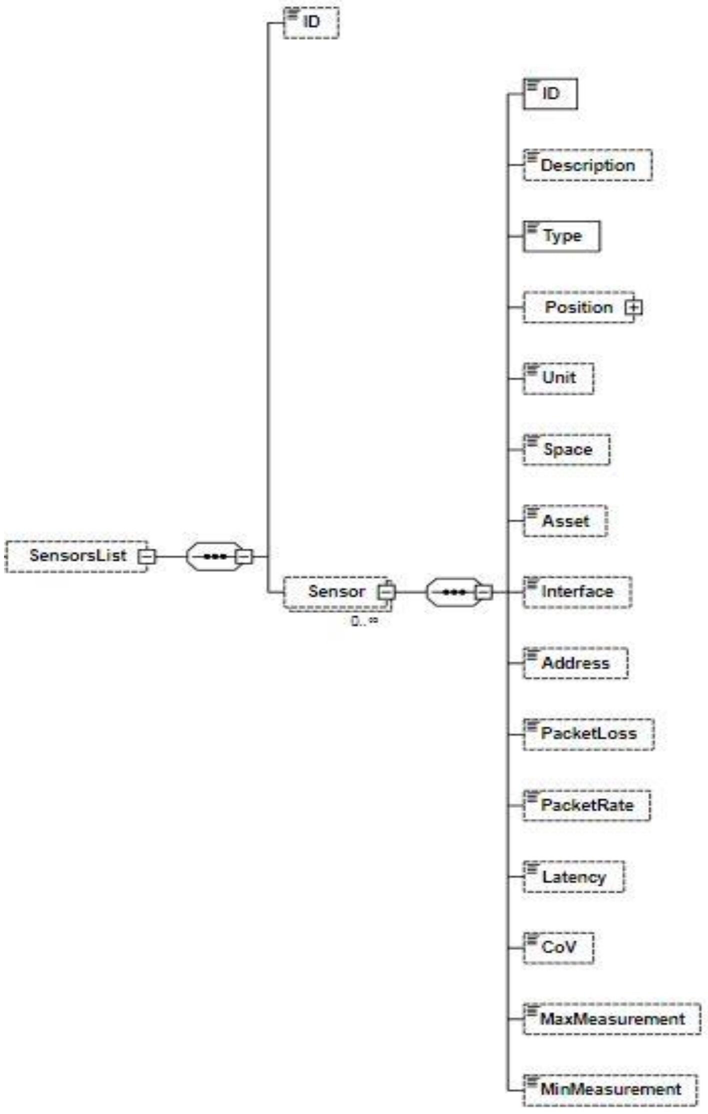


Figure 9: Sensors List schema

6.2.2.7 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:

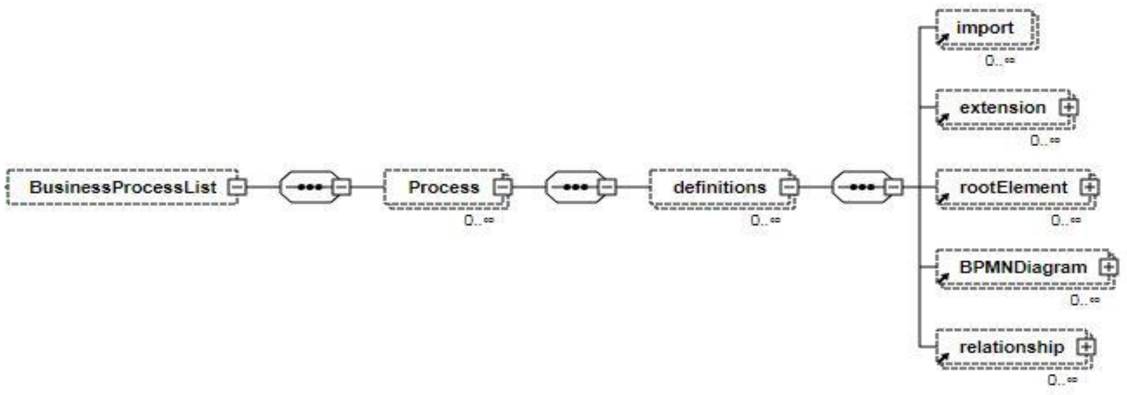


Figure 10: Business Process List schema

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 Events structure in a DFM instance contains all the dynamic data such as sensors' measurements, events and alerts.

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 XML 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. 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.

An example of an Event representation related to a bin's fill level monitoring about UC-KLE-3 Scrap metal and recyclable waste transportation (from bins to container) is described above.

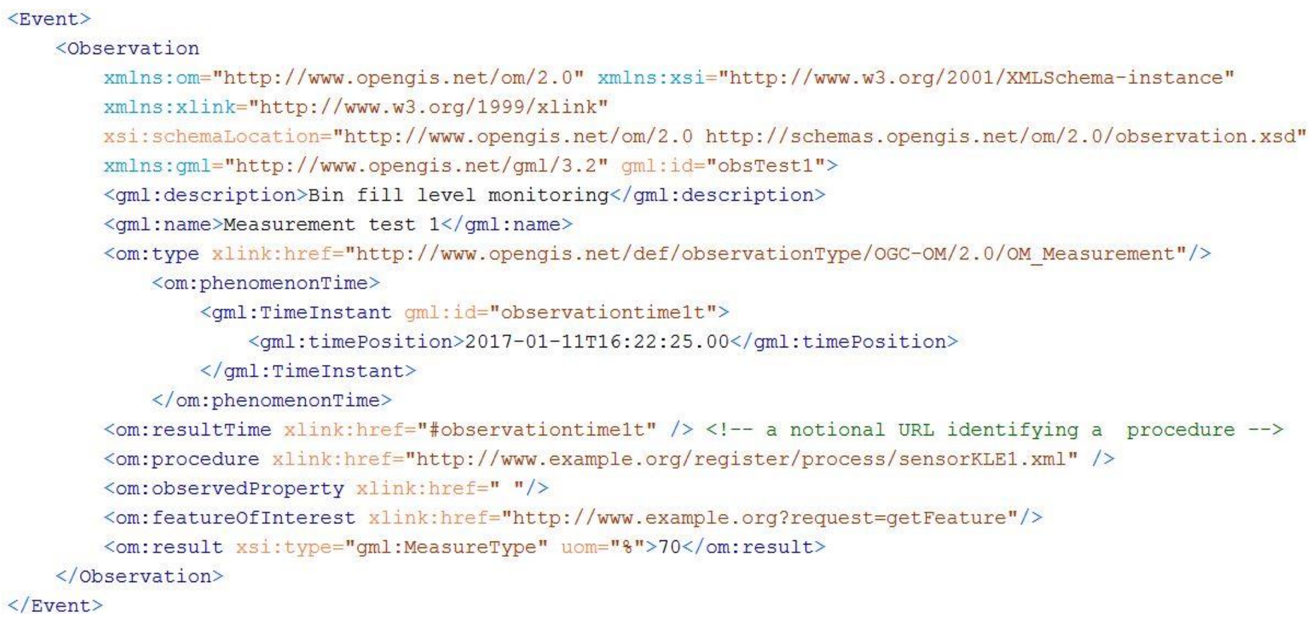

```xml
<Event>
    <Observation
        xmlns:om="http://www.opengis.net/om/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xsi:schemaLocation="http://www.opengis.net/om/2.0 http://schemas.opengis.net/om/2.0/observation.xsd"
        xmlns:gml="http://www.opengis.net/gml/3.2" gml:id="obsTest1">
        <gml:description>Bin fill level monitoring</gml:description>
        <gml:name>Measurement test 1</gml:name>
        <om:type xlink:href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement"/>
            <om:phenomenonTime>
                <gml:TimeInstant gml:id="observationtime1t">
                    <gml:timePosition>2017-01-11T16:22:25.00</gml:timePosition>
                </gml:TimeInstant>
            </om:phenomenonTime>
        <om:resultTime xlink:href="#observationtime1t" /> <!-- a notional URL identifying a  procedure -->
        <om:procedure xlink:href="http://www.example.org/register/process/sensorKLE1.xml" />
        <om:observedProperty xlink:href=" "/>
        <om:featureOfInterest xlink:href="http://www.example.org?request=getFeature"/>
        <om:result xsi:type="gml:MeasureType" uom="%">70</om:result>
    </Observation>
</Event>
```

Figure 11: Example of Event's XML syntax

The next figure presents a high level view of DFM's Events based on OGC Observations and Measurements schema:

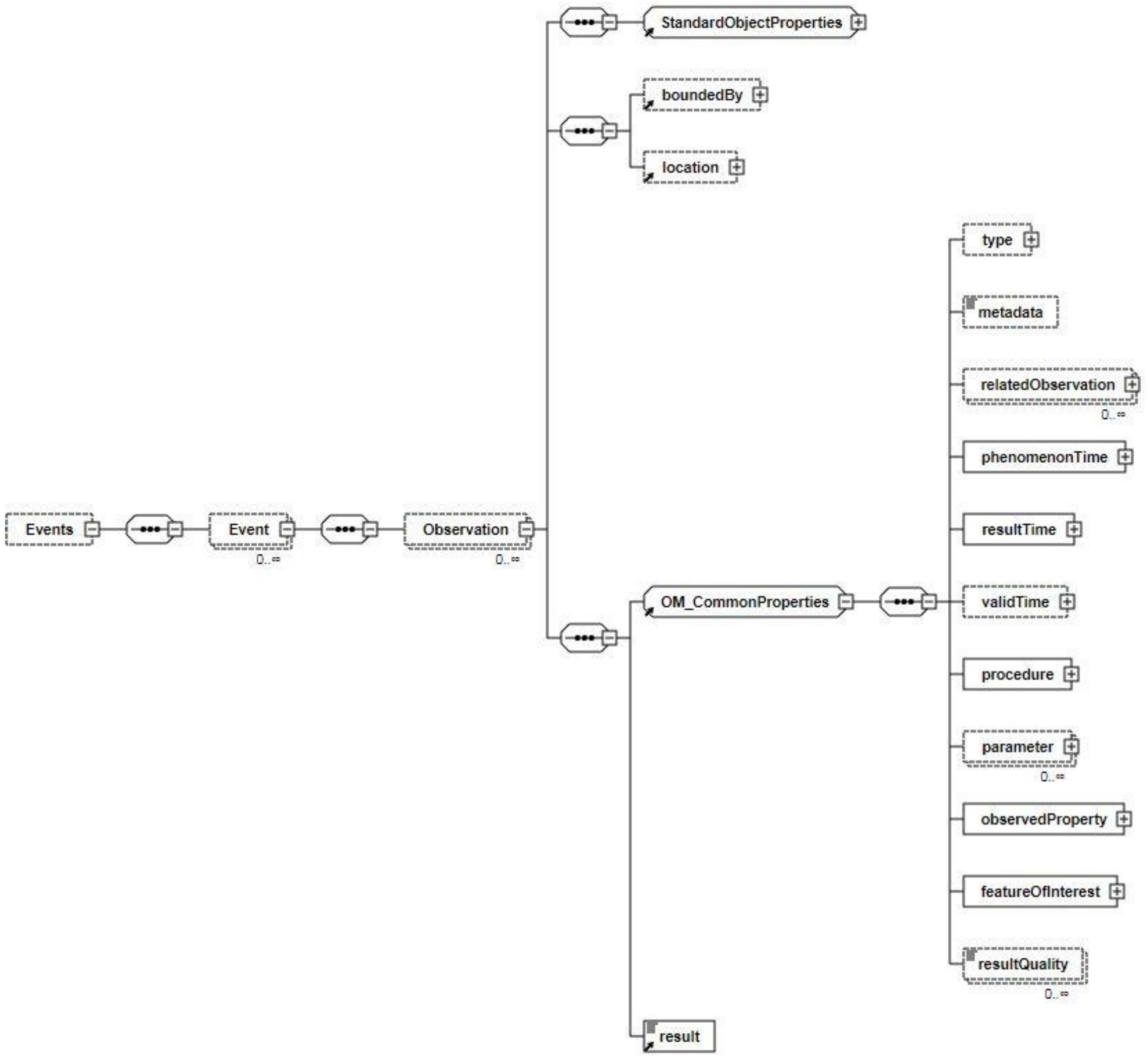


Figure 12: Events schema

7 Digital Factory Model API

As described in the executive summary and introductory sections besides the Digital Factory Model schema, a first version of 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 Data Persistence store 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. Based on section 4 of this report, the DSS, 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 case
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

Requirements

The development of DFM API was driven by COMPOSITION requirements. The initial requirements based on D2.2-Initial requirements specification were mainly related to the modelling processes and the DFM schema and not to the DFM API specifications. Nevertheless, in later stages of the project and after the definition of the current version of COMPOSITION system's proposed architecture as well, the implementation of the DFM API was considered as necessary. Thus, based on the architecture definition and the project's use cases, the following main requirements were set for the DFM API implementation:

- The DFM API should connect the IIMS components with the common data store
- The API should offer the following main categories of functionalities to IIMS components:
  - Get/read data for both static and dynamic information of a DFM instance
  - Set/store data related to static and dynamic information of a DFM instance
  - Delete/remove data related to static and dynamic information of a DFM instance
- The API should ensure that the data exchange will be based on the DFM schema
- The connection should be based on communication protocols and formats accepted from COMPOSITION system's architecture
- It should be well designed and compatible with project's quality control
- It should be designed in a way to be easily extended in order to capture the future project's requirements

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.

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 format is XML. The XML format 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:

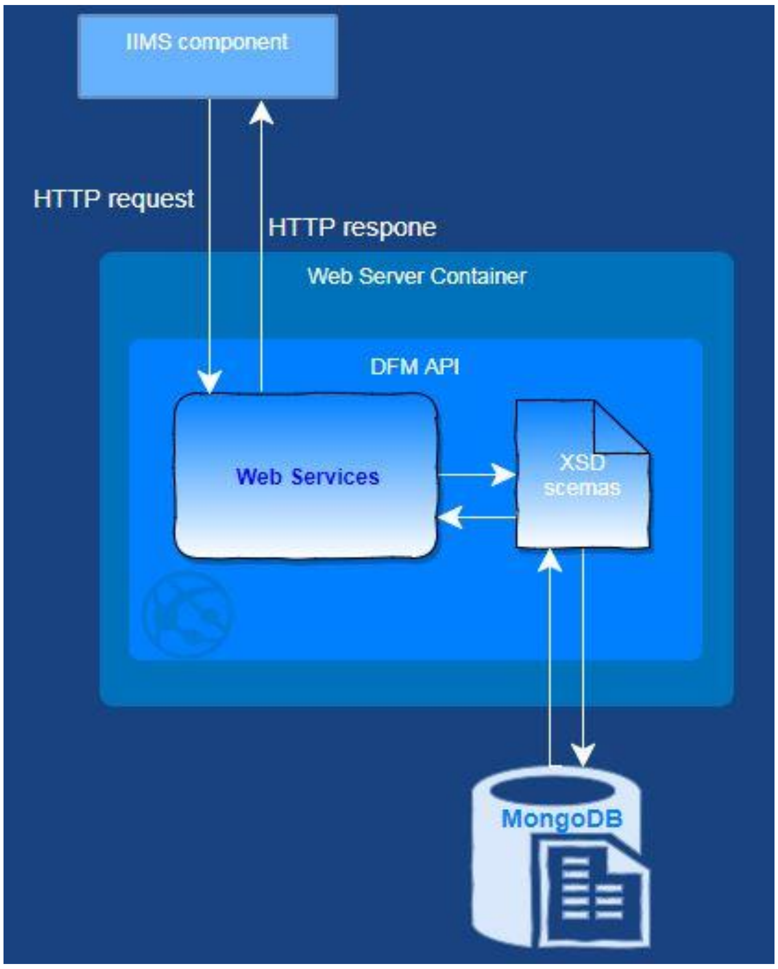


Figure 13: High level overview of DFM API architecture

Testing

After the development of the first version 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, 2017). All the calls results were evaluated and any problems related to supported functionalities were fixed.

DFM Instances

A DFM Instance related to BSL's PCBAs production line has been created. It is implemented using the DFM API's current version. 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 this data is 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. This DFM Instance will be updated with more static information and with dynamic information as soon as the sensors components will be deployed to the shop-floor.

7.1.2 Supported Interfaces

This section presents the current 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 current 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 *getFactoryInformation* (String factoryID, boolean zip)

  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. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding factory's static information. Else, the returned information will be an XML file in text form.

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

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 *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 *getBuildingInformation*(String factoryID, boolean zip)

  It retrieves from the database all the buildings information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding building information. Else, the returned information will be an XML file in text form.

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

- String *getBuildingByID* (String factoryID, String buildingID, boolean zip)

  It retrieves from the database the information of a specific building stored in a DFM instance based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding building information. Else, the returned information will be an XML file in text form.

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

- String *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 *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 *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 *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 *getEquipmentList*(String factoryID, boolean zip)

  It retrieves from the database all the equipment information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the installed equipment. Else, the returned information will be an XML file in text form.

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

- String *getEquipmentByID* (String factoryID, String equipmentID, boolean zip)

  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. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding equipment component information. Else, the returned information will be an XML file in text form.

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

- String *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 *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 *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 *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 *getAssetsList*(String factoryID, boolean zip)

  It retrieves from the database all the assets information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory's assets. Else, the returned information will be an XML file in text form.

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

- String *getAssetByID* (String factoryID, String assetID, boolean zip)

  It retrieves from the database the information of a specific asset related to a DFM instance based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding asset component's information. Else, the returned information will be an XML file in text form.

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

- String *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 *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.

Actors List

- String *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 *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 *getActorsList*(String factoryID, boolean zip)

  It retrieves from the database all the actors information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory's actors. Else, the returned information will be an XML file in text form.

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

- String *getActorByID* (String factoryID, String actorID, boolean zip)

  It retrieves from the database the information of a specific actor related to a DFM instance, based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding actor's information. Else, the returned information will be an XML file in text form.

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

- String *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 *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 *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 *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 *getProceduresList*(String factoryID, boolean zip)

  It retrieves from the database all the procedures' information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory's procedures. Else, the returned information will be an XML file in text form.

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

- String *getProcedureByID* (String factoryID, String procedureID, boolean zip)

  It retrieves from the database the information of a specific procedure related to a DFM instance, based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding procedure's information. Else, the returned information will be an XML file in text form.

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

- String *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 *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.

Sensors List

- String *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 *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 *getSensorsList*(String factoryID, boolean zip)

  It retrieves from the database all the sensors' information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory's sensors. Else, the returned information will be an XML file in text form.

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

- String *getSensorByID* (String factoryID, String sensorID, boolean zip)

  It retrieves from the database the information of a specific sensor related to a DFM instance, based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding sensor's information. Else, the returned information will be an XML file in text form.

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

- String *deleteSensorsList* (String factoryID)

  It deletes based on the factory ID, the complete list of the sensors 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 *deleteSensor*(String factoryID, String sensorID)

  It deletes the information from the database related to a specific sensor, 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 sensorID are URL parameters.

Business Processes List

- String *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 *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 *getBusinessProcessList*(String factoryID, boolean zip)

  It retrieves from the database all the BPMN diagram's information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory's business processes. Else, the returned information will be an XML file in text form.

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

- String *getBusinessProcessByID* (String factoryID, String bpmnID, boolean zip)

  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. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding BPMN diagram's information. Else, the returned information will be an XML file in text form.

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

- String *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 *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 of a factory. The following interfaces are related to dynamic data from DFM instances:

- String *setEventList* (String eventXML)

  It sends an XML 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 from the posted XML file, and a list of information about the factory’s events 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 list with some factory’s events is the body of the request.

- String *setEvent* (String eventXML)

  It sends an XML 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 from the posted XML file, and the event’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 event is the body of the request.

- String *getEventList*(String factoryID, boolean zip)

  It retrieves from the database the entire event’s information stored in a DFM instance based on the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the complete list of the factory’s events. Else, the returned information will be an XML file in text form.

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

- String *getEventByID* (String factoryID, String eventID, boolean zip)

  It retrieves from the database the information of a specific event related to a DFM instance, based on its ID and the factory ID. If the value of the boolean parameter zip is true, then the response will return a zipped file which contains the XML file with the corresponding event’s information. Else, the returned information will be an XML file in text form.

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

- String *deleteEventList* (String factoryID)

  It deletes based on the factory ID, the complete list of the events 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 *deleteEvent*(String factoryID, String eventID)

  It deletes the information from the database related to a specific event, 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 eventID are URL parameters

8 Next Steps

The future work at Task 3.2 Integrated Digital Factory Models will be mainly focused at procedures related to:

- Extension and completion of the DFM instances related to intra-factory use cases of KLE and BSL. More information related to the static data such as actors, machines and equipment will be added to DFM instances based on the DFM schema format. As soon as, sensors components will be installed at the shop-floors, their measurements will start to be added at the corresponding DFM instances. Last, as the Task 3.1 Process Modelling and Monitoring Framework will be finished by M18, all the exported BPMN diagrams of this task will be inserted at the Business Process List of the DFM.
- The replacement of OGC Observation and Measurements XML standard with the corresponding JSON format standard. The use of OGC SensorThings and its corresponding JSON format related to Observations and Measurements for the DFM's events descriptions is an ongoing process as this approach selected lately to be followed. As defined during technical discussions, the JSON format for the dynamic data description covers better the project's needs and it is a more compatible format for many of the related IIMS components as the OGC SensorThings used by BMS which transfers sensors data to Event Broker and then to the rest of the IIMS components. An example of this kind of data representation was presented at section 4.2.6 – Deep Learning Toolkit of this report. As soon as this approach will be followed, the necessary changes in the DFM API and more precisely, in web services related to the Events will be done.
- Possible extension of the DFM schema (if this needed) with more standards in order to cover future needs of the project related to modelling of processes or resources.
- Further extension of the COMPOSITION DFM API's supported web services in order to cover all the IIMS components' needs for data storage or data retrieval.
- Modifications to implementation in order to be fully compatible with project's security requirements and the implemented Security Framework from WP4.
- The creation of a Docker image for DFM API and the deployment of this image to the common COMPOSITION deployment container.

The results of the previous mentioned future procedures will be reflected at the next versions of both DFM schema and DFM API. Finally all the work will be done in Task 3.2 Integrated Digital Factory Models will be presented in D3.3 Digital Factory Model II in M26.

9 Conclusions

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 M15 in this task. Furthermore, this report documents the current versions of COMPOSITION's DFM schema and DFM API. The complete work of Task 3.2 will be presented in D3.3 Digital Factory Model II in M26.

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 XML format 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 Data Persistence store. The data exchange format should be valid against the DFM schema.

The outcome of this deliverable mainly affects the WP3 and its components such as 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 current status 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 should be further extended. The next steps of Task 3.2 are described at Chapter 9 of this report and describe the future actions related to DFM tasks in order to meet all the project's requirements and needs. By the end of Task 3.2 the DFM schema will get its final form, the catalogue of the DFM API's supported interfaces will be extended and all the necessary DFM instances related to the pilot cases will be completed.

10 List of Figures and Tables

10.1 Figures

10.2 Tables

11 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 D2.3, 2017) COMPOSITION EU Project: D2.3 The COMPOSITION architecture specification I http://www.composition-project.eu/downloads/D2.3%20The%20COMPOSITION%20architecture%20specification%201.1.pdf

(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

(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, 2017) 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, 2017) Postman web site: https://www.getpostman.com/

Annex

The following link DFM-Doc-link provides a documentation of the COMPOSITION DFM schema in HTML representation. This documentation aims to describe the following xsd files:

- COMPOSITION DFM schema
- Composition Common schema which contains some common types and elements definitions for the DFM schema's structure
- B2MML schema which contains some modifications from the original schema as they introduced from SatisFactory EU project and contains some extra types for a more detailed representation of the factory's static information.
    - The original B2MML schema is documented in details:

      B2MML–v0600 documentation and schema: https://services.mesa.org/ResourceLibrary/ShowResource/0f47758b-60f0-40c6-a71b-fa7b2363fb3a

The rest of the imported schemas are used in their original forms. The detailed representation and description of the used existing standards are available in the corresponding web pages:

- OGC Observations and Measurements XML Implementation Documentation : http://portal.opengeospatial.org/files/?artifact_id=41510
- gbXML version 5.12 :
    - schema: http://www.gbxml.org/schema/5-12/GreenBuildingXML_Ver5.12.xsd.txt
    - documentation: http://www.gbxml.org/schema/5-12/GreenBuildingXMLv.5.12.Ref.zip
- BPMN20:
    - schema: http://www.omg.org/spec/BPMN/20100501/BPMN20.xsd
    - documentation and imported schemas: http://www.omg.org/spec/BPMN/2.0/
- x3d 3.3:
    - schema: http://www.web3d.org/specifications/x3d-3.3.xsd
    - documentation and imported schemas: http://www.web3d.org/specifications/#AutogeneratedProducts