In2Rail

Project Title:	**INNOVATIVE INTELLIGENT RAIL**
Starting date:	01/05/2015
Duration in months:	36
Call (part) identifier:	H2020-MG-2014
Grant agreement no:	635900

Deliverable D8.3

Description of Integration Layer and Constituents

Due date of deliverable	Month 36
Actual submission date	17-04-2018
Organization name of lead contractor for this deliverable	BT
Dissemination level	PU
Revision	Final

Authors

		Details of contribution
Author(s)	**BOMBARDIER TRANSPORTATION (BT)** Roland Kuhn (DE) Zbigniew Dyksy (PL) Martin Karlsson (SE)	Coordination and document structure of D8.3 Contribution to chapters: 1-10 Authors to chapters: 1-3, 4.1, 4.7, 4.11.1, 4.11.2, 5, 6.1, 6.2, 6.3, 6.4, 6.6, 6.7,6.8, 7.1, 7.3, 7.6, 8.1, 8.2, 8.3, 8.4, 9
Contributor(s)	**Ansaldo STS (ASTS)** Gian Luigi Zanella (I) Matteo Pinasco (I)	Authors to chapters : 4.2, 4.3, 4.4, 6.6.6, 7.5, 8.5 Contribution to chapters : 1-10
	AZD Praha s.r.o (AZD) Martin Bojda (CZ) Michal Zemlicka (CZ)	Authors to chapters: 7.4 Contribution to chapters: 1-10
	CAF Signalling (CAF) Carlos Sicre Vara de Rey (ES)	Authors to chapters: 4.11.3 Contribution to chapters: 1-10
	HaCon (HC) Sandra Kempf (DE) Rolf Gooßmann (DE)	Authors to chapters : 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 Contribution to chapters : 1-10
	SIEMENS (SIE) Stefan Wegele (DE)	Authors to chapters: 4.12, 6.6.4, 6.6.5, 7.2 Contribution to chapters : 1-10
	Thales (THA) Jean-Jacques Rodot (F) Jean-Yves Friant (F) Vishal Bhatt (UK) Christoph Bücker (DE)	Authors to chapters: 4.9, 4,13, 6.5 Contribution to chapters: 1-10

Executive Summary

The overall aim of the In2Rail project is to set the foundation for a resilient, cost-efficient, high capacity, and digitalised European rail network.

There are three In2Rail Work Packages relating to Intelligent Mobility Management (I2M), one of which is WP8 addressing the first system design of a new ICT.

The main innovation is the communication platform (Integration Layer) using standardized data structures and processes to manage the Communication/Data exchange between different services/clients and supporting TMS applications connected to other multimodal operational systems.

The Integration Layer links in the first step, Traffic, Asset and Energy Management Systems, field infrastructure and vehicles. It also provides a gateway for the communication with external clients to receive updates of the different services e. g. Traffic status and to receive demands impacting the planning of the different services.

The proposed architecture, interfaces and the data model allow the integration of legacy installations through APIs (Application Programming Interfaces) handling the data and function mapping being essential guaranteeing an acceptance of the various stakeholders involved in Traffic Control, Traffic Management and other Rail Business Services.

The communication between clients is changing from a Point2Point principle to "Publish & Subscribe" methodology. This allows to apply new possible scenarios for traditional operational processes including how to manage the field signalling infrastructure.

The evaluation of requirements for the ICT has shown that a data centric approach fits best to the defined needs and for the middleware of such communication infrastructure the In-Memory Data Grid (IMDG) is the most promising candidate.

The generic design of a scalable and flexible Canonical Data Model (CDM) needed to secure the integration of legacy and future applications into one communication structure has evolved and is proposed to be developed further to a "Shift2Rail" Data Model. Such a data-format is required to secure seamless and automated data exchange between the different clients and to integrate status data from various sources into the decision-making processes.

Integrated communication processes and permanent availability of status data in one common layer will significantly improve the efficiency of decision processes of all linked rail services. This is key to achieve the targets for capacity growth, reliability improvement and cost reduction.

The Integration Layer support all the different new technologies developed under the S2R program such as ATO, Moving Block (IP2), new advanced Maintenance Strategies (IP3), Freight Transport (IP5) and Passenger Information services (IP4).

TABLE OF CONTENTS

Abbreviations and acronyms

Abbreviation / Acronyms	Description
AF	Application Framework
API	Application Programming Interface
ATO	Automated Traffic Operation
ATP	Automated Train Protection
BSD	Berkeley Software Distribution (licence)
CAPEx	Capital expenditure or capital expense (capex) is the money a company spends to buy, maintain, or improve its fixed assets
CCS	Control Command and Signalling system
CDM	Canonical Data Model
CENELEC	European Committee for Electrotechnical Standardization (fr. Comité Européen de Normalisation Electrotechnique)
CIM	Computation Independent Model
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTC	Centralized Traffic Control
DCPS	Data-Centric Publish-Subscribe
DDS	Data Distribution Service – as OMG standard
DOM	Document Object Model
DTD	Document Type Definition for XML
EMF	Eclipse Modeling Framework
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
EULYNX	European Initiative Linking Interlocking Subsystems
GC	Garbage Collector
GSM-R	Global System for Mobile Communications - Railway
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technologies
IP	Internet Protocol
IL	Integration Layer
IM	Infrastructure Manager
IMDG	In Memory Data Grid
INESS	INtegrated European Signalling System
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
JAAS	Java Authentication and Authorization Service
JMX	Java Management Extensions
JMS	Java Messaging Service
JSON	Javascript Object Notation
JVM	Java Virtual Machine

Abbreviation / Acronyms	**Description**
KPI	Key Performance Indicator
MCPS	Message Centric Publish-Subscribe
MDA	Model Driven Architecture
MOM	Message Oriented Middleware
MW	Middleware
OOD	Object Oriented Design
OPEX	An operating expense, operating expenditure, operational expense, operational expenditure or OPEX is an ongoing cost for running a product, business, or system
PIM	Platform Independent Model
PSM	Platform Specific Model
QoS	Quality of Service
RBC	Radio Block Center
REST	Representational State Transfer
RINF	European Register of Infrastructure
RTM	RailTopoModel
RU	Railway Undertaking
S2R	Shift2Rail
SCI-CC	Standard Communication Interface - Command and Control
SIL	Safety Integrity Level
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SRS	System Requirements Specification
TAF	Telematic Applications for Freight
TAP	Telematic Applications for Passenger Services
TCP	Transmission Control Protocol
TMS	Traffic Management System
UDP	User Datagram Protocol
UIC	International Union of Railways (fr. Union Internationale des Chemins de fer)
UML	Unified Modelling Language
WAN	Wide Area Network
WP7	Work Package 7: System Engineering of Intelligent Mobility Management (I^2M) of In2Rail.
WP8	Work Package 8: Integration Layer of Intelligent Mobility Management (I^2M) of In2Rail.
WP9	Work Package 9: Intelligent Mobility Management (I^2M) - Nowcasting and Forecasting
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSD	XML Schema Document
XSLT	XSL Transformations, Extensible Stylesheet Language Transformations

1. WP 8 – Integration Layer

1.1. Objectives

The overall objective of the WP8 Task1 "Integration Layer" is to define a scalable and interoperable Data Layer providing the data exchange between internal Rail Operation Services in the first step TMS, Asset Management and external services either leveraging on the available Information and/or sourcing required Information e.g. weather forecast to the Integration Layer.

For the first-time data, provided from the different connected applications, clients and services will be available on one "Data-Highway" – the Integration Layer. These Information will have a standardized structure and will follow an overall Data Model. The system has standardized Interfaces, will support High-speed Data transmission and can be extended in the future for Topics (Data-lines) carrying specific Information of different complementing services /clients.

The design comprises in a first step the requirements of TMS, Asset and Energy Management, conventional and Moving Block/ETCS L3 Operation with focus to reduce complexity of technical solutions needed to be implemented on-board/trackside. Wayside subsystems shall be directly connected to the Integration Layer via standardized Interfaces (APIs).

A much higher efficiency of Traffic Regulation processes and the dynamic adaptation of disposition of available resources to dynamic demand changes will be a result of the permanent availability of the different Data for the Business Logic of the different Services.

The works executed under WP8 Task 1 have been focussed on following Objectives:

- definition of an Information and Communication Technology (ICT) environment to integrate all transport operational systems through standardized open interfaces to increase capacity, reliability and to reduce cost of Rail Operations;
- secure the provision of information available in the system via Web-Services to external clients to generate and enhance support of new services and businesses around the Rail Transportation segment;
- enable communication between TMS and CCS Field Infrastructure to enable and support new functionalities such as ATO, Moving Block or Energy Optimizer which will be developed under S2R to be applied in future rail traffic operations;
- enable the automation of business services processes together with integration of asset status information of wayside infrastructure, trains, maintenance service and energy resource management and other further data sources will enable automated decision-making processes contributing to increase the line capacity by the

optimization of route selection and traffic flow and will reduce delay minutes through a faster recovery process after network disturbances;

- support standardisation and integration of processes applied on in different Business Services enabling to use standardized Operators workstations to contribute to optimization of the works in Traffic Control Centres and to reduce failure rate in decision making caused by “non-automated” manual procedures.

WP8 Task 1 will be complemented by the definition of a Standardized Framework/Platform for Business Service Applications (Logic) WP8 Task 2 “Application Framework” (D8.5, D8.6, D8.7, D8.8) based on the requirements for a TMS and the definition standardized Operators workstation (WP7 Task 2 “Standardized Operators Workstation with Deliverable D 7.3 “Specification of the Standardized Operator Workstation”) shall foster the development of Applications which can be Plug-and-Play installed on these platforms and communicate via the Integration Layer.

A proof of concept is executed under WP7 Task 3.

1.2. Impact

In2Rail WP8 represents the start of the design process for a new trackside integrated ICT system. The activities contribute to the objectives of the White Paper Roadmap for a Single European Transport Area in terms of increasing overall capacity and reliability and reducing signalling infrastructure CAPEX and OPEX for train operations.

The following table highlights the impacts of WP8 on key Objectives.

Objective	Impact
Operation applying ATO, ETCS L3, Moving Block functionality	Communication from Business Service Applications located inside a Traffic Control Centre to CCS field infrastructure are key to ▪ enable ETCS L3, Moving Block and ATO operations at all; ▪ achieve expected CAPEX and OPEX improvements; ▪ achieve expected capacity improvement of at least 10%.
Traffic Management System evolution is expected to improve reliability of train operations in terms of punctuality and availability of wayside assets.	The integration of Asset Status information into the Regulation process via the Integration Layer will improve the Traffic regulation process providing less delays and better utilization of track resources both for Traffic and Maintenance activities. Status information will be generated from services managing the different assets. Information available within the CCS (cycles of Point machine/switch) can be further be made available to services via Integration Layer. The integrated processing of this information will lead to an improved planning of slots for preventive maintenance which can be then included into the Traffic regulation process and improve the availability of the different assets. An efficient

Objective	Impact
	maintenance will significantly contribute to increase the overall punctuality.
Standardization	As the current TMS ICT, processes, data structures and Interfaces, are not standardized, CAPEX always includes a high amount of cost for non-reusable adaptations. This is a burden for the client and for the supplier. The new integrated system will eliminate a very high percentage of these adaptation costs and will lead in general to a reduction of CAPEX through standardized: ▪ communication architecture; ▪ interfaces; ▪ data structures; ▪ Operator Workstations; ▪ Plug and Play Applications; ▪ feasible reduction of wayside infrastructure and to complexity of hand-over processes between CCS field Constituents e. g. RBC-RBC applying to a subscribe and Publish principle having all clients permanently listening to updated information instead of a serial Point2Point communication. of at least 10%. OPEX is expected to be reduced for at least 10% through: ▪ provision of updated traffic status information dynamically in a standardized Data format within TMS business applications and to other Rail services e.g. Maintenance and Energy Management Systems to increase the efficiency of their operations; ▪ standardized Operators workstation being able to display the different information available in a Traffic Control Center on one display.
Integration of Data	The integration of Asset Status information into the Regulation process will change the Traffic regulation process, generating for Maintenance and Energy Supply Management purposes leading to a more efficient planning of their activities providing a reduction of asset failures and hence improve the punctuality for another 5%.
Improvement of the overall line capacity	New Applications automated, and integrated processes will compensate the loss of stability of time tables for operations in fix-Block structures if line capacity is increased by shortening the sections.
Reduction of traction energy consumption, and carbon emissions	The availability of dynamically updated traffic information will optimize the energy consumption through onboard based algorithms optimizing a trip or applications embedded in the TMS providing models to improve the usage of energy on corridor or sector level. Savings of up to 10% are be expected.

Objective	Impact
Customer Experience	Definition of a ICT Structure will contribute to increased passenger satisfaction providing a more reliable, punctual and cost-efficient information update to passengers and external services through the Integration Layer and its Interfaces to internal and external Passenger Information systems. New applications supporting Passenger Comfort may be developed based on the continuous availability of Traffic Status data.

1.3. KPIs

The system defined under the Lighthouse Project In2RAIL WP8 has been continuously challenged against existing systems/processes applied in current rail traffic management to match its predicted performance with the objectives of IN2RAIL. The following "KPIs" have been used in the discussion process comparing actual processes with possible new and advanced procedures and applications supported from the Integration Layer and its constituents.

1.3.1. Punctuality

WP8 use the term "Punctuality" in reference to the variation in journey times. Such variation could come from recurring congestion or from non-recurring events, such as incidents. The goal is to minimize train delays due to infrastructure and operation system disturbance.

Targeted improvements are:

- minimizing train delays due to infrastructure and operation system disturbance;
- increasing the reliability of passengers/freight trains;
- reducing the average time required to clear the track after an incident;
- reduce time needed come back to full operation after clear track.

An advanced TMS with integrated ICT structure will support the increase of the total number of train services per track direction (Capacity) which will have a negative impact on punctuality as it reduces the robustness of the timetable. To compensate this weakness a modern TMS will be instrumental to implement advanced business functionalities as a tool/mitigation to minimize a recovery and may apply as integrated applications Automated Traffic Operation (ATO) functionality with embedded Driver Assistance Tool to stabilize the operational Time Table.

1.3.2. Reliability

Reliability shall represent the proportion of scheduled passenger and freight services that are cancelled compared to the number of services with a delay < 5 min for Passenger Trains and < 15 min for Freight Trains.

Reliability and Punctuality stats per member state for 2012 – 2014 are available in EC Database under:

https://ec.europa.eu/transport/sites/transport/files/modes/rail/studies/doc/2016-04-price-quality-rail-pax-services-final-report.pdf

1.3.3. Capacity

The TMS is the overall "Traffic Manager" and without propriate applications supporting the different conventional and new advanced traffic modes such as Moving Block/ETCSL3 or ATO, the targeted capacity improvements will not materialize. To characterize the benefits of the new design the following considerations were taken:

- Does the system support the increase number of trains possible on a specified line/corridor or network?
- Does the system support the optimization of Headway of succeeding trains?

Therefore, the number of possible additional trains per hour due to improvements in Traffic Management Business Applications based on availability of the different asset status data for traffic regulation has been agreed as indicator to challenge the new design.

1.3.4. Availability of Assets

Objective of the project is to provide via the Integration Layer a permanent update of the Traffic Status to Asset Maintenance Services to increase the efficiency of preventive Maintenance operations hence the availability of assets.

Therefore, as KPIs impacted from the TMS the following definition has been agreed to monitor if additional slots per day due to integrated Data exchange are possible.

1.3.5. CAPEX

For CAPEX evaluation no specific KPI has been defined as the system allows for different implementations if the CDM is respected.

The first indication will be available during the succeeding X2RAIL-2/4 projects and the related Open Call which is targeting the to set up a real Integration Layer ICT supported from an Infrastructure Manager.

However, all design decisions have been discussed regarding cost for Development, Approval, Tools, investments costs with various stakeholders not necessarily using same interfaces, standards etc (for TMS) to secure the targets of the overall S2R development program.

1.3.6. OPEX

For OPEX 2 main parameters of life cycle cost impacted from the TMS were considered alongside the progress of the project:

- trains per day on a line/corridor or sector as representation of the utilization of the infrastructure assets;
- total amount of Delay from Disturbance in minutes.

2. Background

Most currently deployed Traffic Management Installations have “functional” sections in the control rooms receiving different kind of operational information from different systems, like telecommunication systems, interlocking systems, notification systems and dispatching systems and requiring enormous efforts to achieve sufficient transfer of data between functional areas. To make a correct decision, the traffic managers and dispatcher need to access different command and control systems in variant location in the command and control centre. This reduces the efficiency and effectiveness of workflow of the traffic managers and therefore business process of the railway system.

Partial Automation is available only on sub-system level. Very often Signallers are required to check the status of the infrastructure by observing on-screen symbols on regular intervals. Status of maintenance related services, energy resources and other traffic-related assets have to be manually integrated into the TMS process.

2.1. Integrated Mobility Management

Integrated Mobility Management (I^{2}M) must be smart and based on a real-time seamless access (without duplication) to heterogeneous railway data sources (signalling data, maintenance plans, environmental conditions, fleet status, passenger’s requests and needs; etc.) with a view to enabling all rail stakeholders to measure their performance and optimise their operations and planning, and ultimately the service they offer to the end users (meaning travellers and freight customers). I^{2}M provides a seamless exchange of information between fixed and mobile services in different transport modes, and achieve a standardisation of Interfaces, processes and data structures on such a level to ensure compatibility independent from supplier´s subsystems and modules.

For freight and passenger operations it is essential to receive high precision and real-time forecast of traffic status and demand to manage resources such as availability of platforms and terminals, locomotives or coaches, operators, drivers and forecast and others in an efficient way. Maintenance services depend strongly on the dynamic planning of free slots to execute the maintenance activities and updated now-casted and forecasted periods of availability of the assets for maintenance are key.

WP8 constitutes one of the issues in the framework of the Project titled “Innovative Intelligent Rail” (Project Acronym: In^{2}Rail; Grant Agreement No 635900) and will provide the first step change towards seamless fully-automated process integration of railway related services and other modes of transport.

The scope of WP8 is to define a standardised integrated ICT environment capable of supporting TMS dispatching services and to describe a plug-and-play framework for business service applications. (Figure 2.1)

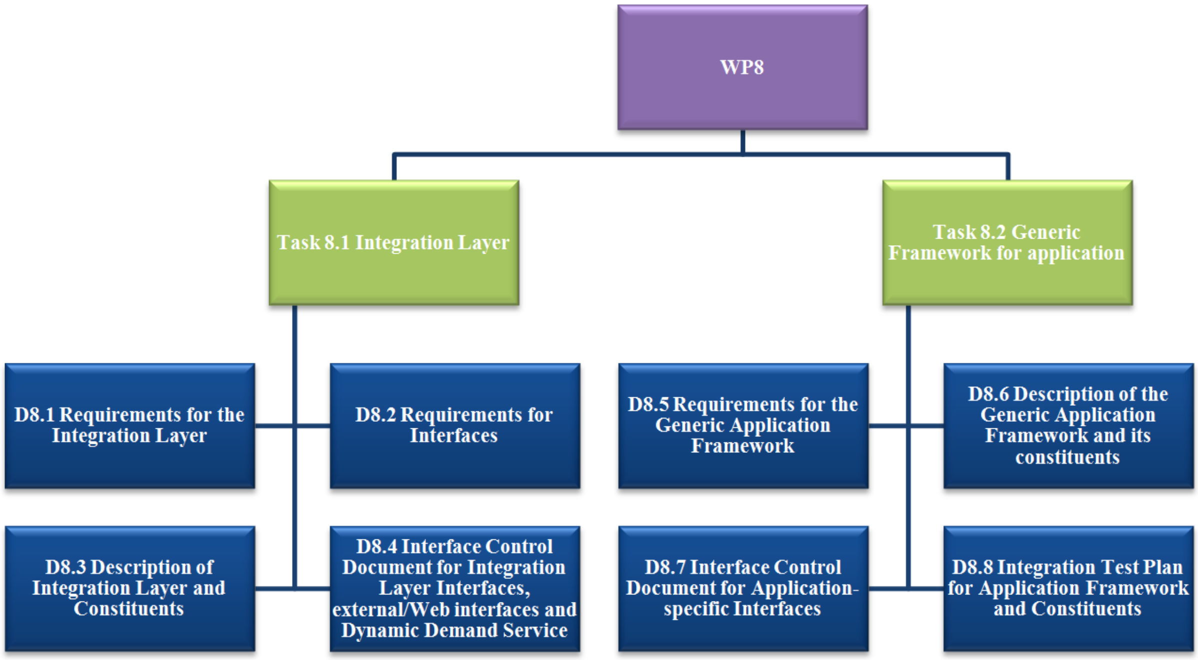

WP8
Task 8.1 Integration Layer
Task 8.2 Generic Framework for application
D8.1 Requirements for the Integration Layer
D8.2 Requirements for Interfaces
D8.5 Requirements for the Generic Application Framework
D8.6 Description of the Generic Application Framework and its constituents
D8.3 Description of Integration Layer and Constituents
D8.4 Interface Control Document for Integration Layer Interfaces, external/Web interfaces and Dynamic Demand Service
D8.7 Interface Control Document for Application-specific Interfaces
D8.8 Integration Test Plan for Application Framework and Constituents

Figure 2.1: Scope of WP8

The Integration Layer is a key enabler for an SOA as it provides the capability to mediate which includes transformation, routing, and protocol conversion to transport service requests from the service requester to the correct service provider.

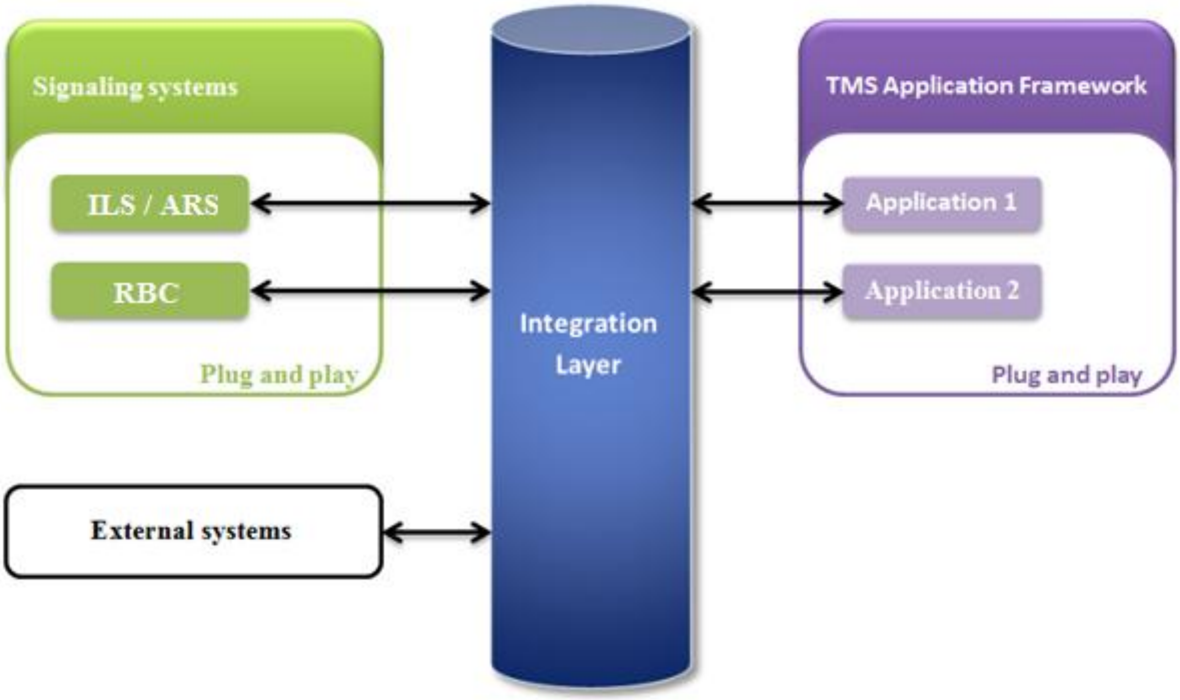


Figure 2.2: Integration Layer and Application Framework

2.2. Integration Layer

The key requirement of a future state of the art integrated Rail Traffic Operation System is the ability to provide sharing of any information relevant to the Integrated Mobility concept.

The Integration Layer (IL) will provide these Data at the right time and deliver them to the right destination.

This integrated process will support intelligent, automated and flexible rail traffic operations, but also an integrated approach to the optimisation of railway architecture and operational systems at network, route and individual train level. These will reconcile business and operational requirements (namely customer service, capacity, speed, timekeeping, energy, asset management) with real-time field and asset condition monitoring and intelligent traffic planning.

The scope of works of WP8 Task 1 “Integration Layer” comprise the non-functional requirements of this “Data Highway”, the description of the architecture, Interfaces, Data Model and Data Structure of status data from different Business Services being integrated into the decision-making process of Traffic Management Systems. This integration of data of different business services into one Data Highway is one of the major innovations of this program.

Another specific focus within the design of the future IL will be to specify the Interfaces e.g. WEB-IF to external business services outside the rail environment such as weather forecast systems or road based services and all links to trackside Traffic Control Systems such as Automated Traffic Operation (ATO) and the communication system for ETCS (Radio Block Centre).

2.3. Application Framework

The second targeted major innovation of WP8 Task 2 “Generic Framework for Applications” is the specification of a standardized framework for business service applications which are connected through the Integration Layer and allowing a plug and play installation and operation of Business Logic SW Modules. The works include the specification of the non-functional requirements for such a framework the description of its architecture including functional description of applied substructures and Interfaces and a test concept.

The standardisation includes specification of the interfaces to external systems and plug-and-play mechanism for the applications inside the Application Framework.

3. Methodology of works

3.1 Execution of Works

All partners of WP8 have agreed to perform jointly all works allocated to the different tasks.

For each deliverable, a coordinator was nominated who was responsible for developing the document structure which was then reviewed by all the partners and when finally agreed was integrated into the contributions of the other partners to form the final document.

Deliverable D8.3 Description of Integration Layer and Constituents is coordinated by Bombardier Transportation.

Telephone conferences and periodic progress meetings were used to align the inputs and to clarify difficult technical topics.

All partners listed in Chapter 1 of this document provided assigned input and conducted reviews to all chapters of the document.

Deliverable D8.3 is complemented by D8.1 “Requirements of the Integration Layer”, D8.2 “Interfaces” and D8.4 “Interface Control Document for Integration Layer Interfaces, external/WEB interfaces and Dynamic Demand Management”

This set of Deliverables are key inputs to the S2R IP2 TD9 “Evolution of Traffic Management Systems” and S2R CCA WA4.2 “Integrated Mobility” programs.

The Proof of Concept of the specifications developed under WP8 is part of IN2RAIL WP7 Task 3 “Proof of Concept” which define [D7.4] an application scenario to show the concept of Integrated Mobility Management. The relevant parameters are identified and method for a proof-of-concept is also defined. Additionally the deliverable D7.5 [D7.5] reports and explains results of the proof-of-concept of the Integrated Mobility Management.

3.2 Document overview

The aim of this document is to present the concept of the Integration Layer and its constituents, and a discussion of possible design concepts and technologies.

The description of the Integration Layer is structured as follows:

- Chapter 4 presents applications that will utilize and communicate via the IL;
- Chapter 5 explains different approaches regarding the design of the IL:
  - message centric approach,
  - data Centric approach,
  - present conclusion of the working group around this general discussion;
- Chapter 6 describes the Canonical Data Model
  - the data modelling approach with central modelling concept,

  - present structure of proposed data model to be use for integration purposes,
  - discussion of the metadata and data formats to be used with the model;
- Chapter 7 presents different aspects of IL related to architecture and non-functional requirements;
- Chapter 8 presents technology / product candidates for the IL;
- Chapter 9 contains the conclusion of the different evaluations summing the final design proposal;
- Chapter 10 list all References.

4. Business Applications supported from IL in the Scope of In2Rail

This chapter shows the needs of the different subsystems linked with the IL. The description includes basic functionality from both perspectives – the consumer and provider side.

4.1. TMS functions

The TMS (Traffic Management System) comprises many functionalities which are needed to manage and optimize railway operations.

The concept of TMS is to produce a single source of data for train time-tabling, rolling stock allocation, crew deployment and to use this data for operations planning and comparison to real time operational events, thus automating train trips in the optimum way.

To indicate the provisions to be taken in the design of a Integration Layer, the typical functionalities of a TMS are presented in the following chapters.

4.1.1. Indications and Commands

The primary functionality is the ability to remotely monitor and control field devices. Functionality in this layer includes the following:

- Indication /Command related to track occupancy;
- Indication /Command related control of switch point position;
- Indication /Command related to routes;
- Indication /Command related to signal states

4.1.2. Train Tracking.

Track occupancy detectors (Track Circuits or Axle Counters) deliver the operator the information which sections of the track network are occupied by trains.

The Train Tracking functionality (either as part of TMS or the Interlocking) then transforms these "Track Occupied Messages" into a "Train Movement" representation on an HMI or wall screen.

In the future, the Train Position reports including the length of a train broadcasted from the RBC to the TMS will be another source to generate a visual presentation of the train moving along the track.

The System, by default, automatically assign a unique train identifier to each new occupancy. The operator can edit the identifier to one that is consistent with the customer's operational train identification scheme.

4.1.3. Timetable Management

Timetables are at the heart of railway operations, defining the train services to be run on a given day/period. TMS includes a capability to define and activate timetables to define all planned train movements on the system.

System uses the active timetable to automatically assign the correct train identifiers to trains. If trains arrive on the system in a different sequence than in the timetable, the operator can modify the assigned identifier.

4.1.4. Train Routing

Routing Applications use the timetable to determine how each train should be routed through the network. As the trains move along the track, system sets the appropriate routes in front of the train to keep it moving through the system without operator intervention. Before setting a route, system can execute a configurable pre-test to avoid sending a command to the interlocking that will be rejected. It is possible to configure system also with an alternate route for cases where the preferred route is not available.

4.1.5. Traffic Forecasting and Conflict Detection

System continuously gathers statistics on point-to-point train running time and uses these statistics, along with timetables and the current state of the rail network, to forecast train movements in the future. With the forecast available for all trains, the system highlights on the operator display conflicts between trains (e.g. two trains on the same track at the same time) in order that the operator can act to solve/prevent a conflict as early as possible.

4.1.6. Automatic Traffic Regulation and Conflict Resolution

At its highest automation level, TMS optimizes traffic movement according to a specific objective. This optimization is achieved by shortening or lengthening the duration of stops at stations and terminus locations by increasing or decreasing train speeds between locations and automated Traffic regulation without human intervention. These principles are applied in several installations for Mass Transit.

This level of automation is not yet achieved in Main Line Operations and is a key target of the Shift2Rail program.

4.2. Asset Management

Asset Management Systems (AMS) encompass operation, renewal, upgrade and maintenance of the rail assets, applying different applications representing tools for planning, decision support management or other elements of this service.

TMS and Asset management system communicate via the IL to exchange e.g. Traffic Status (Source: TMS), Asset status (Source: AMS) and other data needed as input for the different business applications of the services.

Core decisions and activities that take place, from setting initial objectives for the network through to the execution of work and the operation of the railway infrastructure are shown in the following figure

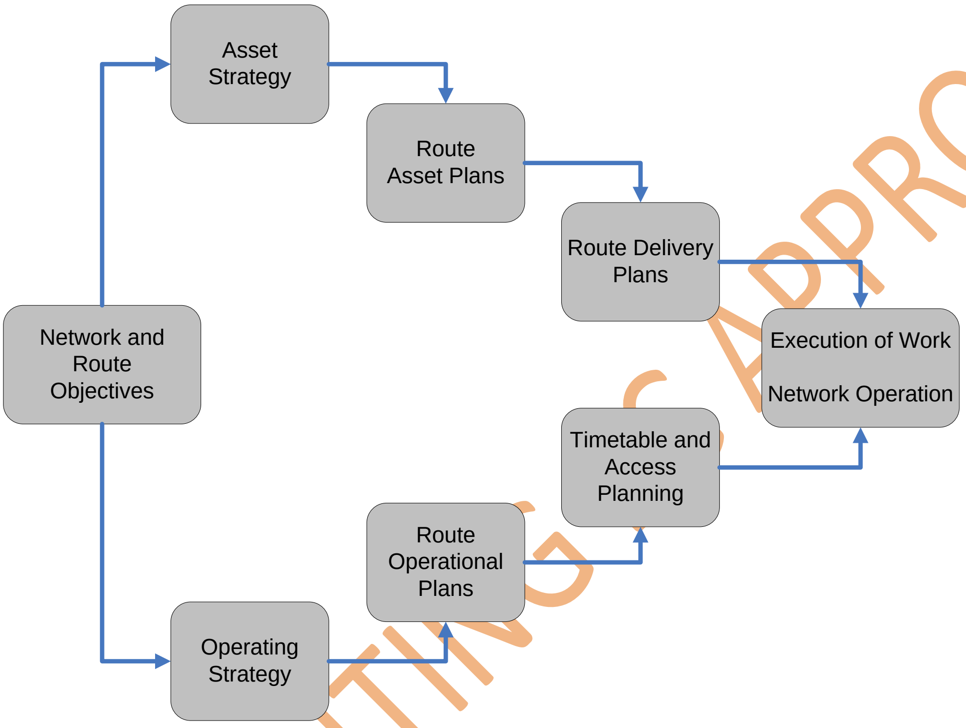


Figure 4.1: Core decision and activities from strategies to implementation – overall schema

Decision are combined for the operation of the infrastructure and for the asset intervention. It includes the long-term and short-term elements of these decisions. The asset branch starts with the strategy to deliver the objectives and progresses through the planning phases to the execution of work on the ground. The operational branch starts with the long-term specification of train service capacity and the associated access necessary for maintenance and renewal. It progresses through the planning of train paths to meet specific passenger and freight train operator requirements to the preparation of the timetable and access plan to the day to day operation of the network.

A description of all the components in the previous figure is provided in the UIC "Railway Application Guide: Practical Implementation of Asset Management through ISO 55001".

4.3. Maintenance operations

The quality of the railway infrastructure plays a fundamental role in railway transportation and is represented and monitored by the IMs using KPIs for availability, reliability and safety. Railway systems need substantial maintenance activities to guarantee the best possible performances at any time.

Reactive Maintenance

Reactive maintenance means that assets are maintained when they are broken. Basically, there are two kinds of reactive maintenance. This is traditionally considered as the worst case as failures will occur unexpected and provide typically the highest disruption of the operational time table. Failure of assets triggers an Alarm within the controlling asset which is then broadcasted to the TMS

Condition-based Maintenance (CBM) - In condition-based maintenance, failures or break-down will be avoided by maintaining the assets when they show signs of decreasing performance or upcoming failure. The assets must be monitored continuously to see condition changes in time. This information gathered from “system internal” (e. g. Information of the asset which is available within the system e. g. number of movements of a point machine) and from external sensors (e. g. device to monitor the level of current applied from the point machine motor) will be available for all clients subscribe on the Integration layer. Fore-casting algorithms and methodologies are scope of In2Rail WP9.

Example:

As soon as monitoring shows that the condition is below the given trigger values (dashed line in figure 4.2), maintenance is requested and executed, resulting in a condition improvement. It is necessary to define trigger values for the measures that initialise need for maintenance. Thereby, the time between two condition measurements, as well as the time between maintenance request and execution, must be considered to ensure punctual maintenance. The best maintenance activity should be defined in advance. Decision support in resource allocation can help to reduce the time between maintenance request and execution. Then, trigger values can be higher, and the maintenance effort can be reduced.

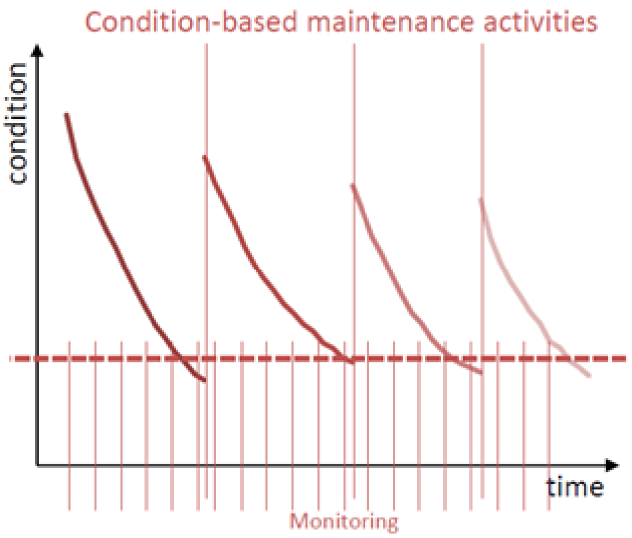


Figure 4.2: Condition-based Maintenance

4.4. Energy Management (Power Grid)

The IL supports systems managing the power grid by broadcasting the updated forecast of Traffic. In return the status of the Electric Traction (Power) System (ETS) and its constituents is a significant input to the TMS to organize the traffic.

The figure below shows a general description of an Electric Traction System. The source of the graphic is the standardization working group preparing the EN50562 within the CENELEC (see [EN50562]). Please note, that the figure itself is taken from the preliminary standard FprEN 50562.

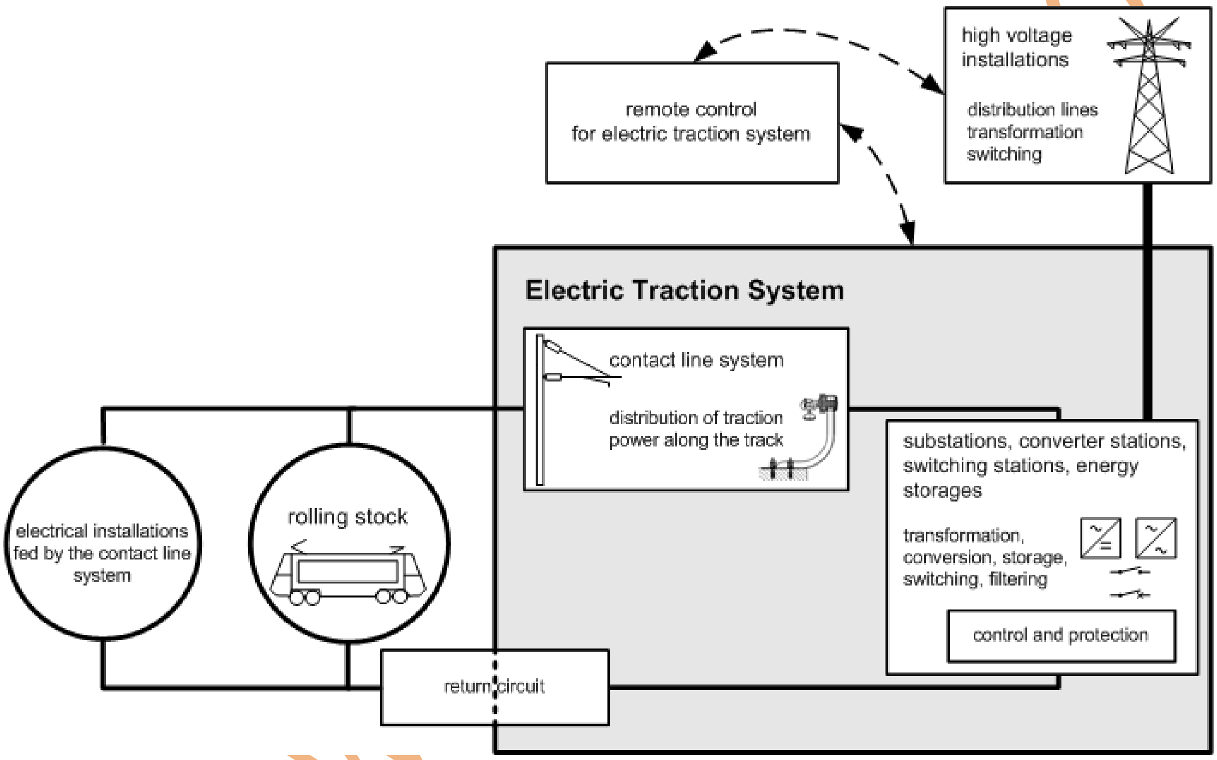

high voltage installations
distribution lines
transformation
switching
remote control for electric traction system
Electric Traction System
contact line system
distribution of traction power along the track
substations, converter stations, switching stations, energy storages
transformation, conversion, storage, switching, filtering
control and protection
electrical installations fed by the contact line system
rolling stock
return circuit

Figure 4.3: Electric Traction System general description according to FprEN50562

Most electric traction system systems are supplying electrical power to the trains / vehicles via a contact system. This contact system is an overhead contact line or a third rail. The running rails are part of the electric traction system as they provide the return path for the current.

The contact line system is divided in sections. These sections can be energized (switched on) or not (switched off) while usually being controlled by the next substation or switching station. In a lot of cases a feeding section is energized from two sides.

A substation is feeding the contact line system whereas a switching station connects or disconnects parts of the overhead contact system. A substation is fed by an upstream network or in some cases by a generator.

Due to the increasing dynamics and information scope in communication processes between technical components including those of an ETS, more accurate and up-to-second information is required for supporting higher levels of automation. Therefore, the current status and limitations or even non-availability of electrical power for traction units will play an important role for future TMS in order to provide accurate target times and journey locations as resulting from the Traffic Management function.

Multi Train Simulation (MTS) - Particularly the integration of a Multi Train Simulator module for simulating the power consumption of trains running within the area of one ETS substation is deemed to be beneficial as being described in WP10, D10.4 of the In2Rail Project. The MTS provides expected power limitations, e.g., when multiple, heavy freight trains are to start-up and accelerate at the same time within one substation area. Since only reduced traction force can be applied, the dynamic forecast calculation of TMS immediately adapts the forecasted target times which may also include start-up sequence optimisation for the trains according to priority rules. Due to the resulting relaxation of power consumption, trains may accelerate now more than expected leading to another forecast adaptation representing a “control loop” between TMS and MTS.

Power outage information - Consequences resulting from major power outages is enabled and resulting consequences on scheduling of trains can be derived by TMS for dynamic adjustment of the overall production plan.

Status information of ETS components - For classical supervisory capabilities as required in Traffic Management Centres, status information need to be acquired from ETS components such as, e.g., electrical switches and here, especially in relation to possession or emergency activities.

4.5. External services

Integration Layer can handle External Services. The scope of WP8 comprises communication with external services via web interface (web protocols). In the first step information broadcasted from weather stations, demands for specific traffic from external services and Passenger Information via external systems /services are comprised in In2Rail scope. The propriate data trees are defined in In2Rail Deliverable D8.4.

4.5.1. Weather forecast/reports

Weather forecast is used in the traffic management process to decide whether certain restrictions of running, e.g. speed restriction or line/bridge closures shall be implemented or not. This is today mostly involving situations of critical wind speeds or snowfall as reported or expected for certain regions for certain times. Todays’ and future weather services are providing more details with shorter lead times which can be used for provision of more exact forecast calculation of the trains currently running on the network. For instance, solar radiation has a strong impact on catenary resistance as well as frozen catenary. Both are impacting power availability and finally traction force as being applied by electrical motor units. On the other hand, higher humidity and leaf-fall conditions have an impact on wheel/rail adhesion which itself is restricting the acceleration and braking capability

To derive restrictions on driving parameters or infrastructure availability from the information delivered by the weather services, appropriate business rules and logic are required which resides in the TMS and not within the interface itself.

4.5.2. Dynamic Demand

Dynamic Demand communication services allow for a highly dynamic, technical negotiation of existent resource restrictions and effective transport requirements at the time of operation. The idea behind is to improve today's process of verbal discussion between IMs and RUs involving:

- resource issues at IM (available track capacity, station staff, …) or RU (available engine driver/crew of required qualifications, appropriate rolling stock, available unloading facilities…);
- changes of transport demand (RU) because of, e.g., on-demand stop capability for passenger trains or associated freight production processes (ship waiting for containers on train is late in schedule, some waggons carrying semi-finished products or raw materials for automotive industry are not required anymore at plant A but now at plant B, …) or other operational or business reasons;
- it is expected, that the provision of a technical negotiation capability will speed-up the decision making and improve the quality of re-scheduling solutions since very short-term changes in linked production processes, the related resource availability as well as various, today sometimes "hidden" solution options are now considered.

4.5.3. Passenger Information

Traditional passenger information systems (PIS) require last minute information about train delay or cancellation of trains or stops or transfer connections. The information is presented most often at platform boards or central station displays or even within the trains using specialised displays. Today's passenger information services also include support for Journey/Trip Planner Apps as well as Web based information presented on maps or other views.

Passenger information may be used for the traffic management applications of the future. Evolving train technology like, e.g., automated passenger counting or estimation of number of passengers within a coach derived from weight comparison is already technically available. Furthermore, the use of smartphones along with Travel-Apps can be used to identify individual passenger locations within trains and associate their available travel destination (based on ticket) or effective travel priorities within delayed trains for influencing dispatching decisions. Please note that the handling of such information is very much related to the Dynamic Demand service as described in the previous subsection.

4.6. Freight Forwarding Services

For freight operating companies, it is essential to know the current position of payload as located on certain waggons of freight trains. By knowing the respective trains, available train position information systems like, e.g., the European Train Information System (TIS) as being available from RailNetEurope (see http://ittools.rne.eu/TIS/what-is-tis/) can be used today. Most often, the currently estimated time of handover (ETH) or arrival (ETA) is shown at the same time within such systems.

Since waggons are sometimes added to other trains it is also important to locate the payload (e.g. container) or respective waggon itself rather than the train. Freight operators are more and more using GPS technology attached to waggons or containers in order to provide that kind of information to freight transport managers and make it visible on appropriate maps or views within specialised applications.

4.7. Train or Possession Requests

Train or Possession Request services are mostly available today through existing IM portals featuring specialised Web applications. Train paths are requested by RUs and offered by IMs. This process follows a pre-configured workflow since there can be several situations as a result of rejecting, re-bidding, or disputing. The agreed train paths are integrated into the time-table, crew and vehicle resource plans.

4.8. TAF/TAP TSI

As part of the technical specifications of interoperability (TSI), the European Railway Agency (ERA) has published the Telematics Application for Freight (TAF, http://www.era.europa.eu/Document-Register/Pages/TAF-TSI.aspx) and for Passenger (TAP, http://www.era.europa.eu/Document-Register/Pages/TAP-TSI.aspx) which supports communication between freight/passenger RUs and IMs.

The available message types are covering the communication processes for (ad-hoc) freight path requests and offers, information exchange for freight trains (TAF) and tariff, reservation and ticket data (TAP).

Usually, the communication needs to be established via the Common Interface (CI) of RailNetEurope (http://ittools.rne.eu/CCS/common-interface/) which acts as a protocol converter. A external interface connected to the TMS Integration Layer will have to communicate using one of the protocols supported by the locally installed CI (i.e. SOAP(HTTPS), JMS, FTF, File SMTP, JMS-MQ) which itself communicates via SOAP(HTTPS) XML to the external communication partners.

4.9. Neighbour TMS

To provide smooth integration with cross-border traffic, it is required to allow integration with the IL of a neighbouring TMS. Please note that bespoke IL-interfaces will have to be set-up in the case of external IMs using (legacy) TMS without directly compatible IL as provided by In2Rail(WP8)/Shift2Rail(X2Rail-2) activities.

The communication between neighbour TMS includes exchange of train and possession related information.

4.10. Web Services

In similar situations of today's IT-world, the well-known and widely accepted Web-technology is used which comes down on using Web services. When exposing TMS (interface) services to the outside of the local IT environment, a Web server is typically performing the required protocol conversions and – aggregations. The Web server is completely stateless, hence easily scalable and presents a set of high level APIs fitting to the needs of the communication partners. Furthermore, it does not contain any business logic.

The communication between the Web server and TMS services is based on remote procedure call approach such as JSON RPC (see http://www.jsonrpc.org/). Please note that although SOAP style architectures are in widespread use, they are more suited for internal use or for services called by trusted partners.

The Web server communication with the outside IT environment uses HTTP, with XML payload as required, and is usually built based on the Representational State Transfer (REST) paradigm. The data structures used in requests and responses are provided in XML Schema being defined by XSD.

The Web server implementation is recommended to be based on the open source such as, e.g., **Jetty Web** Server which itself has been proven and tested in large scale implementation projects such as Yahoo! Hadoop Cluster or the Google App Engine.

4.11. Command and Control Systems

4.11.1. Interlocking (Controlling Routes, sections and Objects)

The main function carried out by an interlocking system is to set and lock routes as requested to deliver the operational time table. The proposed messaging between Interlocking and TMS is described in Delivery D8.4.

4.11.2. RBC (Controlling Trains)

The RBC is the safe central trackside equipment of the ERTMS/ETCS level 2 and is responsible for the security of all trains running in the level 2 area with which a GSM-R communication has been established. The proposed messaging between Interlocking and TMS is described in Delivery D8.4.

4.11.3. ATO

ATO (separated trackside system or function embedded in TMS) is to perform the automatic regulation of the trains. It calculates continuously Journey Profiles and send them to onboard ATOs.

The main function of on-board ATO is to perform an efficient speed and traction control to accomplish the up-to-date Journey Profile sent by trackside ATO. To do so, it calculates the optimal speed profile of station to station runs and traduces it to traction, braking and speed commands.

ATO is not in scope of WP8 of In2Rail project and will be added to the Data Model in X2R-2 WP6.

4.12. Crew management

Crew management System represents a part of the IT-Infrastructure at a Railway Undertaking. It manages among others train drivers and train conductors by implementing several tasks:

- creating and adjustments of crew rostering, which defines for each participating crew member, on which trains it shall work during current and several next shifts;
- observation of crew members availability (e.g. outage due to illness);
- ensuring compliance to the tariff requirements (e.g. shift duration, breaks, rest intervals etc.);
- communication with crew members for reallocation (request/reply pattern).

Besides the complexity of each task itself the management of big amount of people represents an additional challenge for automation. Especially bigger RUs manage up to 100.000 train drivers [IND2017].

To fulfil its tasks a Crew management system has typically a centralized architecture having databases and backend functionality either in private control centre [IND2017] or in a cloud.

In case of disturbances the TMS often requires following data from the CMS:

- crew rostering: assignments of crews to trips including start and end station of each assignment. This must be considered if rerouting options are analysed due to disturbances;
- static crew data:
  - names of crew members,
  - communication means,

  - list of allowed railway lines, to identify possibilities for train rerouting;
- contractual requirements:
  - end of the shift,
  - list of breaks,
  - other required data (examples: skills, certificates).

This data allows identification if for specific disturbance the Railway Undertaking must be taken into discussion and rearrange crew rostering/agree on changes in shift times or provide another crews for rerouted trains.

All necessary data are exchanged via the IL.

4.13.Fleet management

The Traffic Management System of the future shall be linked with Systems for Fleet / Rolling Stock Management. These systems typically are used from Train Operating Companies (TOC) to manage the rostering of trains and vehicles.

A Fleet Management Application shall provide in time the information needed to assign a train consists (locomotives, power heads, wagons, sequences …) to a specific demand. On the other hand, the TOC keeps the rostering plan synchronized based on updated fore-cast of the TMS.

All necessary communication can be handled via the IL.

Interaction between TMS and Fleet Management Systems are:

- detection of technical conflicts of the assignments, e.g. electric traction assigned to a train that was planned to run on a path that does not provide catenary power;
- detection of operational conflicts in the assignments, e.g. a locomotive cannot be used yet for a train run because of a delay of the train it is currently assigned to;
- Fleet Management Application provides updated assignments to scheduled train services;
- placement of the reserve assets.

5. General description of IL

The key aspect that distributed system applications can exist and fulfil their role is to exchange information. The communication paradigm, how to get the right information from the right producers to the right consumers at the right time, is perhaps the first and most important design consideration. Because it impacts every application, this choice drives many key properties, including timeliness, scalability, reliability, availability, configuration, and evolution.

Distributed systems must also share and manage state. In a complex system, every application both generates and uses distributed and local state. Like the communications paradigm, the strategy to understand and manage this state is a fundamental design decision.

There are many, many other considerations, including joining and leaving the network, discovery, error handling, and legacy integration. However, matching the communications paradigm and the state management approach to the system are foremost. This is where the difference between message and data centricity has the greatest impact [Electronic Design] – the two approaches that should be taken into consideration to design the Integration Layer of a distributed system.

5.1. Message centric approach (Messaging)

This technology enables high-speed, asynchronous, program-to-program communication with reliable delivery. Programs communicate by sending packets of data called messages to each other. Channels, also known as queues, are logical pathways that connect the programs and convey messages. A channel behaves like a collection or array of messages, but one that is magically shared across multiple computers and can be used concurrently by multiple applications. A sender or producer is a program that sends a message by writing the message to a channel. A receiver or consumer is a program that receives a message by reading (and deleting) it from a channel.

The message itself is simply some sort of data structure—such as a string, a byte array, a record, or an object. It can be interpreted simply as data, as the description of a command to be invoked on the receiver, or as the description of an event that occurred in the sender. A message contains two parts, a header and a body. The header contains meta-information about the message—who sent it, where it's going, etc.; this information is used by the messaging system and is mostly (but not always) ignored by the applications using the messages. The body contains the data being transmitted and is ignored by the messaging system. [EIP Hohpe, Wolf]

Messaging capabilities are typically provided by a separate software system called a messaging system or message-oriented middleware (MOM). It coordinates and manages the sending and receiving of messages.

The reason a messaging system is needed to move messages from one computer to another is that computers and the networks that connect them are inherently unreliable. Just because one application is ready to send a communication does not mean that the other application is ready to receive it. Even if both applications are ready, the network may not be working, or may fail to transmit the data properly. A messaging system overcomes these limitations by repeatedly trying to transmit the message until it succeeds. Under ideal circumstances, the message is transmitted successfully on the first try, but circumstances are often not ideal.

In summary a message centric approach requires that:

- middleware exchange messages;
- infrastructure is unaware of the content;
- applications send messages between participants using MOM.

Message transmitted can be structured in five steps:

1. Create — The sender creates the message and populates it with data:
2. Send — The sender adds the message to a channel;
3. Deliver — The messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver;
4. Receive — The receiver reads the message from the channel;
5. Process — The receiver extracts the data from the message.

This diagram illustrates these five transmission steps, which computer performs each, and which steps involve the messaging system:

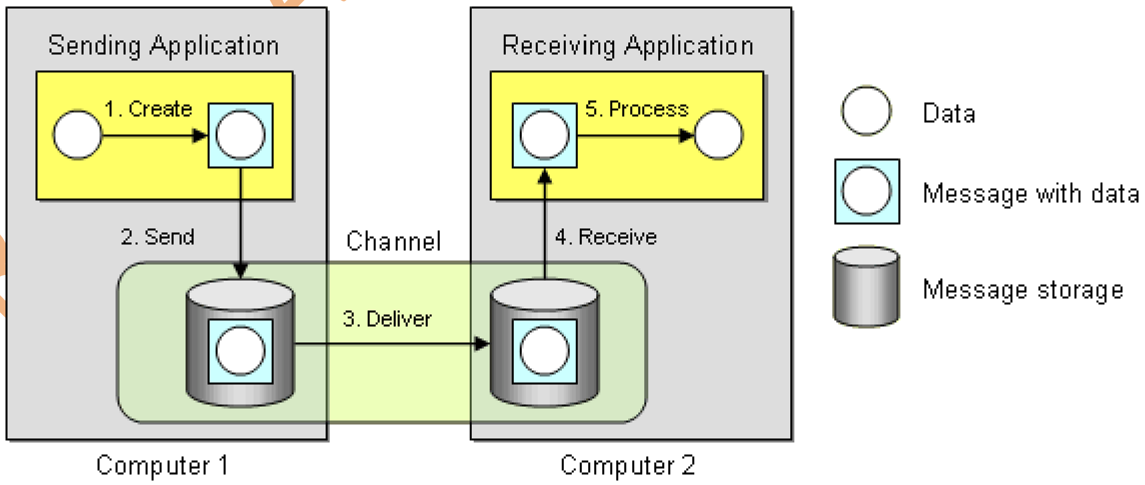


Figure 5.1: Message transmission step-by step [EIP]

This diagram also illustrates two important messaging concepts:

- Send and forget — In step 2, the sending application sends the message to the message channel. Once that send is complete, the sender can go on to other work while the messaging system transmits the message in the background. The sender can be confident that the receiver will eventually receive the message and does not have to wait until that happens;
- Store and forward — In step 2, when the sending application sends the message to the message channel, the messaging system stores the message on the sender's computer, either in memory or on disk. In step 3, the messaging system delivers the message by forwarding it from the sender's computer to the receiver's computer, and then stores the message once again on the receiver's computer. This store-and-forward process may be repeated many times, as the message is moved from one computer to another, until it reaches the receiver's computer. [EIP]

Next to described Point-to Point scenario also Publish-Subscribe channels are available. In such scenario one input channel splits into multiple output channels, one for each subscriber. When an event is published into the channel, the Publish-Subscribe channel delivers a copy of the message to each of the output channels. Each output channel has only one subscriber, which is only allowed to consume a message once. In this way, each subscriber only gets the message once and consumed copies disappear from their channels.

Another aspect is that messaging generally provides one-way communication, so the question is how to get the response for the message already send. This can be achieved with two channels:

- requestor sends a request message and waits for the reply message;
- replier receives the request message and response with a reply message.

The request channel can be a Point-to-Point channel or a Publish-Subscribe channel. The difference is whether the request should be broadcasted to all interested parties or should be processed by only single consumer.

When a caller performs a remote procedure invocation the caller's thread must block while it waits for the response. With Request-Reply. The requestor has two approaches for receiving the reply:

- synchronous block – a single thread in the caller sends the request message, blocks to wait for the reply message, and then process the reply;
- asynchronous call-back – one thread in the caller sends the request message and sets up a call-back for the reply. A separate thread listens for reply messages. When the reply message arrives, the reply thread invokes the appropriate call-back, which re-establishes the caller's context and process the reply.

Typical features characterizing messaging systems are:

- Message Routing;
- Message Format\Value translation;
- Message Bridging.

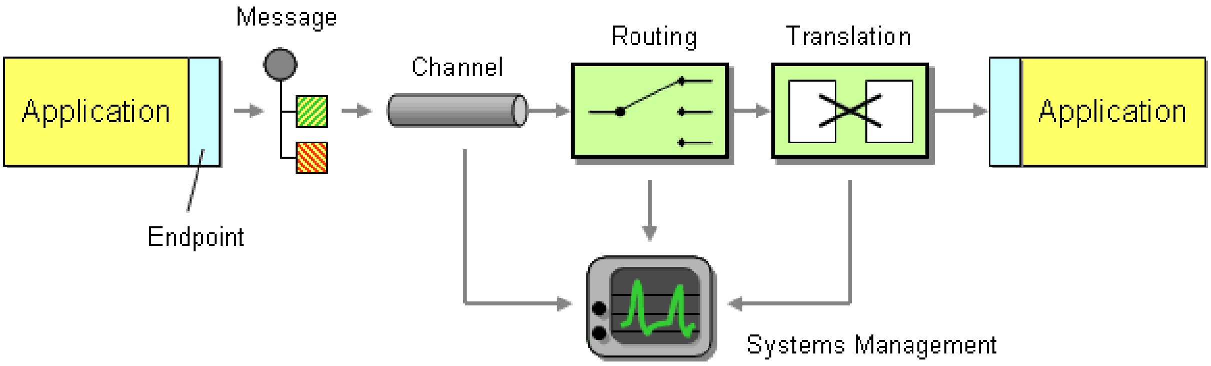


Figure 5.2: Basic Elements of an Integration Solution [EIP]

The above diagram illustrates the integration of two applications via messaging platform where a one-way communication is presented. Applications connect to the messaging channel with the endpoints. In case the message has to reach also one or even more systems the routing component was added to the design. The translator component allows to change format of the message when the second application does not understand the initial format in which the message was sent.

All these or other components of messaging middleware is good to monitor and manage. The System Management monitors the flow of data, makes sure that all applications and components are available and reports error conditions to a central location.

Important messaging benefits:

- remote Communication - enables separate applications to communicate and transfer data;
- Platform/Language Integration - a messaging system can be a universal translator between the applications that works with each one’s language and platform on its own terms;
- Asynchronous Communication - The sender does not have to wait for the receiver to receive and process the message; it does not even have to wait for the messaging system to deliver the message. The sender only needs to wait for the message to be sent, e.g. for the message to successfully be stored in the channel by the messaging system. Once the message is stored, the sender is then free to perform other work while the message is transmitted in the background. The receiver may want to send an acknowledgement or result back to the sender, which requires another message, whose delivery will need to be detected by a call-back mechanism on the sender;

- Reliable Communication - When the sender sends a message, the messaging system stores the message. It then delivers the message by forwarding it to the receiver's computer, where it is stored again. Storing the message on the sender's computer and the receiver's computer is assumed to be reliable;
- Mediation - The messaging system acts as a mediator between all of the programs that can send and receive messages. The messaging system can be used to provide a high number of distributed connections to a shared resource, such as a database. The messaging system can employ redundant resources to provide high-availability, balance load, reroute around failed network connections, and tune performance and quality of service. [EIP Hohpe, Wolf]

Challenges of asynchronous messaging:

- Complex programming model - asynchronous messaging requires developers to work with an event-driven programming model;
- Sequence issues - Message channels guarantee message delivery, but they do not guarantee when the message will be delivered. This can cause messages that are sent in sequence to get out of sequence. In situations where messages depend on each other special care must be taken to re-establish the message sequence;
- Synchronous scenarios - Not all applications can operate in a send and forget mode. Therefore, many messaging systems need to bridge the gap between synchronous and asynchronous solutions;
- Performance - Messaging systems do add some overhead to communication. It takes effort to make data into a message and send it, and to receive a message and process it. It is not wise to exchange huge amount of data via it as for example to replicate some initial information between systems. Messaging is best suited to keeping the systems in sync after the initial data replication;
- Limited platform support - Many proprietary messaging systems are not available on all platforms;
- Vendor lock-in - Many messaging system implementations rely on proprietary protocols. As a result, different messaging systems usually do not connect to one another. This can lead to a whole new integration challenge: integrating multiple integration solutions! [EIP Hohpe, Wolf].

Examples of commercial messaging platforms:

- IBM MQ;
- Java Message Service (JMS);
- Microsoft MSMQ.

5.2. Data centric approach

Data centricity, by contrast, makes the data the means of interaction. A data centric infrastructure must define the data it manages. Then, the infrastructure imposes rules that determine how data is structured, when data is changed, and how it is accessed. There is a notion of a "global data space", where data values, in known structures, are exchanged. The infrastructure is aware of the contents. With a data-centric approach, developers write applications that read and update entries in this data space [Electronic Design].

This difference to MOM may not be obvious until considering what happens, for instance, when an application joins after missing 50 updates to the value for example of some temperature sensor. The message centric system will send all 50 updates, and the receiver must then process each one. The data-centric middleware distributes the data space, and then simply allows the late joiner to read the latest value of each object, only one in this case. Both end up with the right value, but the network traffic and application processing is very different.

Considering again the temperature sensor - the middleware knows the temperature is a floating-point number. It updates that number and then notifies all consumers to read the new value. It can even make judgments about the values, such as storing the last N values, only sending values that exceed a limit (e.g. to an alarm monitor application), or warning applications if the value has not been updated recently. The infrastructure itself manages the distributed state.

This means that the infrastructure is responsible for providing a consistent view of "truth". In a dynamic distributed system, it's not simple to have "absolute" truth, but the middleware can be crisp about exactly how reliable the information is, how old it is, how many past versions are saved, what delivery guarantees there are, what happens if a producer fails, etc. These "Quality of Service" (QoS) settings specify how to manage the state of each data item.

Like a database, data-centric middleware imposes known structure on the transmitted data. The infrastructure, and all its associated tools and services, can now access the data through CRUD-like operations. Clear rules govern access to the data, how data in the system changes, and when participants get updates. As in databases, this accessible source of consistent truth greatly eases system integration. Thus, the data-centric distributed "databus" does for data in motion what the structured database does for data at rest [Electronic Design].

Some general assumptions regarding data centric approach are:

- data-centric middleware maintains state of the published information;
- infrastructure manages the content;
- applications read and update a virtual global data store.

To compare message centric approach with data centric one it can be said that message centric systems usually send “verbs”, while data-centric systems update “nouns”. It is a key difference that verb-based system (incorporated messaging), applications interact directly with each other. In a data-centric noun-based system, applications interact with the data model, not directly with each other. This reduce somehow coupling.

It is also here worth to mention then in a message centric approach coupling can be even less. Thanks to the MOM often the sender even not knowing the receiver of his message, and is quite easily to exchange the receiver or extend the system to several receivers without any change on the message sender side.

Another important aspect Verb-based (message center) systems is fact they need sometimes many commands and actions, and this set grows with the number of applications. Noun-based systems support only a few actions on the data, and that set does not grow with the number of applications.

Data-centricity and message-centricity are not black and white opposites. For instance, message-centric designs can use databases or other data-centric means to manage state. Data-centric designs can guarantee acknowledgement of each update message. Some systems even combine both approaches. Nonetheless, infrastructure that fits the problem simplifies application code greatly. Choosing the right paradigm deserves careful thought.

Overall, a distributed application has a lot of state. If the infrastructure doesn't manage state, then every application must store and maintain its own state. Unmanaged state leads quickly to inconsistency. This is brittle; inconsistent systems break in many ways. With a data-centric approach, state is managed, redundant producer failover is natural, and multiple consumers can easily access consistent state. In many ways, a data-centric design is easier to make reliable and available.

There are many communications middleware standards and products, like DDS which is uniquely data centric.

However, delegating state control to the infrastructure has it´s cost. The system needs a shared data model, so data schema and properties must be globally agreed (or later mediated, a topic beyond this treatment). Messaging systems can be much laxer in their upfront data definitions. Keeping state control in the applications, while an added burden, also gives application developers more design flexibility.

Data-centric middleware disseminates and manages state. It doesn't send messages about the data; it sends the data itself. Consuming applications then pick up reflections of the global state. To do this, the middleware must know the data schemas, treat instances of data as known objects, and attach behaviors to the data, not to the flow. [Electronic Design]

Unlike MCPS MW, DCPS MW supports being “*told*” by the app tier pub-sub components which Quality Of Service (QoS) parameters are important to each of them. For example, a publisher can "*promise*" to send topic samples at a minimum rate and/or whether it will use a best-effort UDP-like or reliable TCP-like protocol for transport. On the receive side, a subscriber can tell the MW that it only wants to see every third topic sample and/or only those samples in which certain data-field-filtering criteria are met. DCPS MW technologies like DDS support a rich set of QoS parameters that are usually hard-coded and frozen into MCPS MW

5.3. Conclusion

Both Message centric design and Data centric methodology can be applied to connect complex systems.

In a message-centric design, the unit of information exchange is the message itself. The infrastructure's role is to ensure that messages get to their intended recipients. Like a file system, a message can include anything in any format. The infrastructure just passes the information around.

Data-centric technologies understand the data itself and can therefore manage different types of state behavior. The fundamental unit of communication is a data value. The infrastructure's job is not to just deliver messages, but to ensure that all nodes have a synchronized and common understanding of the data's value. Like a database, data-centric middleware understands different message formats and enforces access rules.

Data-centric middleware offers better scalability and better state management, at the cost of moving some complexity into the middleware.

AMQP (data centric) arose from the banking industry. It can process thousands of reliable queued transactions. Main focus is to on enable fast and reliable business transactions, tracking of all messages and ensuring that each message each is delivered as intended, regardless of failures or reboots. [Electronic Design].

DDS, a data-centric middleware standard has its roots in high-performance defense, industrial, and embedded applications. It can efficiently deliver millions of messages per second to many simultaneous receivers.

DDS further provides the real-time, many-to-many, managed connectivity required by high-performance machine applications.

For real implementation both technologies AMQP and DDS can be considered and have their strengths and weaknesses. Combining them can also be a good choice offering flexible solution for the future.

In summary the proposal of WP8 is for the data-centric solution as the most useful technology for integration layer purposes based on the features addressing fast machine to machine data exchange possibility and the ability to maintain state of the data with support for a long list of QoS attributes.

A recommendation for a specific technology is kept open as the succeeding project of WP6 will apply all the different possible technologies and find overall recommendation after comparison of real implementations.

6. Canonical Data Model

6.1. The CDM Pattern

WP8 has chosen to apply a Canonical Data Model (CDM) pattern, as described in ref. [EIP]. The CDM is an essential enabler of the interoperability between the services of the Integration Layer. System components may have different data representations internally, but whenever exporting or importing data to/from other components, they must translate this data to the canonical form. The result is that new communication paths may easily be added, as both end points are independent of the other end points internal data model. There is always one way of unambiguous translation and interpretation of data from the CDM to the connecting data models and vice versa, which is precondition for standardisation.

To further leverage on the benefits of the CDM, the same model shall be applied to IL and AF. There will for sure be some information items that only apply to one of the environments – nevertheless, they are still part of the same CDM.

Each application has basically three options how to adapt to the CDM

- apply the CDM format also for internal data representation. This may be suitable in case of new developments;
- implement a Messaging Mapper ([EIP]), i.e. an internal module which maps data between the applications domain objects and the CDM. This may be suitable for migrating existing applications to the IL/AF context;
- implement a Message Translator ([EIP]), i.e. an external adaptor, receiving data according to CDM which it relays according to some other interface specification, and vice versa. This is suitable for adapting legacy systems without changing them at all internally.

In the following, the expression the "Shift2Rail CDM" will be used to identify "our" CDM, in contrast to the general CDM concept. It is also used to indicate that there is intension for the model to be data model for a whole Shift2Rail project.

6.2. Data Modelling Approach

WP8 has started this activity looking at existing standards, to avoid starting from scratch.

A more comprehensive data model survey was done by the ON-TIME project in 2013 [ONT]. We have looked at their conclusions, but also added more recent observations of our own.

Interface specifications focus only on the aspects of data that need to be shared through the interface and is considered as not complete for CDM purposes. However, to implement standard interfaces between the IL and the different clients, contents described in a functional Interface structure in D8.4 will be covered by the Shift2Rail CDM.

6.2.1. Evaluated Data Models

6.2.1.1. railML v2 (timetable, rollingstock, infrastructure)

railML is a data exchange format developed by a European consortium of railway companies, academic institutions and consultancy firms. Although not officially endorsed as a standard by any legal authority, it has gained widespread acceptance with many IMs as a de facto standard.

Formed in 2002, the railML.org project aims to continuously develop this format in order to facilitate its use in a wide range of railway applications. The project started as a partnership between the Fraunhofer Institute for Transportation Systems and Infrastructure (FhG-IVI) and the Swiss Federal Institute of Technology's Institute for Transportation Planning and Systems, and is currently led by a small independent coordinators team.

railML is published as a series of XML schemata and sub-schemata, each of which encompasses an area of the railway domain. The current version of railML is 2.3, which specification contains subschemas for three main areas: infrastructure, timetable and rolling stock. These topics are themselves further divided into additional subschemas that address more specific areas.

6.2.1.1.1. Timetable

The railML® timetable sub-schema is focused on the description of the railway timetable including all its various facets that are needed by the data exchange applications. The railML timetable schema contains the following information:

- Operating Periods: The operating days for train services or rostering;
- Train Parts: The basic parts of a train as a sequence of operation or control points with the same characteristics such as formation and operating period. The train part includes the actual information regarding the path of the train as well as the corresponding schedule information;
- Trains: One or more train parts make up a train and represent either the operational or the commercial view of the train run;
- Rostering: Train parts can be linked to form the circulations necessary for rostering (rolling stock schedules).

6.2.1.1.2. Rollingstock

The railML® rollingstock sub-schema is focused on the description of the railways rolling stock including all its various facets that are deemed to be needed by the data exchange applications. In particular, the railML rollingstock schema contains the following information:

- Vehicles: The characteristics of individual railway vehicles or vehicle families are described in this part of the schema. The description of vehicles considers some general data used for organising assets like naming, classification or vehicle numbers as given by its operator. The majority within the schema is providing the structure to store the various technical aspects of railway vehicles with regard to their propulsion system, car body features, brakes or services installed within the vehicle;
- Formations: The features of train sets or parts of it formed of several different or similar vehicles are described in this part. This combination of vehicles is used to describe train features as needed e.g. in timetables. However, the logical consistency between the formation and the vehicles it is made of is not enforced by the schema. It must be ensured by the application producing the data.

6.2.1.1.3. Infrastructure

The railML® infrastructure sub-schema is focused on the description of the railway network infrastructure including all its various facets that are needed by the data exchange applications. In particular, the railML infrastructure schema contains the following information:

- Topology: The track network is described as a topological node edge model;
- Coordinates: All railway infrastructure elements can be located in an arbitrary 2- or 3-dimensional coordinate system, e.g. the WRG84 that is widely used by today's navigation software;
- Geometry: The track geometry can be described in terms of radius and gradient;
- Railway infrastructure elements enclose a variety of railway relevant assets that can be found on, under, over or next to the railway track, e.g. balises, platform edges and level crossings;
- Further located elements encompass elements that are closely linked with the railway infrastructure, but that "cannot be touched", e.g. speed profiles and track conditions.

6.2.1.2. RailTopoModel and railML v 3

In 2013, by initiative of SNCF Réseau, UIC launched the RailTopoModel (RTM) initiative. The goal is to create a comprehensive model for data exchange within the railway sector. As the name implies, the initial focus was on modelling the topology of the railway network, and a purely mathematical model of topology, positioning and location is also the scope of the first released version [RTM]. However, there is a roadmap aiming to “support the whole Railway

System description, life cycle and operation" by addressing also functional assets, physical assets, geometry and time/project dimensions.

At an early stage, a cooperation was established between RTM and railML. A version 3 of railML was published 31 October 2017. It is compliant to RTM. The main difference between the two is that RTM is a conceptual model, described in UML, whereas railML is an implementation of said model into a detailed XML schema, which can act as a specification of actual data storage or exchange.

There are some substantial differences to railML v2:

- within the Infrastructure domain, there is a clear division between topology, geometry, functional and physical assets. These data were interleaved to a large extent in railML 2;
- the structure is simpler and "flatter" (less hierarchical levels). This makes it easier to isolate information related to single objects/entities, which is useful for the mainly messaging-based contexts of IL and AF;
- an Interlocking domain is planned to be added, providing the logical links between infrastructure assets and interlocking logic definitions;
- the topology model (from RTM) is quite complex. This is partially due to the ambition to keep the topology model purely mathematical, without links to physical assets or even dimensions. Partially it is because of an aggregation feature, which allows applications to "zoom out" to a description level of less detail, e.g. a network of stations and lines instead of a network of tracks and switches.

6.2.1.3. RINF

The European Register of Infrastructure [RINF] refers to Article 49 of Directive (EU) 2016/797 and provides for transparency concerning the main features of the European Railway infrastructure. The common technical specifications are set out in a Commission Implementing Decision (RINF Decision).

The most recent RINF Decision (Decision 2014/880/EU from 26 November 2014) repeals the previous Decision 2011/633/EU and introduces a computerised common user interface (CUI) which simplifies queries of infrastructure data. This interface, set up and managed by the European Railway Agency, is publicly available.

Furthermore, the RINF Decision obliges each Member State to nominate an entity (NRE) in charge of setting up and maintaining its register of infrastructure and to notify an implementation plan.

The primary purpose of RINF is to support technical compatibility between fixed installations and rolling stock within the European community.

For that purpose, the railway network is at the macro-level a series of operational points and sections of line. At the micro-level, subsystem features are assigned to infrastructure elements, such as tracks and sidings. Ultimately macro- and micro-level should be presented in terms of digital maps.

For RINF:

- the railway network is a series of operational points (OPs) connected by sections of line;
- a line is a sequence of one or more sections, which may consist of several tracks;
- a section of line is the part of line between adjacent OPs and may consist of several tracks;
- operational points are locations for train service operations for example where train services can begin and end, change route and where passenger or freight services are provided;
- stopping points for passengers on plain line are also regarded as Ops;
- operational points may be locations where the functionality of basic parameters of a subsystem are changing for example: track gauge, voltage and frequency, signaling system;
- operational points may be at boundaries between MSs or IMs;
- passing loops and meeting loops on plain line or track connections only required for train operation do not need to be published (however, if parameters change at the connection it would be considered an Operational Point and included in the register);
- sidings are all tracks not used for train service movements.

Figure 6.1 shows an example of the railway network structure of RINF, the elements of which belonging to different IMs.

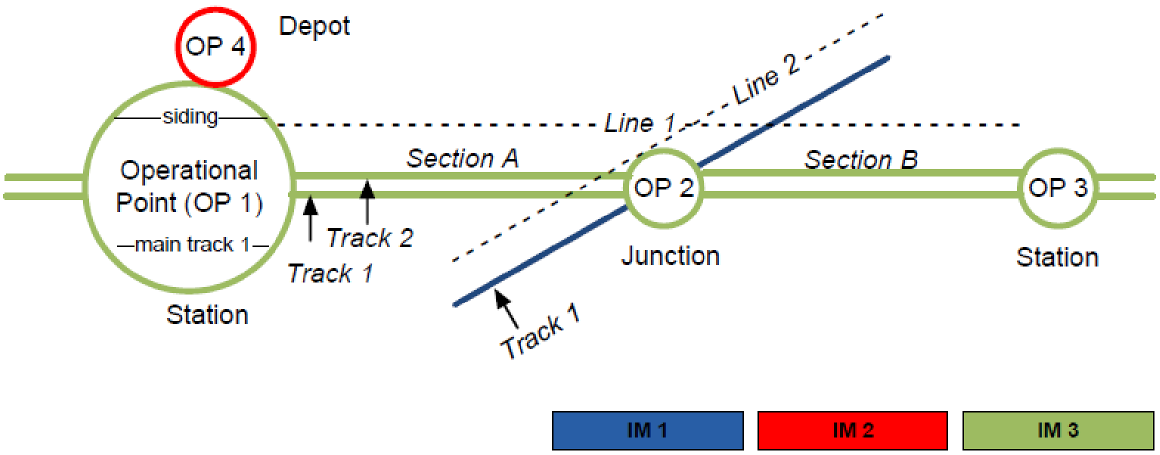


Figure 6.1: Structure of the railway network for the register [RINF]

Items collected in RINF must be accessible for end-users (process of data retrieval). This requires an implementation using IT-means with the need to define a harmonized model of the railway system.

Regarding the transition period for infrastructures placed in service the RINF WG decided to set the final deadline for transition to five years. All types of network shall be included in RINF within five years after entry into force (1st January of 2015) of the RINF specification.

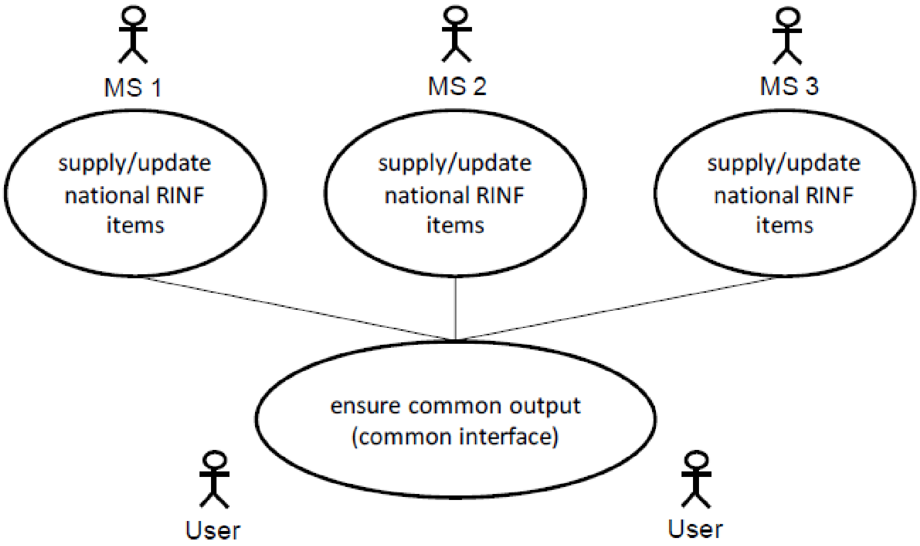

MS 1
MS 2
MS 3
supply/update national RINF items
supply/update national RINF items
supply/update national RINF items
ensure common output (common interface)
User
User

Figure 6.2: Principle of common interface for RINF [RINF]

Providing data to RINF can be done via web application under "Data management" where user has possibility to upload XML dataset compressed within a .zip file. The XML file is then validated against the corresponding XML Schema Definition (XSD).

From Shift2Rail CDM perspective all data covered by RINF should also be contained in CDM, to enable a bridge between the systems and thereby ensure data consistency.

6.2.2. Interface specification

6.2.2.1. TAF/TAP TSI

The Telematic Applications for Freight / Passenger Services Technical Standards for Interoperability ([TAF], [TAP]) are EU regulations specifying the exchange of information between relevant stakeholders, in order to enable cross-border rail services.

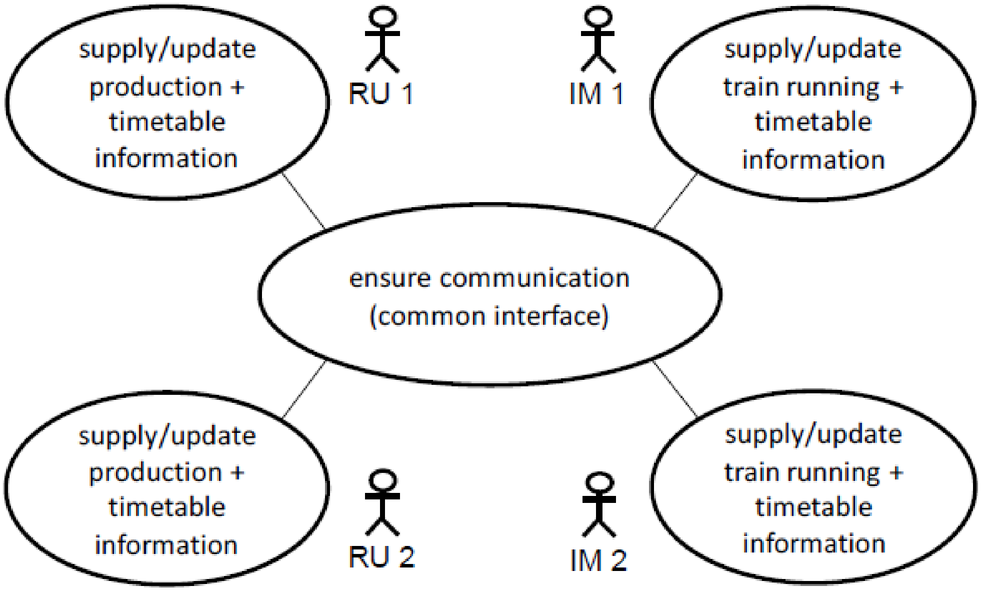

supply/update production + timetable information
RU 1
IM 1
supply/update train running + timetable information
ensure communication (common interface)
supply/update production + timetable information
RU 2
IM 2
supply/update train running + timetable information

Figure 6.3: Principle of common interface for TAF/TAP TSIs [TAF] [TAP]

All RU/IM messages described in TAP are common with TAF (which contains additional messages specific for freight traffic). The common processes are related to path allocation, train readiness, train running reporting and service interruption. Consequently, messages are harmonised between TAP and TAF and gathered in the same data model.

The Technical Specification for Interoperability on “Telematics Applications for Passengers” (TAP TSI) prescribes protocols for the data exchange of:

- timetables;
- tariffs;
- reservations, fulfilment;
- information to passengers in station and vehicle area;
- train running information;
- etc.

which must be expected by the European rail sector (railways, infrastructure managers, ticket vendors etc.) according to the European Rail Passengers’ Rights Regulation EC/1371/2007 and to the Interoperability Directive EC/2008/57.

Data formats used in TAP TSI:

- EDIFACT (timetabling);
- Fixed length text files (tariff data);
- Binary messages (reservation messages),
- XML-messages (home printed tickets, PRM reservation).

The Technical Specification for Interoperability on “Telematics Applications for Freight” (TAF TSI) drafted by ERA prescribes protocols for the data exchange of:

- Path request;
- Train Running Forecast;
- Service Disruption Information;
- Shipment Estimated Time of Interchange / Arrival,
- Etc.

TAF TSI prescribes furthermore databases which must be implemented by European RUs, IMs, or Freight Customers:

- Reference Files (such as location ID, company ID etc.);
- Rolling Stock Reference Databases;
- Wagon and Intermodal Unit Operational Database;
- Trip plan for wagon / Intermodal unit.

TAF TSI prescribes the mandatory use of a so called “common interface” which is mandatory for all RUs and IMs:

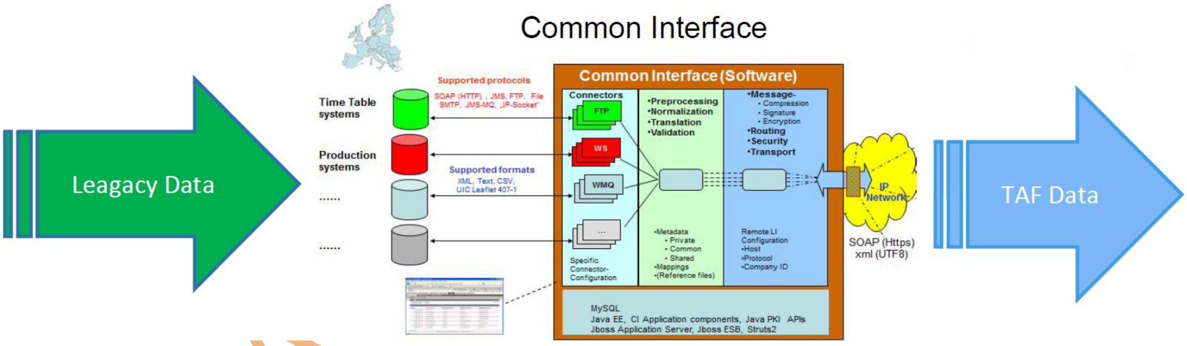


Figure 6.4: Common interface for TAF TSI [TAF]

The Common Interface support following connectors that are used by existing legacy systems:

- IP socket connector (customized application protocol);
- JMS connector;
- MQ (IBM)/JMS connector:
  - FTP Connector,
  - FS Connector,
  - SMTP Connector,
  - Web Service Connector.

Supported data formats are:

- XML;

- Text;
- CSV;
- UIC 407-1.

Of interest to the Shift2Rail CDM is data exchange RU-IM and IM-IM.

6.2.2.2. EULYNX

The EULYNX (European Initiative Linking Interlocking Subsystems) [EULYNX] project is an initiative by ten European IMs, which aims to create many Standard Communication Interface specifications between trackside technical systems, with the electronic interlocking (ILS) at the heart of the architecture.

EULYNX project includes items like system architecture, modelling & testing, data preparation, interfaces between interlocking, interfaces to track vacancy detection and adjacent interlocking or signaling subsystems.

Results of previous European initiatives concerning interlocking system standardization (e.g. Euro Interlocking, INESS and ERTMS) provide the basis for the project.

The EULYNX initiative acts as a cooperation based on a mutually accepted agreement following democratic principles and membership fees. The project community expects to have different kinds of partners: rail infrastructure managers as core members, other active members like signaling or industrial partners, engineering bureaus and universities.

The current phase of the project will provide a full set of specifications. The project has started on 19 February 2014, with a three-year lifespan for this stage. After three years the project organization will evolve into a standing organization for standardization of interfaces, based on a full set to be published in 2017 (Baseline Set 1 - partly released in March 2017 rest of documents planned to release in June 2017; Set 2 is scheduled for December 2017 – including formal models).

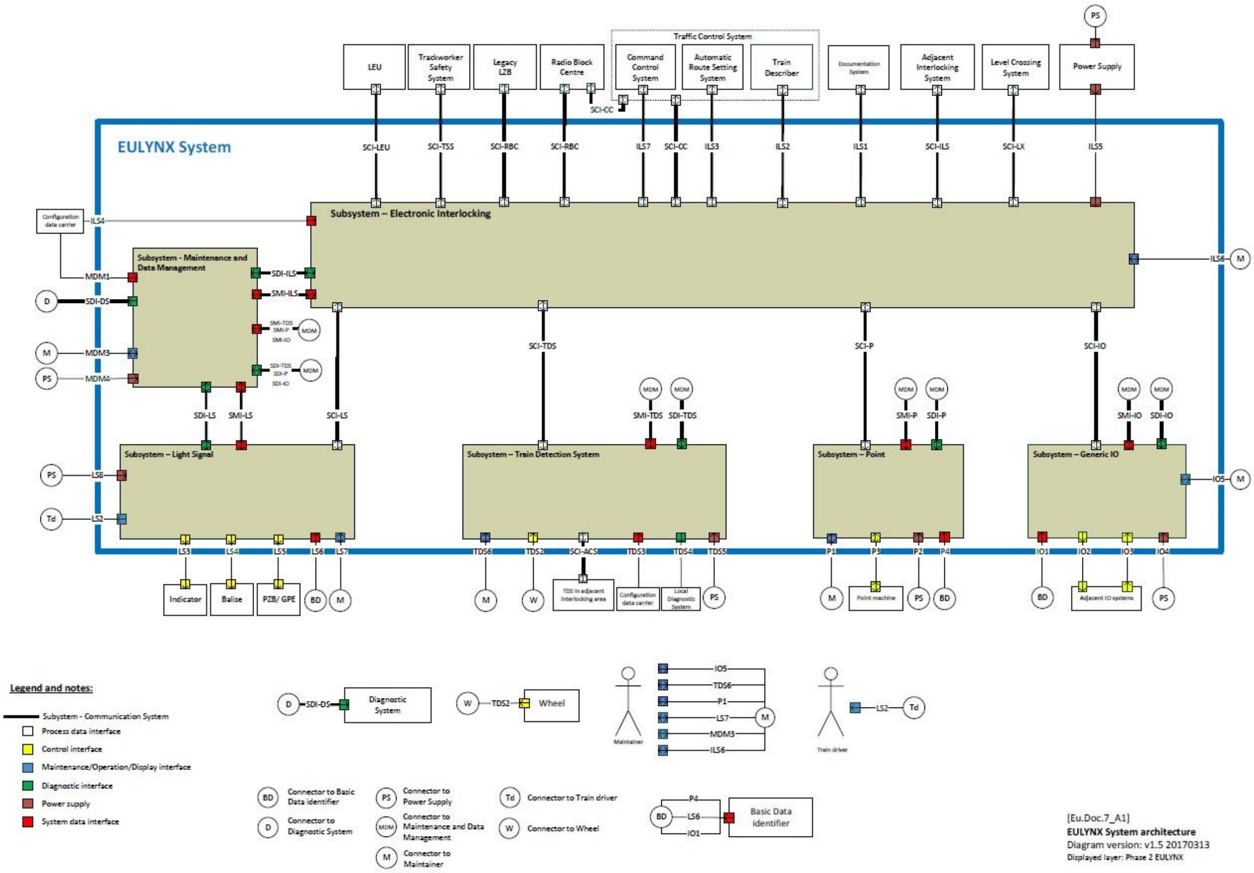


Figure 6.5: EULYNX System architecture [EULYNX]

6.2.2.3. ETCS

ETCS [ETCS] is the European Train Control System promoted by the European Commission for use throughout Europe, and specified for compliance with the High Speed and Conventional Interoperability Directives.

ETCS is in fact an automatic train protection system, based on in-cab signaling and spot and/or continuous track to train data transmission. It ensures that trains receive safe movement authority directly presented to the driver on the cab display. The ETCS Messaging is described in TSI CCS Subset 026.

6.2.2.4. ATO

ATO for mainline trains is a “Game Changer” for rail traffic in the coming years.

ATO takes over partially or entirely the tasks of the train driver. These range from controlling the train’s speed by braking and accelerating to driving up to a precisely defined stopping point. ATO thus allows regular operation to be automated so that all trains run with the same, optimal speed profiles.

Mandatory inputs for any ATO system coming from TMS are the time table and infrastructure (Topology) Data. A specific communication between TMS and Trackside ATO installations is necessary.

For the time being a strong focus is on the methodology to apply ATO under ETCS supervision. The development of specifications is part of X2RAIL-1 but not finished. Therefore ATO has not been part of the scope covered from IN2RAIL WP8. The propriate enhancement of the Data Model is allocated to the works proposed to be executed in X2RAIL-2 WP6.

The draft specification for “ATO over ETCS” describes train trips in terms of Journey Profiles, composed of Segment Profiles, Timing Points and contains the specification of relevant infrastructure information e.g. max speed limits, tunnels, track gradients, curves, neutral sections, etc. All these entities need to be added to the CDM in the S2R program.

6.2.3. Conclusion

Different Data Models have been analysed to evaluate requirements imposed to the new Integration Layer design.

WP8 in coordination with WP9 decided to use railML V2 scheme as a starting point to create an IN2RAIL/S2R Canonical Data Model.

This decision how to start up was backed up by conclusions taken by the ON-TIME project [ONT].

During the progress of the project it became clear that the targeted CDM is different from RailMl.

It has been agreed in WP8 to use in the first step the railML Class Diagrams presentation style as many experts involved in IN2RAIL WP8 are also members of the railML community and bring high degree of experience to the table.

The elements for the Class Diagrams needed to cover the scope of IN2RAIL have been defined and are described in D8.4.

The model is easily expendable adding elements of other services to the structure.

Figure 6.6 illustrates and summarizes the process taken from WP8 to derive from the initial evaluations of railML V2 to a sound proposal for an In2Rail/S2R Data Model.

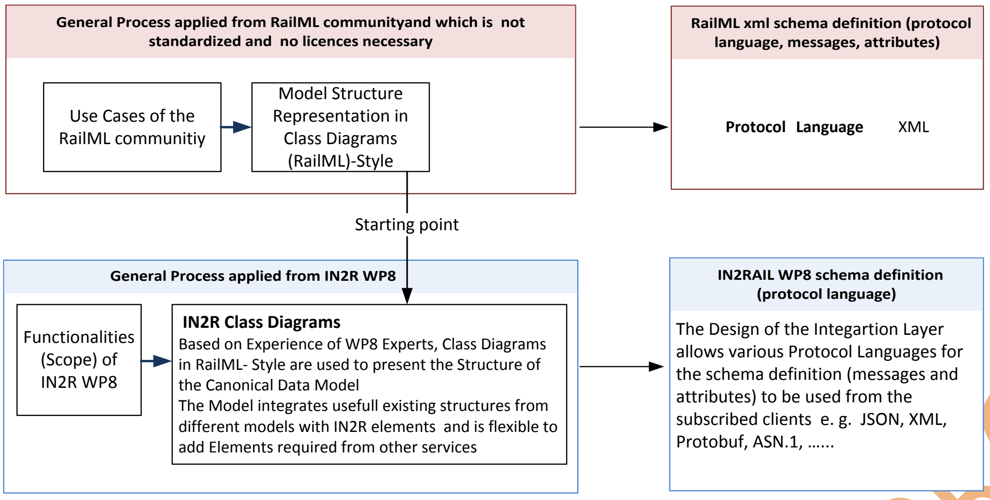

General Process applied from RailML communityand which is not standardized and no licences necessary
Use Cases of the RailML communitiy
Model Structure Representation in Class Diagrams (RailML)-Style
RailML xml schema definition (protocol language, messages, attributes)
Protocol Language XML
Starting point
General Process applied from IN2R WP8
Functionalities (Scope) of IN2R WP8
IN2R Class Diagrams
Based on Experience of WP8 Experts, Class Diagrams in RailML- Style are used to present the Structure of the Canonical Data Model
The Model integrates usefull existing structures from different models with IN2R elements and is flexible to add Elements required from other services
IN2RAIL WP8 schema definition (protocol language)
The Design of the Integartion Layer allows various Protocol Languages for the schema definition (messages and attributes) to be used from the subscribed clients e. g. JSON, XML, Protobuf, ASN.1, ……

Figure 6.6: Process towards an In2Rail/S2R Canonical Data Model

6.3. Central Modelling Concept

6.3.1. Modelling Principles

WP8 has established the following fundamental modelling principles for the Shift2Rail CDM:

- **Model Driven Architecture:** the CDM is to be a platform independent model, defined in a UML class diagram. It is to define the entities, their attributes and their relations, without being concerned about exactly how this is to be implemented in an application. From this, many platform specific implementations, e.g. a transmission format, a database schema or a software module, can be derived;
- **Incremental growth:** we cannot hope to cover all present and future needs of all railway-related systems in the first CDM release. Instead, it will grow incrementally according to user needs. Each Shift2Rail project will define their own part of the CDM, based on their use cases. The CDM will have a structure where data can be easily added, without disrupting the existing definitions;
- **Adaptable:** each conceivable piece of information cannot be included in the model. The model itself shall cover generic aspects of commonly used entities. For local specialisations, there shall be a defined method for extending the model;
- **Divisibility:** equally important to incremental and local extensibility is to be able to structure the complete model in divisible parts, for example by namespaces or modules. Application developers should not be concerned about model complexity outside the needs of their individual applications;
- **Consistent design patterns:** when the model is extended, for new use cases or for local adaptations, it needs to be done in a coherent and consistent way. For this reason, the design patterns to use are documented in [D9.1A].

6.3.2. Model Driven Architecture

The Canonical Data Model is represented in UML diagrams. This approach supports the MDA (Model Driven Architecture) up to software development processes proposed by OMG (Object Management Group) [MDA OMG].

The architecture reflects the specification of the parts of the system and the rules for the interactions of the different elements represented by connectors.

MDA specifies three default viewpoints (an abstraction technique for focusing on a set of concerns within a system while suppressing all irrelevant detail) on the system:

- computation independence;
- platform independence;
- platform specifics.

The computation independent viewpoint focuses on the context and requirements of the system without consideration for its structure or processing.

The platform independent viewpoint focuses on the operational capabilities of a system outside the context of a specific platform (or set of platforms) by showing only those parts of a complete specification that can be abstracted out of that platform.

A platform specific viewpoint augments a platform independent viewpoint with details relating to the use of a specific platform.

Corresponding to the three MDA viewpoints defined above MDA specifies three default models of a system:

- Computation Independent Model (CIM): a CIM is also often referred to as a business or domain model because it uses a vocabulary that is familiar to the subject matter experts (SMEs). It presents exactly what the system is expected to do, but hides all information technology related specifications to remain independent of how that system will be (or currently is) implemented. The CIM plays an important role in bridging the gap which typically exists between these domain experts and the information technologists responsible for implementing the system. The CIM can exist in form of business requirements, uses cases or other free text system description;
- Platform Independent Model (PIM): a PIM exhibits a sufficient degree of independence to enable its mapping to one or more platforms. It is a UML model that is independent of the specific technological platform used to implement it, is an abstract model which contains enough information to drive one or more Platform Specific Models (PSM);

- Platform Specific Model (PSM): a PSM combines the specifications in the PIM with the details required to stipulate how a system uses a particular type of platform. In other words: the PSM is a more detailed version of a PIM. Platform specific elements are added.

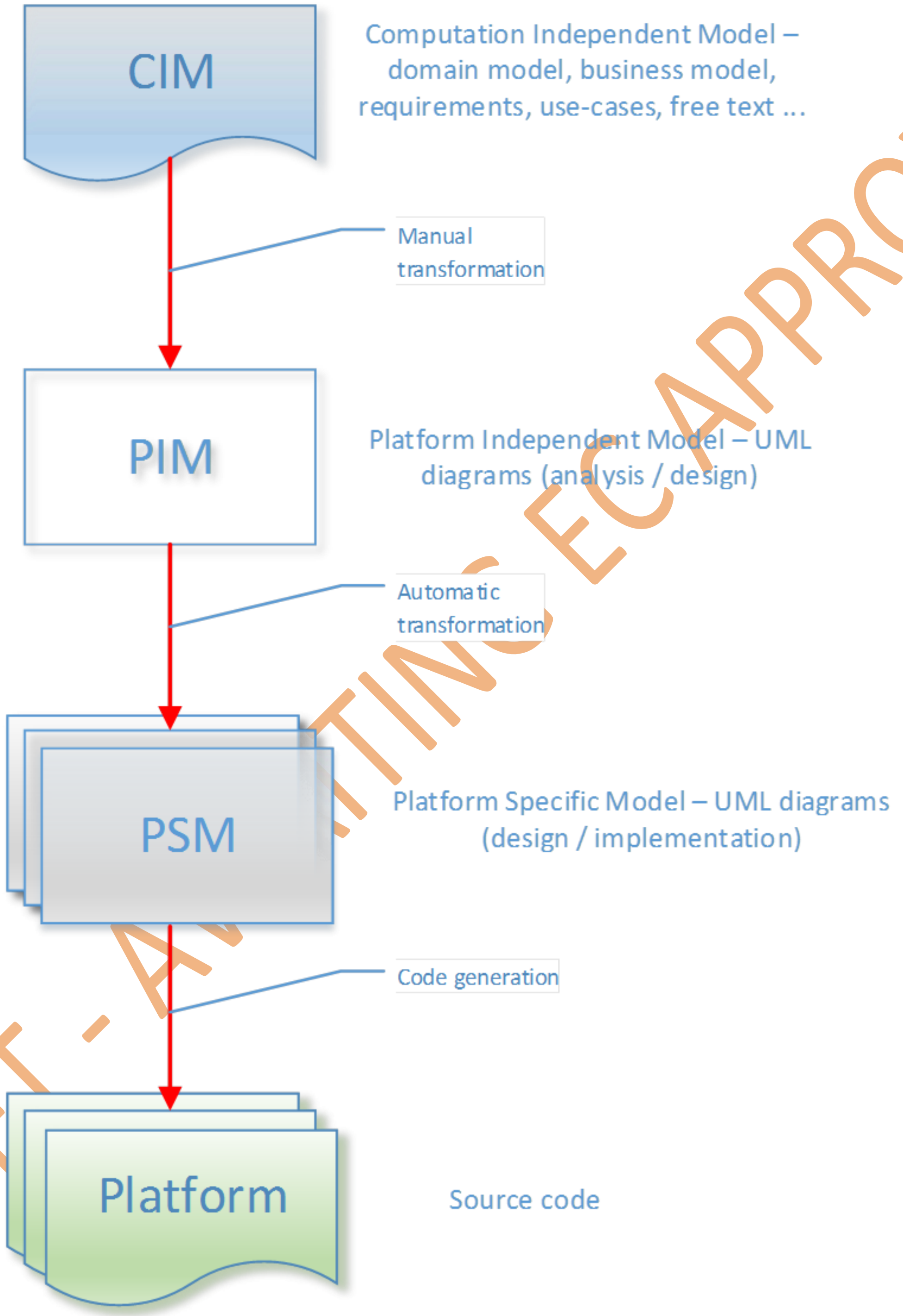

CIM
Computation Independent Model – domain model, business model, requirements, use-cases, free text ...
Manual transformation
PIM
Platform Independent Model – UML diagrams (analysis / design)
Automatic transformation
PSM
Platform Specific Model – UML diagrams (design / implementation)
Code generation
Platform
Source code

Figure 6.7: MDA transformation

An important aspect of MDA is model transformation. It is the process of converting one model to another within the same system. The transformation combines the platform independent model with additional information to produce a platform specific model.

An MDA mapping provides specifications for how to transform a PIM into a particular PSM. The intent is clearly to automate as much of the process as allowed by the toolset in use.

The final step in a transformation process is to generate the implementation code as well as perhaps other kinds of supporting artefacts such configuration files, component descriptor files, deployment files, build scripts, etc. The more fully the application semantics and run-time behaviour can be included in the PSM, the more complete the generated artefacts can be.

Depending on the maturity and quality of the MDA toolset, code generation varies from significant to substantial or, in some cases, even complete. Note that even minimal automation simplifies the development work and represents a significant gain because of the use of a consistent architecture for managing the platform-independent and platform-specific aspects of applications.

Here we do not want to oblige our self to follow each of MDA principles, but we are inspired of MDA and want to follow its idea.

6.3.2.1. MDA tools

railML and RailTopoModel use Enterprise Architect from Sparx company [Enterprise Architect] to define a model according to MDA principles.

The Enterprise Architect offers number of tools supporting MDA driven development. The core is a template based transformation engine that generates PSM model elements from a PIM source. It offers also code generation templates to C#, DDL, EJB, Java XSD and others.

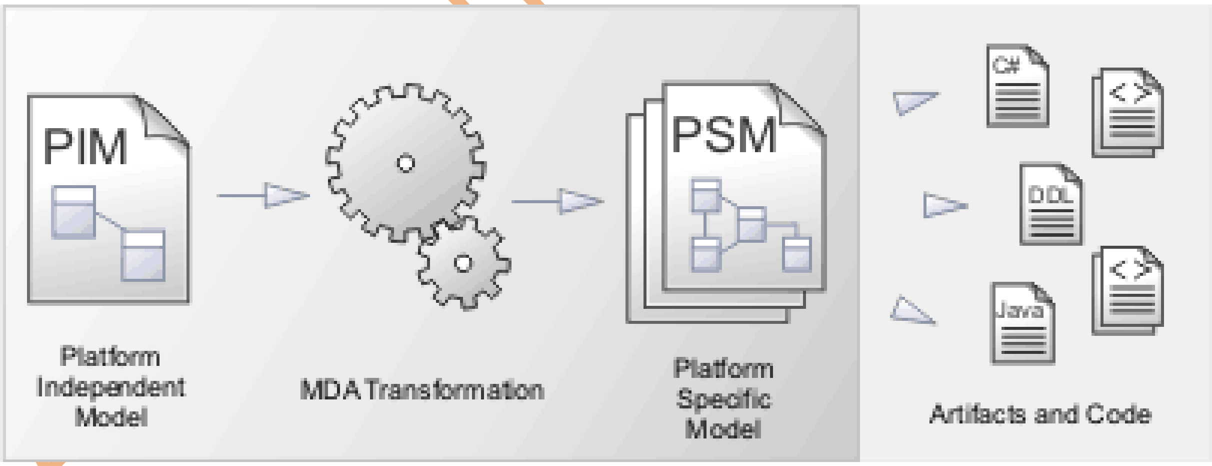


Figure 6.8: Enterprise Architect - Model Driven Architecture (MDA) Transformation [Enterprise Architect]

Enterprise Architect provides a facility to create your own transformations; this can be useful to automate the process of generating more specific models from more general ones, reusing the transformation and preventing errors from being introduced as they might if the models were created by hand.

All transformations in Enterprise Architect create an intermediary language form of the model to generate. You can access and edit the file containing this intermediary language code using an external editor.

The MDA Transformation templates are similar and have the same syntax as, the code generation templates within EA.

At the end the newly added or modified transformation / code templates can be exported to external XML file and exchanged with other users.

Next to Enterprise Architect there are other tools like Eclipse Modelling Framework (EMF) that can support the MDA process too. It contains dedicated frameworks and tools that allow to develop model-to-model transformations and model-to-text transformations. It is possible to export data model directly to the XML format. Next with proper generator to transform it into Protocol Buffers schema or JSON file when needed.

6.4. Structure of The Canonical Data Model

6.4.1. Scope and objective of the Shift2Rail CDM

This section is a summary of the Design Patterns for the CDM and is part of WP9 from [D9.1A].

All data, which needs to be exchanged in the In2Rail platform context needs to be addressed by the Shift2Rail CDM. This includes data about physical objects (trackside and rolling stock) and logical entities (e.g. traffic restrictions, operational areas, crews) of the railway system, data about the control system itself and its users, as well as external data that is relevant to the planning and execution of rail traffic.

The design patterns described in [D9.1A] define the guidelines how the CDM can be gradually extended towards this goal, according to the individual needs of the different projects. It is the first definition of those patterns, which may be adapted during the progress of the succeeding activities of the Shift2Rail program.

The decision taken by the TMT of IN2RAIL to allocate these works to WP9 was based on the availability of key experts in WP9. The budgets have been transferred from the different WPs to WP9 accordingly.

6.4.2. Tree structure

The model is developed according to the principles of Object Oriented Design (OOD). Each real-life entity is modelled as an *object*. Data about the entity are *attributes* of the object. The “template” describing objects of the same type is a *class*. Hence, the class lists a set of attributes, while the object (an instance of the class) fills the attributes with individual values. The class is part of the data model, while the object is part of a data set, based on the model.

Inheritance is used, meaning that one class may inherit the attribute set of a base class and then add its own attributes, thereby creating a specialisation. If the base class models a category of entities, the inherited class models a narrower subcategory.

Aggregation is used to form an ownership hierarchy, where each class is contained by another class, making the complete data model into a tree structure of classes (not to be confused with the inheritance hierarchy, which may also form tree structures).

More detailed parts of the tree are presented in Deliverable D8.4. The work on data model will be continued in X2RAIL-2 WP6 Task 2 and IMPACT-2 WP7 to cover the extended scope of these projects.

The model defines an ownership hierarchy as a tree structure, where each object except the model root is contained by one specific other object.

There are distinguished five levels under the root in the ownership hierarchy: **/domain/view/container/object/component**. Deviations to this pattern are allowed and may be necessary in some cases. The significance of the different levels is as follows:

- the domain is currently one of Infrastructure, Timetable, Rollingstock and Interlocking. Only if there is a need to model data that is outside of all these domains should another domain be added. This needs to be discussed with the core coordination team;
- a view (term introduced in cooperation with railML community) collects data about a specific aspect of the domain, which is of interest to a specific group of use cases but can be excluded in other use cases. Current views in the Infrastructure domain are Topology, Positioning, Geometry, FunctionalInfrastructure and PhysicalAssets. Further views are suggested in the specific patterns below. New use cases may result in new views – to be discussed with the core coordination team;
- a container represents a type of objects within the view, which is functionally different from the objects in other containers. Example containers in FunctionalInfrastructure are switches, crossings, tracks, signals…;
- an object represents a physical or logical entity of the type indicated by the container. Note that one real life entity may be represented by objects in multiple views (e.g. a switch will have one projection in FunctionalInfrastructure and another in PhysicalAssets).

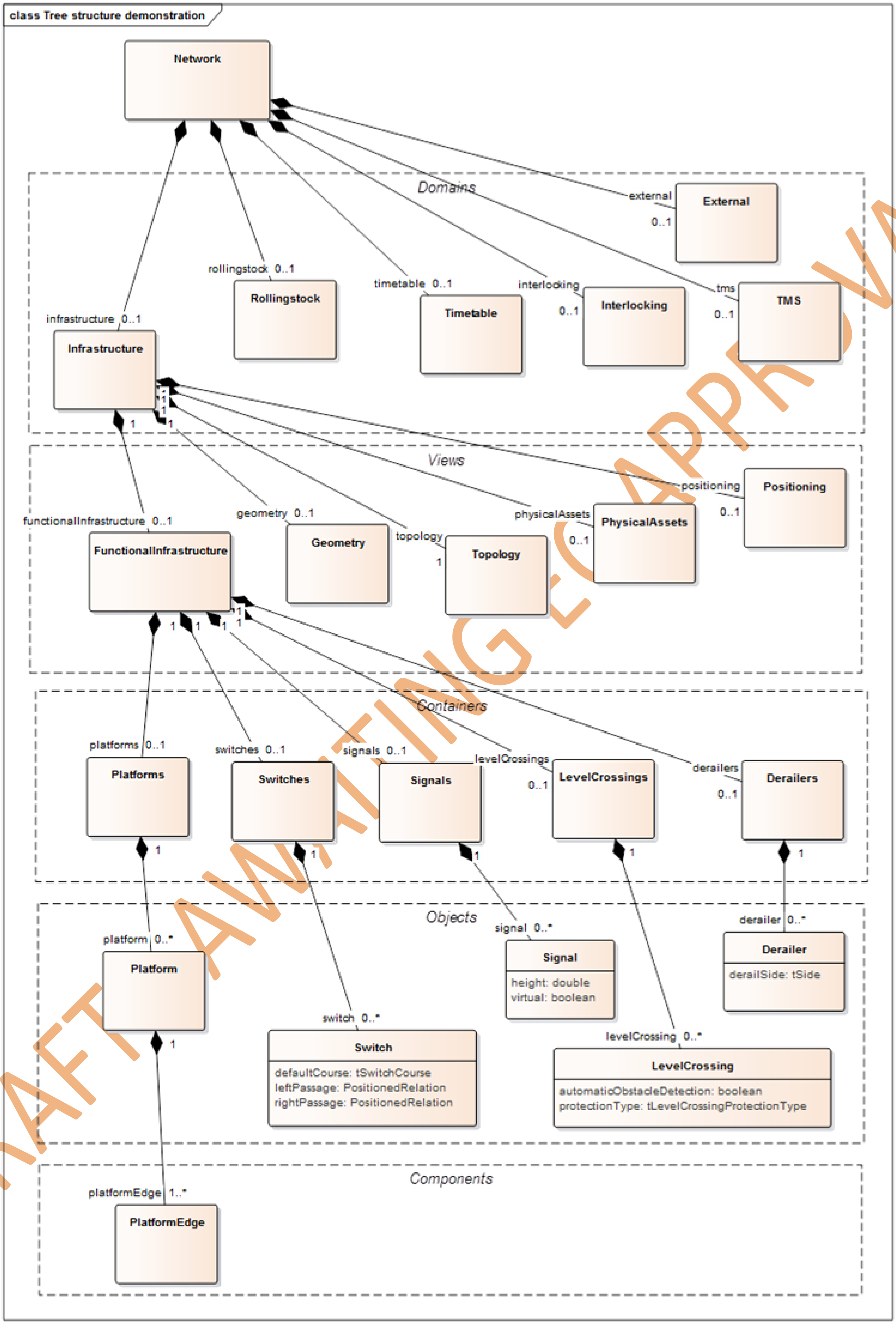


Figure 6.9: Data model tree structure illustration

6.4.3. Referencing

As mentioned above, the CDM maps all data in a system instance on to a large tree of objects. But there must be a possibility to access an object somewhere in the middle of the tree, e.g. to send its data in a message. Also, objects must be able to reference each other, across the tree structure. In data management terms, we need to establish a structure of *key/value pairs*, where the value is the data structure representing the object, and the key is the identity that enables an API to locate the desired object.

Class “BaseObjects” are defined from which all classes that represent “addressable” objects shall inherit. An attribute of BaseObject is a unique identifier. This identifier is a key to the value of the object.

A key does not necessarily refer to a leaf object in the data tree. It may also point to a higher level, in which case the whole branch of objects from that point constitutes the value.

This referencing method requires that the identifier of the object is known.

More efficient method of referencing is usage path expression based on the objects position in the data tree structure. In such situation the search path can be extended with object identifier, however can be used also when the identifier of the object is not known at all.

E.g. `/network/infrastructure/functionalAssets/switches[0]` is a key which refers to the value of the first Switch object in the FunctionalAssets container.

6.4.4. Topics

It will be necessary in many cases to limit access to some data to specific systems and/or specific users. This may be due to legal, contractual, operational, intellectual properties or other reasons. But making that kind of limitations on the level of individual objects would be impractical. We need to define groups of data, for which it becomes logical to either allow or block access to the whole group. We define such a group of data as a *topic*. Technically, a topic is a representation of a composition relation in the data model tree, which can be addressed by a key.

Defining the topics is closely tied to defining the data model, and the two activities need to be performed iteratively. The data model needs to be structured according to the desired topics, but the topics cannot be set before the data model contents are known. For this reason, a list of topics will not be presented in this document.

When selecting the topics, the following rules may serve as guidelines:

- objects that normally belong to different organisations should be separated in different topics, e.g. infrastructure assets, rolling stock, energy system;
- objects that interact closely should be in the same topic, e.g. signals, switches and other signalling elements;

- safety related data should be separated from non-vital data;
- the three main classes of data persistence spans should be separated, i.e. static data (e.g. topology data), dynamic data (status information, which changes in real time) and historic data (e.g. event logs);

A topic would normally correspond to a “view” in the data model, as described in Chapter 6.4.2.

6.4.5. Filtering patterns

In the publish–subscribe model, subscribers typically receive only a subset of the total messages published. The process of selecting messages for reception and processing is called filtering. There are two common forms of filtering: topic-based and content-based.

In a topic-based system, messages are published to "topics" or named logical channels. Subscribers in a topic-based system will receive all messages published to the topics to which they subscribe, and all subscribers to a topic will receive the same messages. The publisher is responsible for defining the classes of messages to which subscribers can subscribe.

In a content-based system, messages are only delivered to a subscriber if the attributes or content of those messages match constraints defined by the subscriber. The subscriber is responsible for classifying the messages.

Some systems support a hybrid of the two; publishers post messages to a topic while subscribers register content-based subscriptions to one or more topics. [Publish-Subscribe pattern - Wikipedia]

6.5. Metadata

6.5.1. Definition and context

Metadata are data on data. In the WP8 context, they are data on data which are resident in the datastore (and described by the CDM). Usual metadata include for instance the date at which some data has been changed. This specific information is already maintained by the IMDG, so, in this case, the data exist, but there is no standardized method to access this data in the frame of WP8 compatible development. This example shows that three aspects on metadata are to be considered:

- generating the metadata (in the above example, it is generated by the IMDG itself, when data is published);
- storing the metadata (in the above example, the storage is imposed by the IMDG implementation)

- accessing the metadata (in the above example, the access requires to use the IMDG proprietary API).

6.5.2. Usage

In the different use cases transferred from WP7 and WP9, none was requiring the usage of metadata, mainly for the following reason: when some data was required to fulfil the needs of some use case, then the data was added (or already present) in the data described by the CDM. Nevertheless, we can foresee cases when use of metadata could be useful.

For instance, we can imagine usage specific to WP8 general objectives or list some known usage of metadata given by literature (MASTER DATA MANAGEMENT AND CUSTOMER DATA INTEGRATION / BERSON & DUBOV / 226349-0 has been used):

- Tagging data to accelerate retrieval (Data synchronization): depending on IMDG implementation, it could be possible to tag some data or data set, to trigger synchronization of this piece of data to some IMDG node, which could require fast access to this data;
- Statistics, and optimisation based on these statistics: metadata on data access (read or write) could be used to optimize the implementation of the IMDG the network;
- Future security constraints: at this time, we do not know what will be the IT security constraints in few years, but we know that security plans for critical infrastructure (including train application) will impose to keep up-to-date with these future unknown constraints. Meta-data could be a good place for storing data related to these constraints, without changing the business data structure (CDM), even if security data change structure several times in the life of the system;
- Extension that impact less compatibility than data model changes: to explore new algorithms or procedures, it can be necessary to store or exchange additional attributes in CDM nodes. It may be more effective to conduct this exploration using metadata, as it as less impact on neighbour application. When extension is fully accepted, then it can be incorporated in the CDM using the backward compatibility extension mechanism;
- Enforcing or assisting in maintaining consistency of definition: this possible usage of metadata supposes that metadata include data describing the CDM. Then, metadata could be used to audit changes and check some referencing constraints. Nevertheless, in the case of In2Rail, the definition of the CDM can be also processed to help enforcing the same consistency. Therefore, this usage of metadata seems redundant with other already available data;
- Data lineage: IL and AF already specify mechanism to ensure that data are accessed only with appropriate rights (especially using topic feature). In use cases that have

been detailed in WP8, the access rights are given to applications and modules and not to operators. Operators' rights should be managed by applications, and they are not managed directly by the IMDG. If more detailed access information are needed (which operator was using some application to issue a command), this could be stored in metadata to improve the traceability of data changes;

- Auditing: each organization can require specific auditing rules on the data it uses. Metadata can be used to tag data or data type which should be audited, or to indicate when the data was audited, or which rule is applicable for this data or data type;
- Data cleansing: metadata can help maintenance of data store. Statistics on data access can be used to move some data to IMDG nodes which have larger storage capacity but lower performance, or data can be tagged for future deletion, or data type can be tagged for cleansing strategy;
- Support for ETL (Extract Transform Load) tooling: metadata are often required or improving performance of ETL tools by holding information on preferred external format and date, and logical name in previous extraction. Of course, the data structure (defined also by CDM) can also be stored in metadata for these tools, but we can assume that, as the CDM defines this structure, it is already available for such tools.

6.5.3. Implementation

Depending on the precise metadata which is addressed, the implementation of all three aspects mentioned above (generating, storing, accessing) must be implemented differently. This section defines a frame which is flexible enough to allow efficient implementation to be developed for each problem for which metadata concept can be selected as part of the global solution.

6.5.3.1. Metadata generation considerations

The first constraints to be considered is that elements of IL or AF which are not using metadata should not be impacted by metadata implementation, even when they participate (indirectly) to the value of stored metadata. For example, if a TMS module computes some data, it should not be requested to call a special interface so that the metadata generating module is aware of this publication; the simple fact that it publishes the data should, if required, trigger the metadata update. This requires that the metadata generation modules are inserted between the actual data storage (in the DDS) and the module generating the data. The best place for achieving that is the implementation for the generation part of these metadata is the IL-Wrapper (see D8.3).

Yet, in some cases, the generation of the metadata does not require such tight integration with DDS. For instance, some statistics like number of updates can be obtained by subscribing to changes; In that case a specific constituent can be added to the IL or AF to generate these metadata.

Finally, some metadata can be generated before runtime. E.g. metadata describing the CDM can be obtained by parsing and processing the CDM definition.

6.5.3.2. Metadata storage considerations

For the metadata storage, the main point to consider is the decentralized nature of DDS. This implies that if data generation is centralized, then metadata storage can be centralized too, but if metadata generation is decentralized, this should be considered in the storage, and DDS is also a good place for metadata storage.

From this it appears that DDS storage is fully compliant with the storage of metadata. This is true even if some metadata have also some private storage. We can imagine for instance that metadata describing the CDM (produced before runtime) are stored in some file. One constituent, responsible for publishing these metadata is reading this file, and publishing the corresponding data to the DDS. Private data exist (the file), but data are also distributed through the DDS.

6.5.3.3. Metadata access considerations

Some metadata are already generated by the DDS implementation. For this data, the best choice is to add specific functions to access (read) this data in the IL-Wrapper constituent. This concern access to:

- owner of some data (which constituent writes the data);
- last update time of the data.

Other data is stored in the DDS (as indicated above). So, the logical solution is to use CDM to access this data. This CDM is different from the application data CDM and is named the MCDM (Meta-CDM). The separation allows to use MCDM to conduct cost effective experimentation and changes to metadata, as they do not impact the applicative part, which remains costly to modify (recompilation of applicative constituents may be required, which implies some intensive testing and validation). On the opposite, modifying the MCDM implies recompiling only constituents which uses metadata, meaning only those directly implied in the experiment.

The MCDM structure is very small and is shown in Figure 6.10.

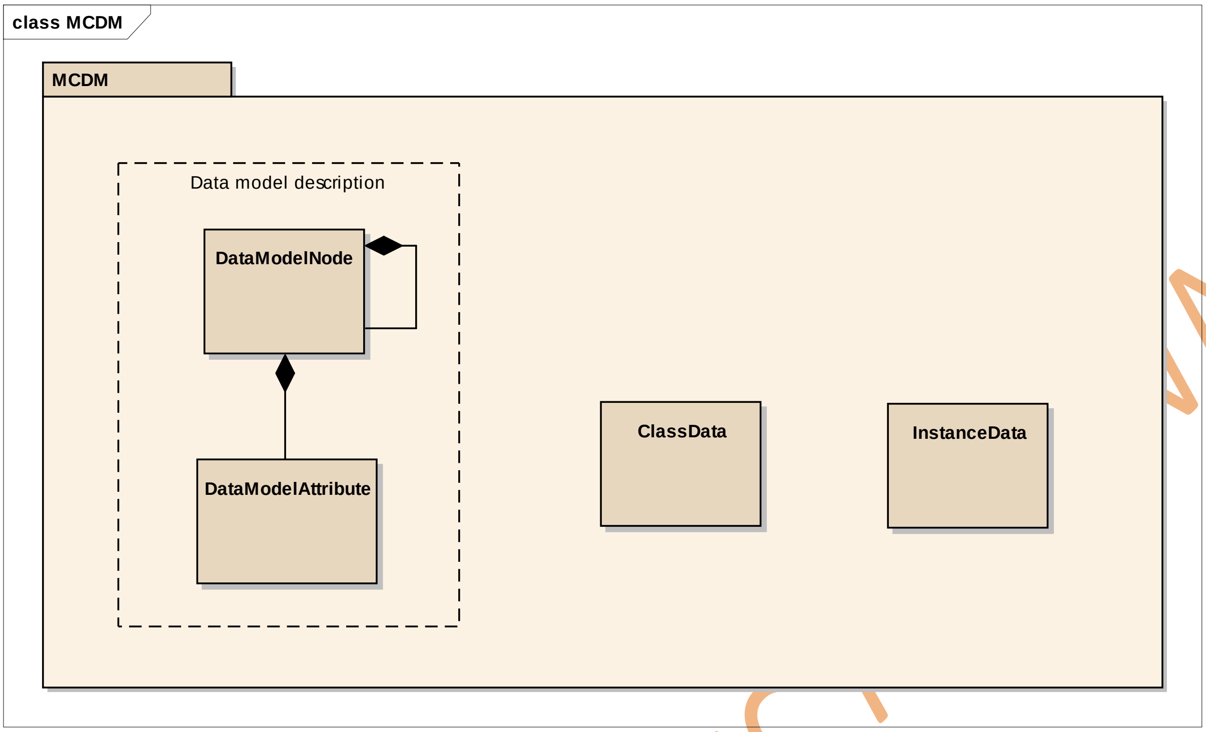


Figure 6.10: MCDM: data model for Metadata

The MCDM is composed of 4 classes. Those in Data model description box are used to describe the CDM (that is to provide a description of the CDM which can be used during runtime).

The *ClassData* class contains data on class types. For instance, it can be extended to contain the number of nodes of each type.

The *InstanceData* class contains data on nodes of the CDM. For instance, it could be extended to contain the last update time of each node of the CDM, if this data where not already made available by DDS.

As the MCDM is a specific CDM it has its own DDS instance (which can be imagined as a specific root, similar to "network" in the CDM). In this DDS instance, three nodes (*DataModelDescription*, *ClassData* and *InstanceData*) is the start address of and the rest of the address correspond to a node in CDM.

Example: Address

`/Metadata/ClassData/Network/infrastructure/functionalInfrastructure/signals/signal[12]`

refers to data common to all instances of Signal. Same address without index ([12] in the example) would refer to the "array of Signal" and not to the Signal class.

Similarly,

`/Metadata/InstanceData/Network/infrastructure/functionalInfrastructure/signals/signal[12]`

refers to metadata on the specific instance on one Signal.

Obviously, the MCDM on Figure 6.10 does not contain useful data. To enable any useful application, the MCDM should be enriched by adding some attributes to existing classes, but this enrichment is specific to a project, network or experiment.

6.5.4. Conclusion

The above descriptions, covering the three aspects of metadata generation, metadata storage and metadata access addresses all use cases of metadata usage which have been foreseen in the usage section. It is also compatible with any DDS technology chosen, with possible performance limitations depending on combination of technological choice with metadata generation need.

Moreover, this approach has the advantage of being completely separated from the applicative data. A complete TMS can be deployed using AF/IL frameworks, but with no metadata implementation at first, and then metadata implementation can be added at a later stage, with only little or even no impact on the existing code (except maybe the IL-Wrapper layer).

6.6. Data Interchange Formats

The current chapter provides description with the comparison of several examples of data interchange formats. Below presented formats are seen as good candidates to be used to communicate with the integration layer and for which the data model (CDM) can be easily mapped.

6.6.1. XML

XML [XML] is one of the most widely-used formats for sharing structured information between programs, peoples (it is human readable), both locally and across networks.

The Extensible Markup Language (XML) is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more. It was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use.

Important features of XML are:

- XML is a markup language much like HTML;
- XML was designed to store and transport data;
- XML was designed to be self-descriptive.

An example note stored as XML:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
</note>
```

Figure 6.11: Example of XML message

The “extensible” means that it only specifies the structural rules of tags. It does not specify tags on them self.

Sharing information between different kind of systems, applications, organizations in XML format is mainly possible because of simple syntax rules that it uses. XML data is stored in plain text format. Different incompatible systems allow to share data without needing to pass them through many layers of conversion. If applications can share data (through HTTP, file sharing, or another mechanism), and have an XML parser, they can share structured information that is easily processed.

It supports Unicode, allowing almost any information in any written human language to be communicated. Its self-documenting format describes structure and field names as well as specific values.

The strict syntax and parsing requirements make the necessary parsing algorithms very simple, efficient, and consistent.

The important feature of XML is use of schema languages (XML Schema - XSD, DTD) which allows effective document validation, also makes contractual specification and software construction easier.

Additional support for namespaces allows for sharing of standard structures.

The data stored in XML can easily be transformed to other formats using XSLT transformations, support for querying (XQuery, XPath) also is provided.

Besides many advantages and popularity of XML still can think about some week points of it

- XML does sometimes cause a significant increase in data size and processing time - robustness of redundant labels in start and end tags, Unicode support increase the amount of space XML requires in disk storage or the amount of bandwidth for moving it over a network;
- possible eternal references / links extend parsing times leading to some serious performance problems.

Link: http://www.w3.org/XML/

https://www.w3schools.com/xml/xml_whatis.asp

6.6.2. JSON

JSON stands for JavaScript Object Notation [JSON]. It is text-based, lightweight, data-interchange format based on a subset of the JavaScript programming language. It is language independent format but uses conventions that are familiar to programmers of the C-family of languages.

JSON defines a small set of structuring rules for the portable representation of structured data.

In many cases JSON format is similar to XML Some important similarities are:

- text based, data-interchange format, easily to parse;
- language independent;
- self-describing;
- support for hierarchical values;
- support for schema validation.

An example note stored as JSON:

```
{
 "note": {
   "to": "Tove",
   "from": "Jani",
   "heading": "Reminder",
   "body": "Don't forget me this weekend!"
 }
}
```

Figure 6.12: Example of JSON message

There are important differences that distinguish JSON from XML:

- JSON has a much smaller grammar and maps more directly onto the data structures;
- JSON is shorter that XML (do not have end tag) – smaller message size;
- JSON is easier to handle with Javascript;
- JSON describe structural data also with arrays.

however:

- JSON does not provide support for namespaces;

- lack of parsing standards like DOM;
- lack of standards for querying like XQuery and XPath;
- lack of standards for transforming a document like XSLT.

Links: http://www.json.org/

https://www.w3schools.com/js/js_json_intro.asp

6.6.3. Protobuf (Protocol Buffers)

Protocol Buffers [Protocol Buffers] is a method of serializing structured data. It is useful in developing programs to communicate with each other over a wire or for storing data. The method involves an interface description language that describes the structure of some data and a program that generates source code from that description for generating or parsing a stream of bytes that represents the structured data.

Important features of Protocol Buffers:

- small and fast data interchange format developed as open source project by Google;
- Messages are serialized into a binary wire format which is compact, forward- and backward-compatible;
- An alternative to data-centric C++ classes and structs, especially where interoperability with other languages or systems might be needed in the future.

Protocol Buffers is a typed protocol, so work with it starts with some kind of Interface Definition Language named 'proto'. Running the protocol buffer compiler on a .proto file generates the code in chosen language with the message types described in the file, including getting and setting field values, serializing your messages to an output stream, and parsing your messages from an input stream.

Currently protocol buffer compiler supports languages: C++, Java, Python, Go, C#, Objective C, and Javascript. According to Google official site support for several more languages are in the pipeline.

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

Figure 6.12. Example of Protocol Buffers schema (.proto)

The protocol buffer message is a series of key-value pairs. The binary version of a message just uses the field's number as the key – the name and declared type for each field can only be determined on the decoding end by referencing the message type's definition (i.e. the .proto file).

When a message is encoded, the keys and values are concatenated into a byte stream. When the message is being decoded, the parser needs to be able to skip fields that it doesn't recognize. This way, new fields can be added to a message without breaking old programs that do not know about them.

Currently the version 3 of Protocol Buffers is available. It introduces several improvements over version 2 like “reserved” keyword to control reserved field names and field numbers, removing “optional” keyword – all fields are optional by default or support for new languages (Objective-C and C#).

Version 3 supports a canonical encoding in JSON, making it easier to share data between systems.

According to official Google information it is said that protocol buffers comparing to XML are general 3 to 10 times smaller, 20 to 100 times faster, generate data access classes that are easier to use programmatically [Protocol Buffers].

Link: https://developers.google.com/protocol-buffers/

6.6.4. Data interchange formats summary

This section summarises information about above mentioned data formats. Although on the market exists much more different formats the comparison and analysis focus on the most common, broadly used ones and do not consider less popular and more specialized formats.

The first – XML is human-readable and human-editable protocol which is also – to some extent – self-describing. It is a big advantage of one of the most popular data interchange format. As disadvantage XML introduces significant overhead connected with metadata details causing the message size to be the biggest of all compared protocols.

The second JSON provide much lower overhead on a message size comparing to the XML. It is a good choice for services directly consumed by a web browser.

If open text format is not the top priority, Protocol Buffers offer several compelling advantages over JSON and XML. Especially in terms of speed of encoding and decoding, size of the data on the wire.

In case of communication done between internal services these benefits can have significant importance to choose this data interchange format.

Serialization / deserialization based on a schema is also important aspect. In Protocol Buffers the object stream need to be serialized, de-serialized with the proper .proto file. Without knowing the schema is hard to or even not possible to decode a message. Data format internally is ambiguous and need schema to clarify. This has also another aspect – message is more secure from unintending reading than open text format. De-serializing based on a schema also helps to ensure the message completeness.

Forward and backword compatibility provided due to numbered fields in .proto definitions allows easily extend definitions and maintain compatibility. It obviates also the need for version checks which is a problem in JSON format.

The following chapter the selected data formats will be compared.

Inside of TMS there are two major data patterns:

- Real Time Traffic Plan containing structured data with low update rates (at average once per 10 second);
- field data containing a lot of small pieces of data with low update rate each, but with high total update rates 50-1000 updates per second TMS-wide.

To get an impression for trade-off between text based and binary encoding we selected XML and Protocol Buffers – based encodings for real data in a country wide TMS with 5000 km tracks and over 1000 train runs per day.

6.6.4.1. Use case “Snapshots of Real Time Traffic Plan”

The Real Time Traffic Plan contains a timetable, microscopic topology and engineering data for different views (track view and time-distance graph). The measurement were taken on Windows 7 64 bit, Microsoft 2012 C++ compiler, on Intel i5-2520M CPU@2.5 GHz notebook with 16 GB RAM using a single thread test application. In all cases DOM-based API (tree-based) was used. The numbers of XML were used as 100% and values for JSON and Protocol buffers are represented as %-value in reference to XML.

	XML	JSON	Protocol Buffers
Message size, bytes	24.039.242	58% (unformatted) 90% (formatted)	30% (6.137.415)
Compressed message size,	1.862.437	74%	65% (1.454.940)

LZ4, level=3			
Export time [ms], C++	479	90%?	20% (96)
Compression duration [ms]	154	112	44% (68)
Decompression duration [ms]	7,6	5,46	45% (3,4)
Message->memory representation, [ms]	47	?40	62
Memory representation ->custom model [ms]	218	100%	82% (180)

Table 6.1: Comparison JSON – XML – Protocol Buffers

Conclusions:

- the usage of Protocol Buffers for compressed snapshots will save ca. 35% of RAM and network bandwidth in comparison to XML;
- in a tree data set with relatively few numerical data and with many object references, IDs and human readable names represented by strings, the binary representation provides only modest benefit in size and speed.

6.6.4.2. Use case “Updated single trip”

In opposite to the use case “Snapshot of Real Time Traffic Plan” containing over 1000 trips in this use case only one changed trip is encoded. A trip consists of:

- a list of references to stations and waypoints it has visited;
- arrival and departure times.

For a trip with 18 station visits the following telegram sizes were produced.

	XML	JSON	Protocol Buffers
Message size	4611	2416	1114
Compressed message size, LZ4, level=3	1028	920	713
Export time [ms], C++	0,078	0,08	0,015
Compression duration [ms]	0,031	0,031	0,015
Decompression duration [ms]	0,0016	0,0016	0,0016
Message->memory representation, [ms]	0,01	0,01	0,014
Memory representation ->custom model [ms]	0,052	0,05?	0,028

Table 6.2: Comparison JSON – XML – Protocol Buffers for messaging attributes

Conclusions:

- in case of uncompressed messages the binary encoding would require 4 times less hardware (RAM in IMDG-Servers) and bandwidth in comparison to XML;

- in case of compressed messages XML representation of one trip takes 3,6 times longer on sender side, and 30% at receiver side;
- decompression of messages takes less than 2% of message handling time at receiver side.

6.6.5 Use case “Train position reports”

Depending on configuration a train running in ETCS mode produces position reports every ca. 6 seconds. Each position report contains field shown in Table 6.3.

Nr.	Variable	Content	Bits in ERTMS	Nearest number representation in C++
1	NID_MESSAGE	Message identifier	8	unsigned char (8 bits)
2	L_MESSAGE	Message length	10	short int (16 bits)
3	T_TRAIN	Trainborne clock	32	unsigned long (32 bits)
4	NID_PACKET		8	Unsigned char (8 bits)
5	L_PACKET		13	Short int (16 bits)
6	Q_SCALE		2	Unsigned char (8)
7	NID_LRBG		24	Unsigned long (32)
8	D_LRBG		15	Short int (16)
9	Q_DIRLRBG		2	Unsigned char (8)
10	Q_DLRBG		2	Unsigned char (8)
11	L_DOUBTOVER		15	Short int (16)
12	L_DOUBTUNDER		15	Short int (16)
13	Q_LENGTH		2	Unsingned char (8)
14	L_TRAININT		15	Short int (16)
16	V_TRAIN	Current speed in 5 km/h steps	7	Unsigned char (8)
17	Q_DIRTRAIN		2	Unsigned char (8)
18	M_MODE		4	Unsigned char (8)
19	NID_STM		8	Unsigned short (8)
	Total		23 bytes	30 bytes

Table 6.3: Comparison JSON – XML – Protocol Buffers for Train Position Reports

Message size and parsing times for one train position report are shown in the following table.

	XML	JSON	Protocol Buffers
Message size, bytes	272	214	41
Compressed message size, LZ4, level=3	241	202	56
Export time [μs], C++	20	19	0,53

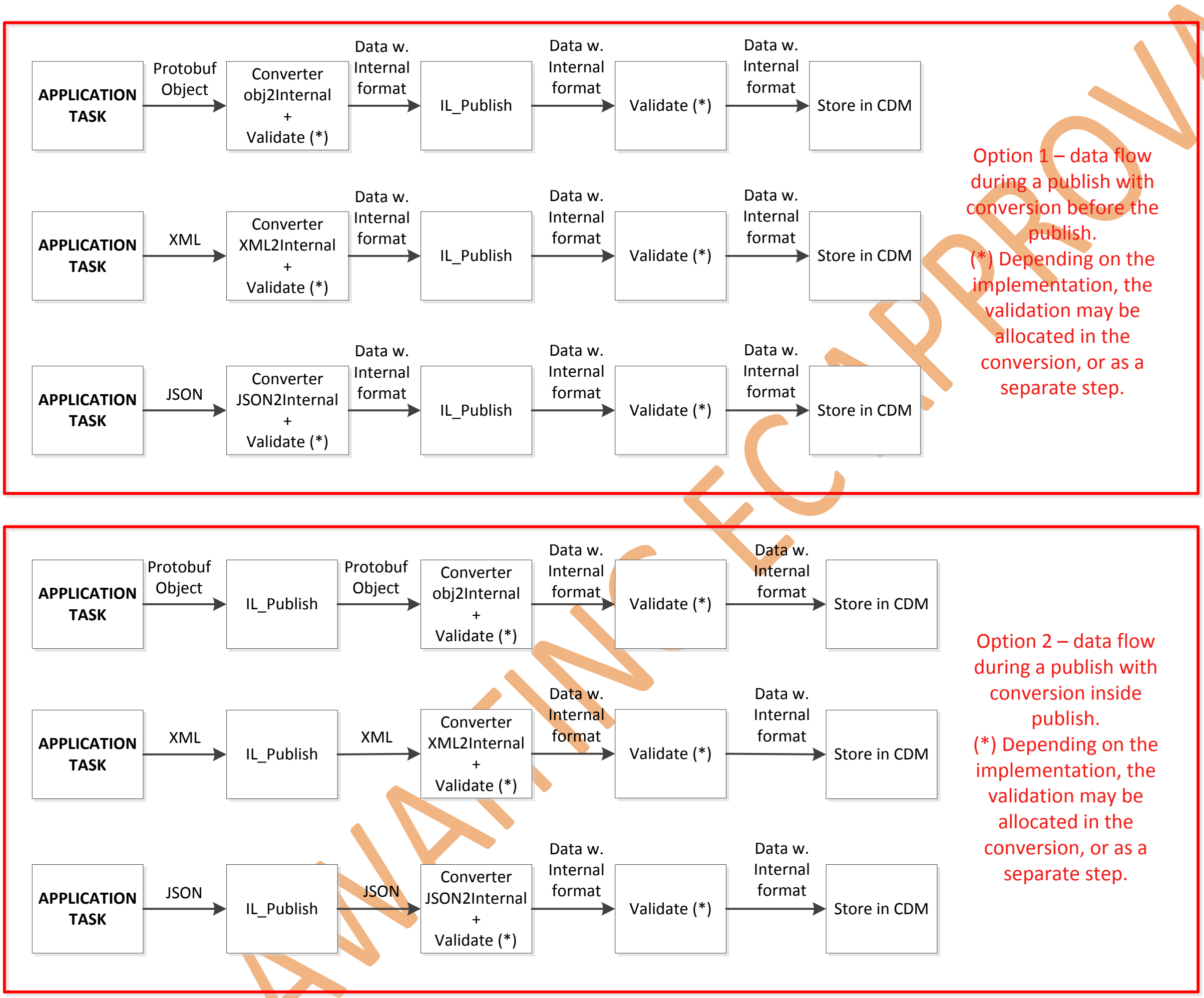
Table 6.4: Comparison JSON – XML – Protocol Buffers for message size of Train Position Report

Conclusion: for a message consisting of integers with different length the Protocol Buffers encoding is very efficient - 6 times less RAM and 40 times less parsing time in comparison to XML.

6.6.6. Message transformation

Access to the integration layer requires message conversion from the format specific for the client requesting the access to the internal raw one – internal for the IL.

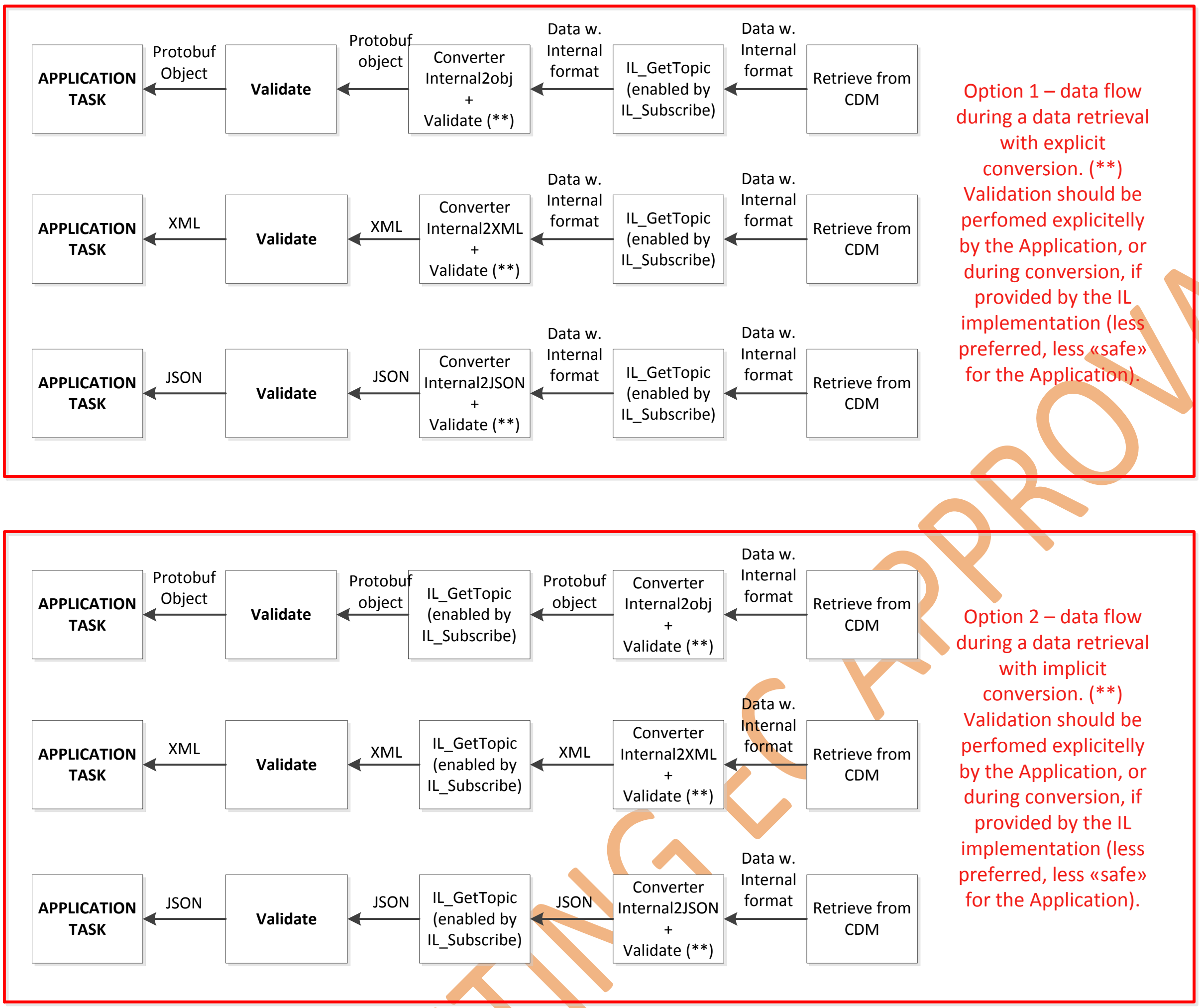
The possible conversion scenarios illustrate here data flows presented for discussed message formats.



Option 1 – data flow during a publish with conversion before the publish.
(*) Depending on the implementation, the validation may be allocated in the conversion, or as a separate step.

Option 2 – data flow during a publish with conversion inside publish.
(*) Depending on the implementation, the validation may be allocated in the conversion, or as a separate step.

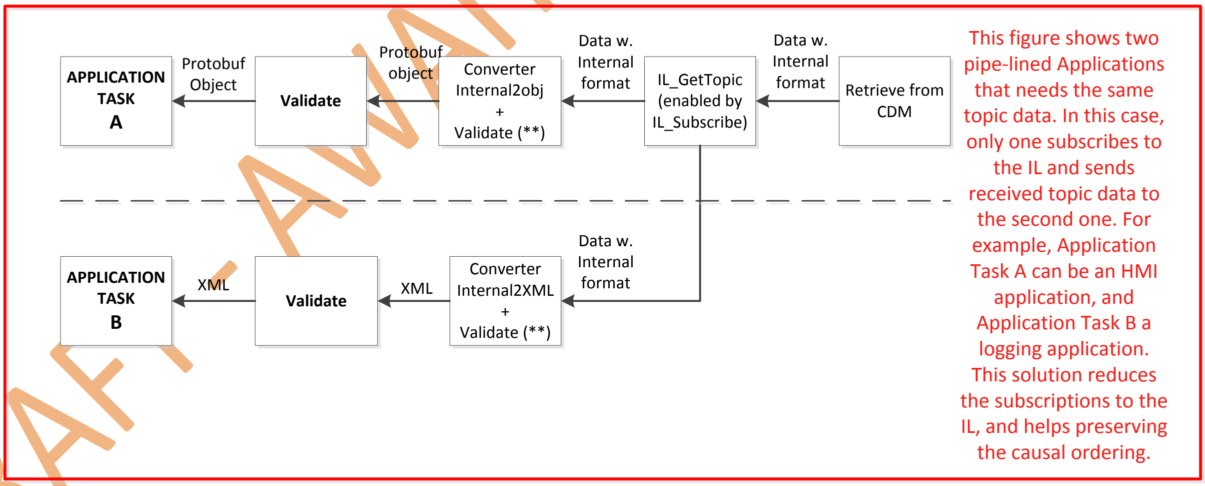
Figure 6.13: Possible data flows during the "publish" operation for the IL



Option 1 – data flow during a data retrieval with explicit conversion. (**) Validation should be perfomed explicitelly by the Application, or during conversion, if provided by the IL implementation (less preferred, less «safe» for the Application).

Option 2 – data flow during a data retrieval with implicit conversion. (**) Validation should be perfomed explicitelly by the Application, or during conversion, if provided by the IL implementation (less preferred, less «safe» for the Application).

Figure 6.14: Possible data flows during the "subscribe" operation from the IL



This figure shows two pipe-lined Applications that needs the same topic data. In this case, only one subscribes to the IL and sends received topic data to the second one. For example, Application Task A can be an HMI application, and Application Task B a logging application. This solution reduces the subscriptions to the IL, and helps preserving the causal ordering.

Figure 6.15: Example of data flow during "subscribe" operation for the same topic data

6.6.7. Conclusion

From the presented comparison of the selected data interchange formats the Protocol Buffers [Protocol Buffers] especially from the performance point of view is the best option. It could be used internally in the Integration Layer for efficient storage and transfer of data.

For the tightly coupled components within the Application Framework (i.e. the TMS system constituents), it makes sense to use this data format as well. Hence, it should be exposed through the IL API “as is”.

From another perspective Protocol Buffer can be easily transformed into JSON format if needed, so clients can see JSON message while internally is still Protocol Buffers.

For independent systems with a logically loose coupled to each other, as e.g. the Energy Management system when communicating to TMS or Asset Management, the features of XML may prove useful. The self-descriptive nature and strong schema semantics will simplify validation of messages, and the human readable form will aid in tracing message traffic. Also, communication on this level will be less intense, making the compactness of Protocol Buffers less interesting.

The final decision agreed in WP8 is to allow any format clients wants or needs to communicate with IL without indication of any default ones. (so-called “democratic” scenario). The used format has to be set at the start of the communication between client and IL.

The list of possible formats in such communication can easily be extended without risk of breaking backward compatibility in communication with any client.

Looking from perspective of integration layer any external, client format will be translated into internal raw representation for integration layer. In opposite direction, the translation will be done from raw format, so the response to the client will always be delivered in requested by it format.

The model will be extended during the Shift2Rail program, adding elements needed from different projects.

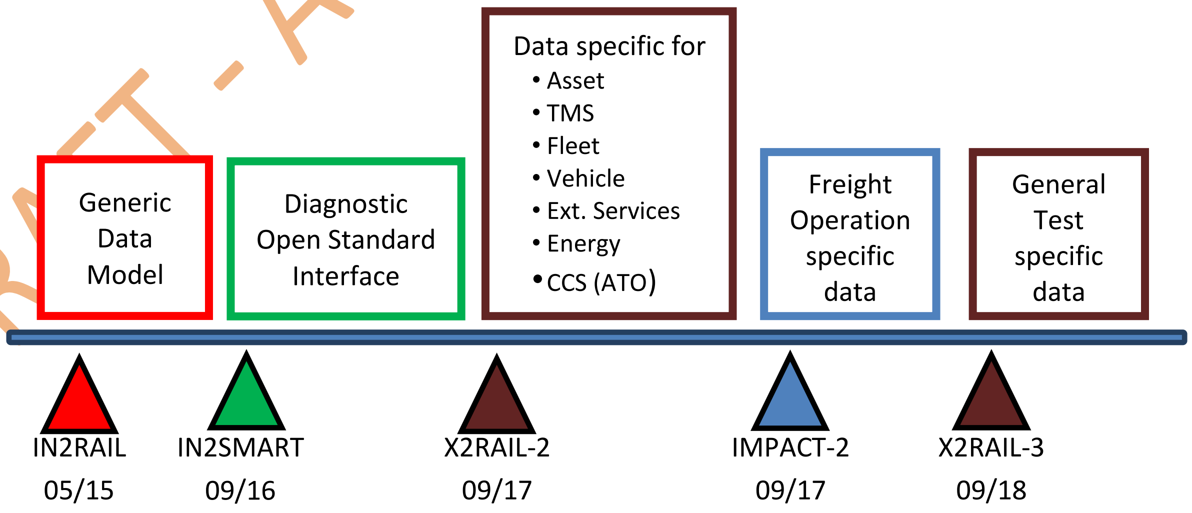


Figure 6.16: Preliminary CDM evolution roadmap

7. INTEGRATION LAYER CHARACTERISTICS

7.1. Architecture

The Integration Layer must provide high-availability and scalability by distributing data across multiple machines.

In-memory technology architectures take advantage of low-latency transaction processing. This is a consequence of the fact that the price of RAM is dropping significantly and rapidly and as a result, it has become economical to load the entire operational dataset into memory with performance improvements of over 1000x faster. In-Memory Compute and Data Grids provide the core capabilities of an in-memory architecture.

The figure below shows the architecture of integration Layer (IL) from a high-level perspective comprising several examples of services provided, surrounded by systems that will utilize this layer as well.

The basic idea for the architecture is triggered from ESB (Enterprise Service Bus) models. The IMDG implementation provides the coupling between integrated components via publish-subscribe information exchange pattern.

All the communication towards the IL is done via the unified API and uses one, common data model (CDM). For systems, components that are not supporting this standardized CDM, Interface Services are needed to provide function and Data mapping for proper communication.

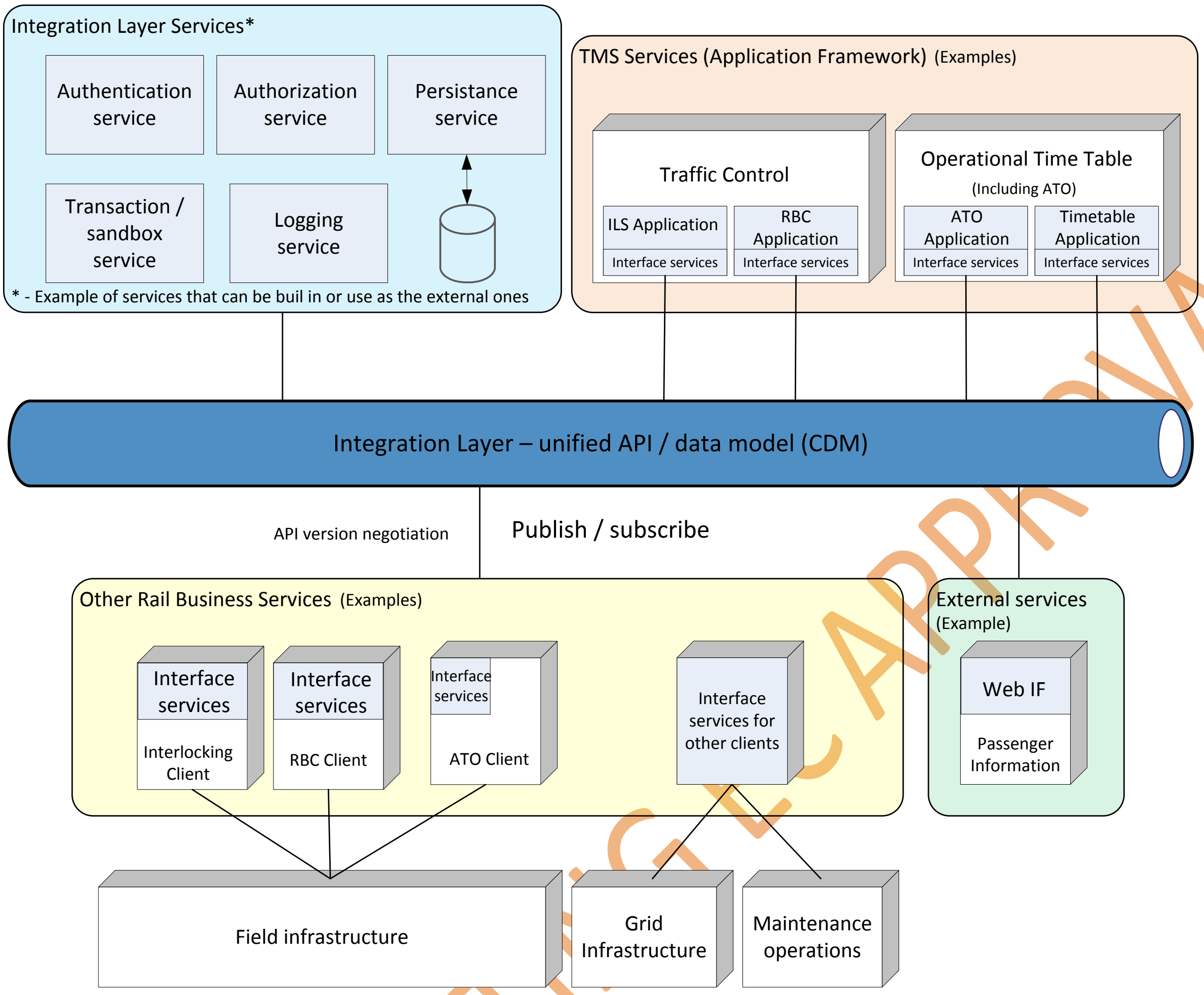


Figure 7.1: Architecture of integration layer - high level perspective

IL services

- Authentication service – to verify system/component/user (IL client) credentials that is willing to access IL (either read or write);
- Authorization service – to provide proper permissions for the IL client, to allow or reject requested operation;
- Persistence service – functionality offers possibility to persist information in a more durable form like database / files or other than IMDG;
- Transaction / sandbox service – to handle sequence of changes performed by the IL client, support for transactions;
- Logging service – to provide logging functionality.

The services listed above can have a form of built-in services offered by example by the IMDG platform itself or can even be external to the IL.

The other boxes represent systems, IL clients that communicate via IL in a loose coupling manner without having any strict, direct dependency to each other.

IL clients are grouped into following categories:

- TMS Services – which can be hosted, managed by the application framework (described in D8.6);
- Other Rail Business Services – not related directly to TMS, but other CCS systems;
- External Services – visualize systems that can benefit from information available on IL like passenger information.

Looking from another perspective, there is assumption to exist several Control Centres each with architecture like describing above. Each Control Center can synchronize, exchange information with another via the gateway and WAN (Wide Area Network). In this way the information that are not only specific to one particular region can be exchanged and available by other systems, components from different centres.

This idea is illustrated on below diagram.

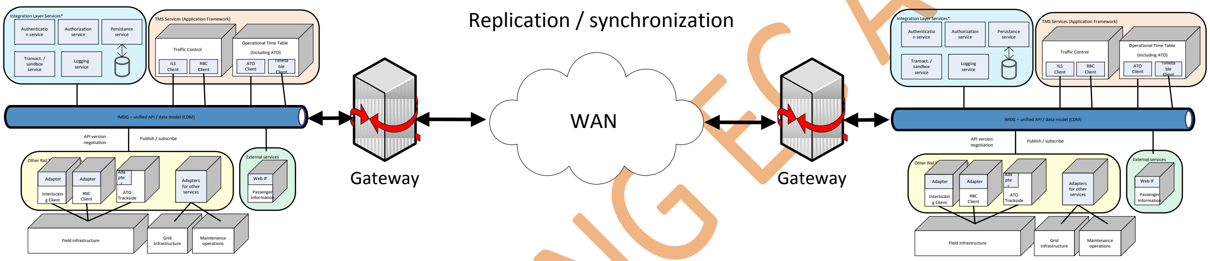


Figure 7.2: Control Centre replication, synchronization idea

7.2. Quality of Services

Quality of services represents the possibility for the application to provide the Integration Layer requirements on communication aspects. Depending on the implementation some Integration Layers can use the requirements to optimise its operation.

Requirement	Description
Time to deliver	Expected duration from writing a message until the first client has received it. Typical use case is batching at the writer side – it combines several messages into one telegram on the wire and increases the performance of IL in some case by factor 10.
Reliable communication	Defines if the message about modified key-value shall be delivered by a protocol with acknowledgement (e.g. TCP), or by a best effort approach (e.g. UDP based).
Validity interval	Identifies how long the data represented in the message is valid. After this time the IL can drop it and the clients can react in some proper way (e.g. raise a warning for missing update).
Transport priority	In case of conflicts due to limited bandwidth, the IL can decide which Topic (Message) is more important based on priority requirement.
Durability:	With this requirement the application case associate the

- Zero live time - Session limited live time - In memory - On disk	durability of the data: - Zero live time means the message is distributed to currently subscribed clients and then deleted - Session limited means the message is send to all current and future subscribers and will be deleted after closing the session. - In memory means the message will be provided to all current and future subscribers as long as sufficient nodes running IL are present. - On disk means the message survives a full restart of Integration Layer.

Table 7.1: Quality of Services (QoS)

The names for the variables defining requirements will be specified in D8.4.

7.3. Maintenance and Diagnostic

Maintenance and diagnostic activities are correlated and serve quite similar purpose – to ensure that system is able to work properly.

7.3.1. Maintenance

The software maintenance work starts once the delivery of the software product is done. Maintenance is considered as part of the software development life cycle and in general is a modification of software product after delivery to correct faults or to improve performance or other attributes.

Application maintenance can be broadly classified under the following categories:

- **Corrective maintenance**: reactive modification of a software product performed after delivery to correct discovered problems;
- **Adaptive maintenance:** modification of a software product performed after delivery to keep a software product usable in a changed or changing environment;
- **Perfective maintenance**: modification of a software product after delivery to improve performance or maintainability;
- **Preventive maintenance**: modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

The above classification comes from the standard ISO/IEC 14764.

There is also a notion of pre-delivery/pre-release maintenance which is all the good things you do to lower the total cost of ownership of the software. Things like compliance with coding standards that includes software maintainability goals. The management of coupling and cohesion of the software. The attainment of software supportability goals. [Software maintenance - Wikipedia]

Looking at example of Hazelcast implementation of IMDG – product considered as one of possible platforms to form IL (presented later in this document) it can be seen in its documentation how to proceed with maintenance activities. The documentation covers several aspects connected to its lifecycle, maintenance and updates.

It can be found recommendation regarding configuration management, cluster start-up, topology changes, software updates and others.

7.3.2. Diagnostic

In a production environment, it's important to be able to track the way in which users utilize your system, trace resource utilization, and generally monitor the health and performance of your system. The information can be used as a diagnostic aid to detect and correct issues, and also to help spot potential problems and prevent them from occurring.

7.3.2.1. Monitoring and diagnostics scenarios

Monitoring allows to gain an insight into how well a system is functioning. Monitoring is a crucial part of maintaining quality-of-service targets. Common scenarios for collecting monitoring data include:

- ensuring that the system remains healthy;
- tracking the availability of the system and its component elements;
- maintaining performance to ensure that the throughput of the system does not degrade unexpectedly as the volume of work increases;
- guaranteeing that the system meets any service-level agreements (SLAs) established with customers;
- protecting the privacy and security of the system, users, and their data;
- tracking the operations that are performed for auditing or regulatory purposes;
- monitoring the day-to-day usage of the system and spotting trends that might lead to problems if they're not addressed;
- tracking issues that occur, from initial report through to analysis of possible causes, rectification, consequent software updates, and deployment;
- tracing operations and debugging software releases. [Microsoft: Best Practices – Monitoring and diagnostics].

The following sections describe most common scenarios in more detail.

7.3.2.1.1. Health monitoring

A system is healthy if it is running and capable of processing requests. The purpose of health monitoring is to generate a snapshot of the current health of the system so that you can verify that all components of the system are functioning as expected.

An operator should be alerted quickly if any part of the system is deemed to be unhealthy. The operator should be able to ascertain which parts of the system are functioning normally, and which parts are experiencing problems.

7.3.2.1.2. Availability monitoring

A truly healthy system requires that the components and subsystems that compose the system are available. Availability monitoring is closely related to health monitoring. But whereas health monitoring provides an immediate view of the current health of the system, availability monitoring is concerned with tracking the availability of the system and its components to generate statistics about the uptime of the system.

In many systems, some components (such as a database) are configured with built-in redundancy to permit rapid failover in the event of a serious fault or loss of connectivity. Ideally, users should not be aware that such a failure has occurred. But from an availability monitoring perspective, it's necessary to gather as much information as possible about such failures to determine the cause and take corrective actions to prevent them from recurring.

7.3.2.1.3. Performance monitoring

As the system is placed under more and more stress (by increasing the volume of users), the size of the datasets that these users access grows and the possibility of failure of one or more components becomes more likely. Frequently, component failure is preceded by a decrease in performance. If it is possible to detect such a decrease, also it is possible to take proactive steps to remedy the situation.

System performance depends on a number of factors. Each factor is typically measured through key performance indicators (KPIs), such as the number of database transactions per second or the volume of network requests that are successfully serviced in a specified time frame.

7.3.2.1.4. Security monitoring

The complexity of the security mechanism is usually a function of the sensitivity of the data. In a system that requires users to be authenticated, it should be recorder:

- all sign-in attempts, whether they fail or succeed;
- all operations are performed, and the details of all resources accessed by an authenticated user;
- when a user ends a session and signs out.

Monitoring might be able to help detect attacks on the system. For example, a large number of failed sign-in attempts might indicate a brute-force attack. An unexpected surge in requests might be the result of a distributed denial-of-service (DDoS) attack. A system that has a sign-in vulnerability might accidentally expose resources to the outside world without requiring a user to sign in.

7.3.2.2. Sources of monitoring and diagnostic data

The information that the monitoring process uses can come from several sources. At the application level, information comes from trace logs incorporated into the code of the system.

All exceptions and warnings should be logged, also ensure to retain a full trace of any nested exceptions and warnings. Ideally, also information that identifies the user who is running the code should be captured, together with activity correlation information (to track requests as they pass through the system). Attempts to access all resources such as message queues, databases, files, and other dependent services should be logged. This information can be used for metering and auditing purposes.

Many applications make use of libraries and frameworks to perform common tasks such as accessing a data store or communicating over a network. These frameworks might be configurable to provide their own trace messages and raw diagnostic information, such as transaction rates and data transmission successes and failures.

The operating system where the application is running can be a source of low-level system-wide information, such as performance counters that indicate I/O rates, memory utilization, and CPU usage. Operating system errors (such as the failure to open a file correctly) might also be reported.

The following list presents best practices for capturing and storing logging information:

- the monitoring agent or data-collection service should run as an out-of-process service and should be simple to deploy;
- all output from the monitoring agent or data-collection service should be an agnostic format that's independent of the machine, operating system, or network protocol. For example, emit information in a self-describing format such as JSON, or Protobuf. Using a standard format enables the system to construct processing pipelines; components that read, transform, and send data in the agreed format can be easily integrated;
- the monitoring and data-collection process must be fail-safe and must not trigger any cascading error conditions;
- in the event of a transient failure in sending information to a data sink, the monitoring agent or data-collection service should be prepared to reorder telemetry data so that the newest information is sent first. (The monitoring agent/data-collection service might elect to drop the older data, or save it locally and transmit it later to catch up, at its own discretion).

7.4. Security

Security is a broad subject that encompasses many aspects, ranging from physical security of Information Systems to the security of information held on them. Information Security is about the prevention of unauthorized access, use, modification, disruption, duplication, inspection of information; it is based on the following key concepts:

- **Confidentiality:** information are not made available or disclosed to unauthorized individuals, entities, or processes;
- **Integrity:** data and information data cannot be modified in an unauthorized or undetected manner; accuracy and completeness of the data is guaranteed for their entire lifecycle;
- **Availability:** information is always available when needed; this implies that all of the entities involved in storing, processing, communicating and protecting the information shall function correctly and shall implement measures to prevent and contrast attacks.

Security standards such as ISO/IEC 27000 and IEC 62443 families, provide guidance and techniques that attempt to protect cyber environments, with the aim to evaluate and reduce the risks of cyber-attacks, and to prevent and mitigate them.

7.4.1. Security measures in the Integration Layer

The Integration Layer is embedded in a complex system. The following measures are needed to guarantee the correctness of its operations, and the overall security of its host system.

7.4.1.1. Access Control

The Integration Layer protect topics (see §6.4.4 Topics) and other resources it manages against unauthorized access.

Access Control encompasses:

- **Authentication:** authentication is the act of verifying the identity of an end-user of the Integration Layer. End-users are application running in the context of the Application Framework and, by extension, human users interacting with the applications. Authentication is needed each time an end user starts using the Integration Layer;
- **Authorization:** authorization is the mechanism that grants an authenticated end-user the rights to operate on resources controlled by the Integration Layer. Grants are defined by means of access permissions (e.g. full control, read only, delete) per resource (eg. Topic or CDM data set node). Access permission are given on end-user and/or node (e.g. workstation) basis.

7.4.1.2. Data Encryption

The Integration Layer shall support mechanisms to encrypt its managed data resources, during transmission (topic publish and subscribe) or storage (branches of the CDM). It is expected that Integration Layer does not implement its own mechanisms for encryption, instead it should take advantage of the most technologically updated solution.

7.4.1.3. Support for Incident Management and Intrusion Detection System

The Integration Layer supports Incident Management through the production of metrics and alerts that can be used by a SIEM (Security Information and Event Management) software product, which provide real-time analysis of security alerts generated by applications and network hardware.

The Integration Layer can provide communication metrics to an IDS (Intrusion Detection System); these metrics, combined with metrics provided by the platform hosting the middleware part of the IL, can be used by the external IDS to check unexpected changes on configuration and executables.

It is worth to note that the design and the implementation of the Integration Layer shall be done to take advantage of security products and technologies available on the market. The field of security is in continuous evolution and deals with ever changing threads and risks; it is impossible to face this phenomenon within the framework of the development of the Integration Layer.

7.4.2. Security facilities provided by the Integration Layer

In addition to the measures described in the previous section, the Integration Layer provides some hig- level support methods that obey security checks and controls.

Recall that the Integration Layer run in the context of the Application Framework, that end-users are defined either as applications running in the context of the Application Framework or human users interacting with the aforementioned applications.

The Integration Layer provides the end-user with sessions. Session are sequence of related operation on the CDM data set that can be started after the end-user pass its Integration Layer authentication phase. The end-user is free to start more than one session after the authentication.

During a session, the following methods are available to the end-user: Reservation, Sandbox, HandOver and TakeOver.

7.4.2.1. Reservation

A reservation sets:

- the list of access permissions of the owner (i.e. the end-user) of the reservation over the selected CDM Data Set node(s);

- the list of access permission the owner (i.e. the end-user) of the reservation grants to another end-user.

The reservation method will reserve the requested list of nodes of the CDM data set, provided that the end-user is authorized to reserve them.

More than one Reservation may be performed in the context of the same session.

Reservation can be freed when they are no more useful.

7.4.2.2. Sandbox

The sandbox creation method will check the existing reservations, the permissions to add a sandbox, and eventually add a sandbox node to the CDM data set.

More than one Sandbox creation may be performed in the context of the same session.

7.4.2.3. HandOver

The HandOver is a method that operates on existing sessions, reservations, or a sandbox:

- on session, it prepares the transfer of all the session reservations to another specified session – see TakeOver;
- on reservations, it prepares the transfer of reservations to another specified session – see TakeOver;
- on Sandboxes, it prepares the transfer of sandboxes to another specified session – see TakeOver.

7.4.2.4. TakeOver

TakeOver is complimentary to HandOver. The TakeOver method is provided to acquire all the reservations of another session, single reservations or sandboxes belonging to other sessions. Takeover shall be executed in a specified time since the handover.

7.4.3. Accounting

While not strictly related to Security, accounting services can be offered from the Integration Layer. If provided, access to resource managed by the Integration Layer is authorized, and accounting is performed based on one of the following profile:

- pay per use - the accounting is based on a per unit number/type usage;
- pay per licence – the accounting is based on an agreed fixed fare, dependent upon start date and duration;
- pay per licence – the accounting is based on decreasing the number of pre-paid units during their consumption.

7.5. Safety

An important aspect of system safety is failure handling. This determine if failure can be predicted based on different received notification or other observations or if happen haw fast can be eliminated.

Failures are not desirable but happens and it is a good practice to be prepared for it. Some basic approaches of handling failures are:

1. detecting system mis-functionality and trying to recover from the failure (e.g. by repairing or replacing the failed parts);
2. switching to backup solution (e.g. to some simpler one) and trying to repair or to replace the original one. When repaired or replaced solution is available, switching back;
3. having each subsystem in multiple copies and spread the load to the working ones. Replace/repair the broken copy and rebalance the system load afterwards;
4. accepting the failure and working with the rest of the system if possible;
5. when the reason of the failure ends, try to repair the system (or replace it by a new one) and try to renew the work.

Each of the approaches fits best in different conditions. Within next subchapters it will be discussed in more detail.

7.5.1. Failure handling – approaches

7.5.1.1. Stopping the work and waiting for repair

When something gets broken the process can be stop and wait until the resource is available again. This approach supports predictability (the things happen in the planned way or do not happen at all). Its key disadvantage is that the failed service is not available at all. This is applicable to highly reliable systems (the ones that get broken only very rarely) or when the service unavailability does not harm the user or its clients too much.

7.5.1.2. Switching to Backup Procedure

Such approach requires that the backup solution(s) is/are available. Failures must be expected already in the analysis and design phases and the backup solutions must be prepared. The main advantage is that the service is available with higher probability (although sometimes with limited extent or capacity). The main disadvantage is that the backup solutions must be designed, created, and the staff must be trained in the use not only of the main procedure but also in the use of the backup ones. The solution is recommended for critical/crucial services (critical infrastructure). This approach can get expensive (the backup solution can raise the expenses). But it is not necessary – especially if the backup solutions are restrictions of the previous (or the original) ones.

7.5.1.3. Rebalancing the Load to Peers

When the service is provided by multiple service providers, in the case of service provider failure it is possible to rebalance the load of the system to other service providers (the ones providing given service, of course). It is possible only when such other service providers are available, if they have load reserve and if they have everything that is necessary to provide the work of the failed service (in some cases they need access to some sensors, to some data or other special resources – such requirement can block smooth reassigning the task to other service provider). In some case such solution can overcome the service provider failure without any harm to the user or its clients.

This approach is well applicable when the load is high enough to be balanced between multiple service providers. Presence of multiple service providers as a rule means complex and expensive solutions.

7.5.1.4. Accepting the Failure

Sometimes the failure cannot be simply and immediately overcome (e.g. the ones caused by natural disasters). First, the reason of the failure must end, then the corrections can take place. During such failure the system simply cannot used the directly hit parts and its service can be e.g. geographically restricted. On the other hand, it is advantageous if the system is built so that failure of a part of it (even big one) cannot block (probably limited) functionality of the rest of the system.

7.5.2. Failure types and detection

Failure handling is related with various activities.

At least two basic service failures can be distinguished:

1. service is down;
2. service behaves incorrectly.

7.5.2.1. Incorrect behavior detection

Incorrect behavior detection requires either recognition of correct behavior or identification of some of known incorrect behavior patterns.

7.5.2.2. Correct behavior checking

Correct behavior of a service or actor can be based on analysis of its communication: it is often known how its correct inputs should look like and how it should react on such inputs. We can therefore analyze language of the outputs, timing of the responses, and correspondence of the outputs to the inputs. If some of these factors do not match correct behavior, we can denote the event as an example of incorrect behavior.

In some case it is required that the tested part must behave always correctly, in other cases some incorrectness is tolerated.

7.5.2.3. Detection system is down

When a (sub)system is down or frozen, it does not respond at all. Then there is a need to detect (and later also handle) these situations. In both cases no output appears although there are inputs to them the checked entity should react to.

One possible method is to check if given process is listed in the system. If not, it is down and must be respawn.

Other method is to send it some testing events/methods and wait for response. If it does not respond, handling is necessary. This method is based on expectation that if the system responds to the testing request, it responds also to real requests. It likely only if the same process/thread is used for both purposes. Otherwise it is possible that the testing resource is running but the real one is not. The described testing itself consumes part of the system throughput (small but still some).

When monitoring of outputs is possible and not consuming too much throughput, it is possible to test if the system communicates with partners. It is applicable when the system is communicating frequently enough.

7.5.3. Determining safety level

Safety Integrity Level (SIL) is an indicator of the relative risk-reduction provided by a safety function. The SIL notion results directly from the IEC 61508 standard which is not railway specific. For the rail industry, CENELEC has developed the EN 50126, EN 50128 and EN 50129 standards which were derived from the IEC 61508 to meet railway specific requirements.

The development of Safety Integrity Levels fulfils the need for more approachable and systematic safety management. The SIL probabilistic approach helps to determine the risks associated with a safety function, system or component and to set an acceptable risk level for it.

Four Safety Integrity Levels are defined, ranging from SIL-1 to SIL-4:

	Probability of failure per hour (PFH)	**Risk reduction factor (RRF)**
SIL-1	10^{-5} -10^{-6}	10^{5} -10^{6}
SIL-2	10^{-6} -10^{-7}	10^{6} -10^{7}
SIL-3	10^{-7} -10^{-8}	10^{7} -10^{8}
SIL-4	10^{-8} -10^{-9}	10^{8} -10^{9}

Table 7.2: Safety Integrity levels

As described in the table above, SIL-4 provides the highest risk reduction factor and is reserved for highly critical safety functions that may cause important casualties and must be prevented at all costs. [EKE]

7.5.4. Communication between two safety relevant system through open channel– e.g. using Integration Layer

According to EN 50159-2, communication between safety relevant systems through an open communication channel must be individually evaluated, considering the safety requirements for given pair of safety relevant systems, including the specific requirements for communication functions. They are dependent on properties of considered open channel.

The evaluation of a communication function between two safety relevant entities must undergo the risk identification and overall safety evaluation process. As a result, the necessary safety-relevant measures must be proposed and evaluated, considering the properties of open communication channel (e.g. bit error rate).

As a practical example, communication between safety-relevant entities include safety-relevant functions that allow detection with a given probability any inconsistency in the communication process. The measures may include consistency checking of individual messages (e. g. detection of errors in messages via CRC) as well as any inconsistency in the communication. Typically, safety relevant systems should be able to safely identify the data source (safe authentication process), to detect any incorrect ordering of messages, possible wrong message inclusion /exclusion, and ensuring the timeliness for time critical operations, etc.).

To allow using Integration Layer as a general open communication channel for safety relevant applications, the following aspects should be further considered and analysed:

Considering that Integrity layer have a lower safety integrity level than communicating safety relevant entities, IL must enable at least the following:

- to establish a communication-based safe authentication process between safety relevant entities, independently on any function of IL;
- allowing to transfer any additional data generated by safety-relevant entities that will allow them to implement all the necessary safety-relevant functions required for ensuring safe communication (e.g. massage hashes, CRC codes, numbering, etc.), independently on any function of IL;
- IL should allow such a mode of operation for communicating safety-relevant entities that will make possible to make reasonable assumptions on the properties of open communication channel (e.g. limit the number of data transformations, establishing the dedicated channel, etc.);
- if required, IL should allow evaluation of timeliness of the transferred data between communicating safety relevant entities to support e.g. time synchronization of the safety-relevant systems, independently on any function of IL.

As consequence, for specific, safety-relevant functions and properties, the IL should be able to provide a direct communication channel with given properties between safety relevant entities.

7.6. Synchronization and Transactions

In systems build according to the layers pattern bottommost layer usually is assigned to simple data. Granularity of the data is fine, so each datum can be managed individually. It is desired from technical point of view. Database engine (or another repository management software) has no domain logic, so it doesn't know anything about data semantics and meaning of logically relationships between them.

From domain point of view such granularity would make data maintenance process complicated. Elementary piece of data to be managed on this level has to reflect concepts from problem domain world. Data are only metaphor of real world, i.e. data are model of reality.

From the perspective of the application, data play supporting role in operations/ services/requests/messages. Each operation application performed manipulates data. It reads, updates, deletes and creates. Subject of these operations are a compound of data – not single data itself. The operation is one, atomic unit of work and must be successfully completely performed.

Such a unit of work is called transaction. On data level transaction means "all or nothing". Either all data changes are made or no one of them. Transactions have four important properties, so-called ACID:

- **Atomicity:** all or nothing. If one part of transaction fails whole transaction fails and all changes are revoked;
- **Consistency:** every transaction moves database from one valid state to another. Valid state means the all invariants are fulfilled;
- **Isolation:** all concurrent transaction are performed as if they were executed sequentially (see below);
- **Durability:** all data changed within transaction are permanently stored one transaction has been committed, even the database crashes immediately thereafter.

7.6.1. Distributed transactions

Complex system consists of many subsystems, components and other entities. Components are independent each other. Each component can be deployed on different machine – virtual or real one. This is co called distributed architecture. The components can share the same database or can use individual one. In both cases some operations performed by several components can be part of one transaction from client point of view. Such transaction can cover operations on many databases or other repositories. This type of unit of work we name “distributed transaction”

7.6.2. Transaction isolation levels

Transaction isolations define how two or more transactions can interact each other. There are four type of isolation levels:

- **read uncommitted:** data created/updated within one transaction can be read from within another transaction before original transaction has been committed. It actually means there is no isolation at all;
- **read committed:** data created/updated by particular transaction can be read from within another one only after original transaction has been committed;
- **repeatable read**: this kind of isolation ensures the other transactions read the same set of data every read operation. The transaction which updates the data rows holds a write lock on entire set of data. Transaction which only read data holds a read lock. Every locks are mutually exclusive, so write operation must wait for finish reading set of data it wants to modify. And versa – reading operation must wait until writing one unlocks data set;
- **serializable:** like Repeatable read, but locks are holds on the range of rows they affect.

7.6.3. Long lived transactions

Transactions usually span relatively short period of time. There are also needs to have transaction-like unit of work. That unit of works are so-called long-lived transactions. The long-lived transactions can span few days or even weeks.

Such long duration of transactions requires using special mechanisms, which differs from ones used with normal databases’ transactions. Main difference is that we can switch database engine off during long-live transaction and it must not violate transaction durability. In the other words modifications made during long lived transaction have to be stored in the repository, but they fulfill all ACID properties. In particular the Isolation, which means in long-lived transaction context the modifications still shouldn’t be visible in other transactions.

7.6.4. Versioning

As said above modification are stored during long-lived transactions. In order to be hidden from other transactions the data stored in the repository must be versioned. Each datum has to have unique version mark. Subsequent marks should be monotonic, in order to determine precedence in each pair of marks. As a mark timestamp should be used, but in case of distributed system marks generator must base on the same time (e. g. UTC) or consult time zones.

Inside long-lived transaction newest versions of data are used. Other transactions use the oldest one. Approve long-lived transaction removes all historical versions of data, so there remains only newest one. In subsequent operations newest and oldest versions are the same, so all transactions see the same data.

7.6.5. Conclusion

Nevertheless, of technology and architecture patterns the integrated railway system will use transactions. The Integration Layer (IL) operations (data manipulations, triggering events, send/receive messages, service requests, etc.) always are surrounded by transaction, but IL never controls transactions. Transactions are started/finished on application level. Further analyzes are needed to establish which application needs additional transactions support.

Distributed transactions will be used, as the integrated system will be distributed too. Most imported issue is decoupling transaction´s split by message/event. Message service can send other messages, which start other services and so on. Usually all that service shouldn’t be under only transaction. Messages or publisher/subscriber participants should work independently (i.e. in asynchronous mode).

To support transactions (including distributed ones) verification of middleware should be done to ensure it support them too. It is also required that API must support transactions.

Long-lived transactions probably will be used very seldom. The only places they can be applied is time table manipulation by operators according specific regulations. These actions may be time consuming and may require test operation.

Further transaction concepts will be analyzed within S2R project.

8. Available technologies

Within this chapter middleware technologies / platforms are described which are seen as good candidates to implement the functionality of the IL.

The message centric technology is also presented even the design recommendation is for the data centric principle.

In the succeeding S2R projects, various partners will analyse the different technologies based on real implementation and structured test methodology.

8.1. DDS

The Data Distribution Service (DDS) [DDS] specification standardizes the software application programming interface (API) by which a distributed application can use “Data-Centric Publish-Subscribe” (DCPS) as a communication mechanism.

DDS is the open standard for messaging of the Object Management Group (OMG) that supports the unique needs of both enterprise and real-time systems.

8.1.1. Key features of DDS

- **portability:** DDS was designed from the start to support any programming language. It is the only standard messaging API for C and C++; it also supports Java, C#, Ada, JMS, WSDL/SOAP and REST/HTTP interfaces;
- **wire interoperability:** the DDS Real-Time Publish-Subscribe (RTPS) wire protocol provides seamless interoperability across implementations, platforms and programming languages;
- **semantic interoperability:** applications communicate by exchanging discoverable data objects typically described by a standard data model. The data model can be specified using the OMG Interface Definition Language (IDL), XSD/WSDL, XML or a programmatic interface. It can also be generated from a Unified Modeling Language (UML) model;
- **many implementations:** at least 10 unique middleware implementations support the DDS API or wire protocol.

8.1.2. Advanced Integration Capabilities

DDS standardizes powerful messaging semantics that reduce development and integration costs while improving system scalability and robustness.

- **Data-centricity:** data-centric publish/subscribe messaging provides loose coupling, reduces application and integration logic, and scales to systems of systems;

- **performance:** DDS supports the full range of performance requirements from enterprise to high-performance real-time systems;
- **QoS:** DDS defines a comprehensive set of Quality of Service (QoS) policies. These provide control over dynamic discovery, content-aware routing and filtering, fault tolerance and deterministic real-time behavior.

Data-centric communication provides the ability to specify various parameters like the rate of publication, rate of subscription, how long the data is valid, and many others. These Quality of Service (QoS) parameters allow system designers to construct a distributed application based on the requirements for, and availability of, each specific piece of data.

A data-centric environment allows to have a communication mechanism that is custom-tailored to particular distributed application's specific requirements.

By employing a "publish-subscribe" methodology for data communications, DDS provides abstract communications between data senders and receivers. Publishers of data are not required to know about each individual receiver, they only need to know about the specific data type that is being communicated. The same is true for subscribers. Subscribers do not need to know where the published data is coming from; they only need to know about the specific data type they wish to receive.

The specification for DDS is broken up into two distinct sections. The first section covers Data-Centric Publish-Subscribe (DCPS) and the second section covers the Data Local Reconstruction Layer (DLRL).

DCPS is the lower layer API that an application can use to communicate with other DDS-enabled applications. DLRL is the upper layer part of the specification that outlines how an application can interface with DCPS data fields through their own object-oriented programming classes. DLRL is an optional layer within the DDS specification.

DCPS is comprised of the following primary entities:

- Domain;
- Domain Participant;
- Data Writer;
- Publisher;
- Data Reader;
- Subscriber;
- Topic.

8.1.3. Domains and Domain Participants

The domain is the basic construct used to bind individual applications together for communication. A distributed application can elect to use a single domain for all its data-centric communications.

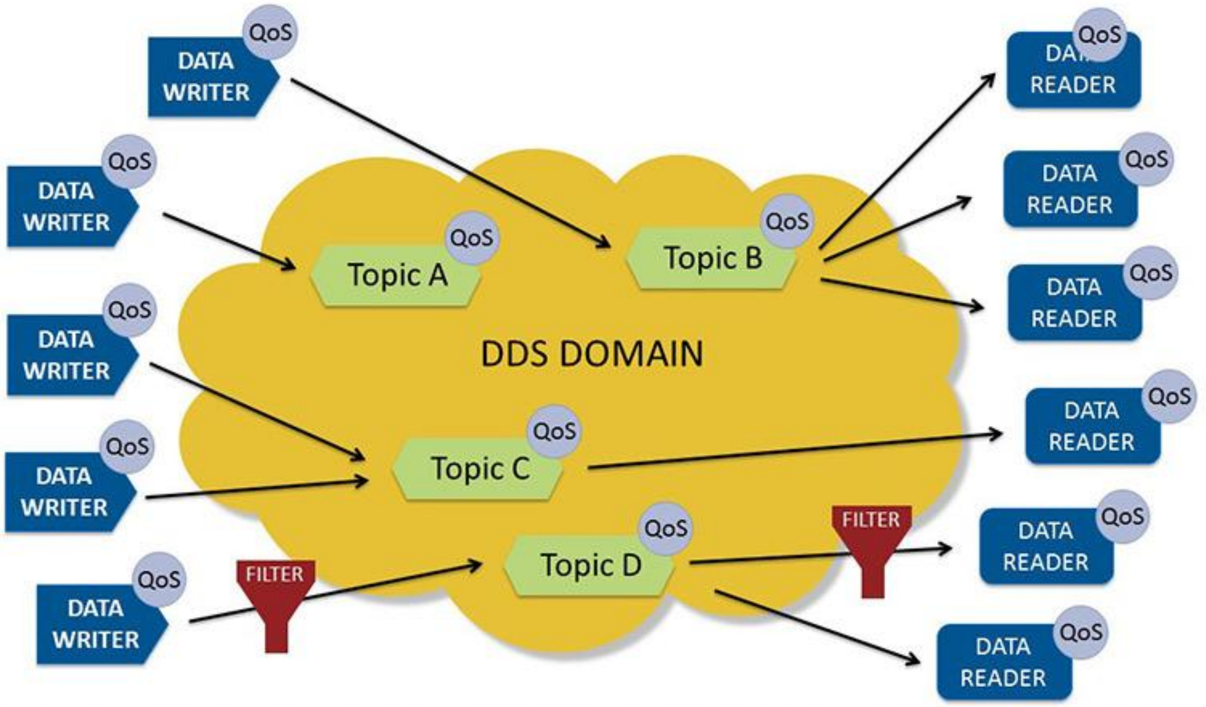


Figure 8.1: Communication within DDS domain by publishing and subscribing to Topics identified by their Topic name. [DDS]

DDS also has the capability to support multiple domains, thus providing developers a system that can scale with system needs or segregate based on different data types. When a specific data instance is published on one domain, it will not be received by subscribers residing on any other domains.

Multiple domains provide effective data isolation. One use case would be for a system to be designed whereby all Command/Control related data is exchanged via one domain while Status information is exchanged within another.

8.1.4. Data Writers and Publishers

Data Writers are the primary access point for an application to publish data into a DDS data domain. Once created and configured with the correct QoS settings, an application only needs to perform a simple write call.

Subscribers may have different requirements for how often they want to receive data. Some subscribers may want every individual sample of data, while others may want data at a much slower rate. This can be achieved by specifying a different time-based filter QoS for each subscriber.

If a time-based filter is specified for an associated subscriber, then the publisher could be implemented to not send data to that subscriber any faster than is required. This would therefore reduce overall network bandwidth. When the write is executed, the DDS software will move the data from its current container, Data Writer, into a Publisher for sending out to the DDS Domain. Figure 8.2 shows how the entities (Domain Participant, Topic, Data Writer, and Publisher) needed to publish data are connected.

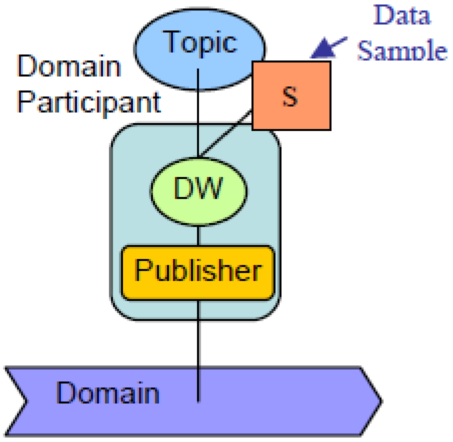


Figure 8.2: Publication model [DDS]

The Publisher entity is just a container to group together individual Data Writers. [DDS]

8.1.5. Data Readers and Subscribers

A Data Reader is the primary access point for an application to access data that has been received by a Subscriber. Figure 8 shows the entities associated with subscriptions.

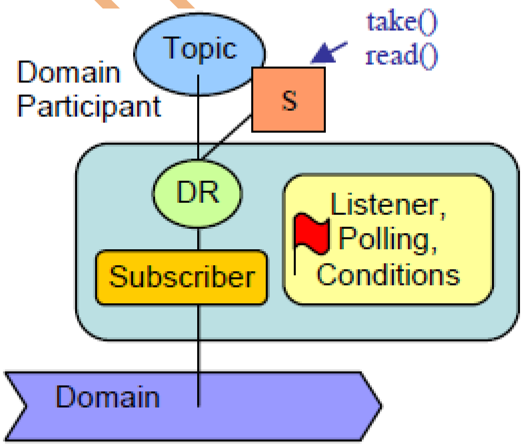


Figure 8.3: Subscription model [DDS]

Once created and configured with the correct QoS, an application can be notified that data is available in one of three ways:

- Listener Callback Routine;
- Polling the Data Reader;
- Conditions and WaitSets.

The first method for accessing received data is to set up a listener callback routine that DDS will run immediately when data is received. You can execute your own specific software inside that callback routine to access the data.

The second method is to “poll” or query the Data Reader to determine if data is available.

Finally, a “WaitSet”, can be set up when until a specified condition is met and the application accesses the data from the Data Reader.

Just as Publishers are used to group together Data Writers, Subscribers are used to group together Data Readers. Again, this allows you to configure a default set of QoS parameters and event handling routines that will apply to all the Data Readers in that Subscriber’s group. [DDS]

8.1.6. Topics

Topics provide the basic connection point between publishers and subscribers. The Topic of a given publisher on one node must match the Topic of an associated subscriber on any other node. If the Topics do not match, communication will not take place.

A Topic is comprised of a Topic Name and a Topic Type. The Topic Name is a string that uniquely identifies the Topic within a domain. The Topic Type is the definition of the data contained within the Topic. Topics must be uniquely defined within any one particular domain.

The DDS specification suggests that types be defined in Interface Definition Language (IDL) files. IDL is the OMG standard language for defining object/data interfaces. The syntax for IDL is very similar to C++.

```
struct RndNum {
  float data;
  short processed;
  unsigned long seqNumber; //key
};
```

Figure 8.4: Example of sample IDL type

In this example, the type RndNum is a Topic Type. The Topic Name may be any string chosen by the application, such as “Random Numbers” or “Radar Samples”.

8.1.7. Topic Keys

Within the definition of the Topic Type, one or more data elements can be chosen to be a “Key” for the type. The DDS middleware will use the Key to sort incoming data. By specifying one data element to be a Key, an application can then retrieve data from DDS that matches a specific key, or matches the next Key in a sequence of Keys.

Keys provide scalability and eliminate needs to define many individual topics.

8.1.8. MultiTopics & ContentFilteredTopics

In addition to standard Topics, there are also constructs in place for MultiTopics and ContentFilteredTopics.

A MultiTopic is a logical grouping of several Topics. It allows an application to select fields from multiple types and recombine them into a new type (something like an SQL SELECT statement).

A ContentFilteredTopic allows you to declare a filter expression by which only individual data samples of the specified Topic would be received and presented to the application.

8.1.9. Quality of Service in DDS

The provision of QoS on a per-entity basis is a significant capability provided by DDS. Being able to specify different QoS parameters for each individual Topic, Reader or Writer gives developers a large palette from which to design their system. This is the essence of data centricity within DDS.

The DDS QoS parameters are: deadline, destination order, durability, entity factory, group data, history, latency budget, lifespan, liveliness, ownership, ownership strength, partition, presentation, reader data, lifecycle, reliability, resource limits, time-based filter, topic data, transport priority, user data, writer data lifecycle.

8.1.10. Summary

The Data Distribution Service is represented from the OMG specification that creates a very simple architecture for data communication, while enabling very complex data patterns.

DDS provides the real-time, many-to-many, managed connectivity required by high-performance machine applications.

DDS implementations can deliver hundreds of thousands of messages (or more with batching) to thousands of recipients per second. Timing control also makes DDS uniquely suited for real-time systems.

Topics allow endpoint nodes to be abstracted from each other, so nodes can enter and leave the distributed application dynamically. DDS is “data-centric”—all the QoS parameters can be changed per endpoint basis. This per endpoint configurability is here the key to supporting complex data communication patterns.

8.2.Artemis ActiveMQ

Apache ActiveMQ Artemis [ActiveMQ Artemis] is an open source project to build a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system. It is an example of Message Oriented Middleware (MoM). Artemis ActiveMQ originates from HornetQ messaging system which was donated to Apache in 2014. It retains compatibility with HornetQ while adding many features.

Main features of Apache ActiveMQ Artemis are:

- 100% open source software. Apache ActiveMQ Artemis is licensed using the Apache Software License v 2.0 to minimise barriers to adoption;
- written in Java and run on any platform with a Java 8+ runtime, that's everything from Windows desktops to IBM mainframes;
- Apache ActiveMQ Artemis has a high level of performance for persistent and non-persistent messaging;
- elegant, clean-cut design with minimal third-party dependencies. ActiveMQ Artemis can be run stand-alone, run it in integrated JEE application server, or run it embedded inside custom product;
- provision of high availability with automatic client failover functionality to guarantee zero message loss or duplication in event of server failure;
- allows a flexible clustering. Apache ActiveMQ Artemis create clusters of servers that know how to load balance messages. It is possible to link geographically distributed clusters over unreliable connections to form a global network. This implies also the configuration routing of messages in a highly flexible way.

Artemis ActiveMQ messaging concepts provides / supports:

- asynchronous point-to-point and publish-subscribe messaging: with point-to-point messaging, there can be many consumers on the queue, but a particular message will only ever be consumed by a maximum of one of them. With publish-subscribe messaging many senders can send messages to an entity on the server, often called a *topic.* There can be many *subscriptions* on a topic, a subscription is just another word for a consumer of a topic. Each subscription receives a *copy* of *each* message sent to the topic;
- transactions: supports the sending and acknowledgement of message as part of a large global transaction;
- delivery guarantees: with reliable messaging the server gives a guarantee that the message will be delivered once and only once to each consumer of a queue or each durable subscription of a topic, even in the event of system failure;

- durability: durable messages will be persisted in permanent storage and will survive server failure or restart;
- messaging APIs and protocols:
  - **Java Message Service (JMS):** it is part of Oracle's JEE specification. It's a Java API that encapsulates both message queue and publish-subscribe messaging patterns. JMS is a very popular API and is implemented by most messaging systems. JMS is only available to clients running Java. JMS does not define a standard wire format - it only defines a programmatic API so JMS clients and servers from different vendors cannot directly interoperate since each will use the vendor's own internal wire protocol. Apache ActiveMQ Artemis provides a fully compliant JMS 1.1 and JMS 2.0 API,
  - **System specific APIs:** API's like JMS are not normally rich enough to expose all the extra features that most messaging systems provide. Apache ActiveMQ Artemis provides its own core client API for clients to use if they wish to have access to functionality over and above that accessible via the JMS API,
  - **RESTful API:** allows to leverage the reliability and scalability features of Apache ActiveMQ Artemis over a simple REST/HTTP interface. The REST Interface implementation sits on top of an Apache ActiveMQ Artemis JMS API and as such exposes JMS like concepts via REST,
  - **AMQP**: is a specification for interoperable messaging. It also defines a wire format, so any AMQP client can work with any messaging system that supports AMQP. Apache ActiveMQ Artemis implements the AMQP 1.0 specification. Any client that supports the 1.0 specification will be able to interact with Apache ActiveMQ Artemis,
  - **MQTT:** is a lightweight connectivity protocol. It is designed to run in environments where device and networks are constrained. Out of the box Apache ActiveMQ Artemis supports version MQTT 3.1.1. Any client supporting this version of the protocol will work against Apache ActiveMQ Artemis,
  - **STOMP**: is a very simple text protocol for interoperating with messaging systems. It defines a wire format, so theoretically any Stomp client can work with any messaging system that supports Stomp. Stomp clients are available in many different programming languages,
  - **OPENWIRE:** ActiveMQ 5.x defines its own wire Protocol "OPENWIRE". In order to support ActiveMQ 5.x clients, Apache ActiveMQ Artemis supports OPENWIRE. Any ActiveMQ 5.12.x or higher can be used with Apache ActiveMQ Artemis.

In addition to the protocols above ActiveMQ Artemis also offers support for it's own highly performant native protocol "Core".

Apache ActiveMQ Artemis core is designed simply as set of Plain Old Java Objects (POJOs). This means it can be instantiated and run in any dependency injection framework such as

Spring or Google Guice. Each ActiveMQ Artemis server has its own high performance persistent journal, which it uses for message and other persistence. ActiveMQ Artemis clients, potentially on different physical machines interact with the ActiveMQ Artemis server.

ActiveMQ Artemis high level architecture is presented on Figure 8.5.

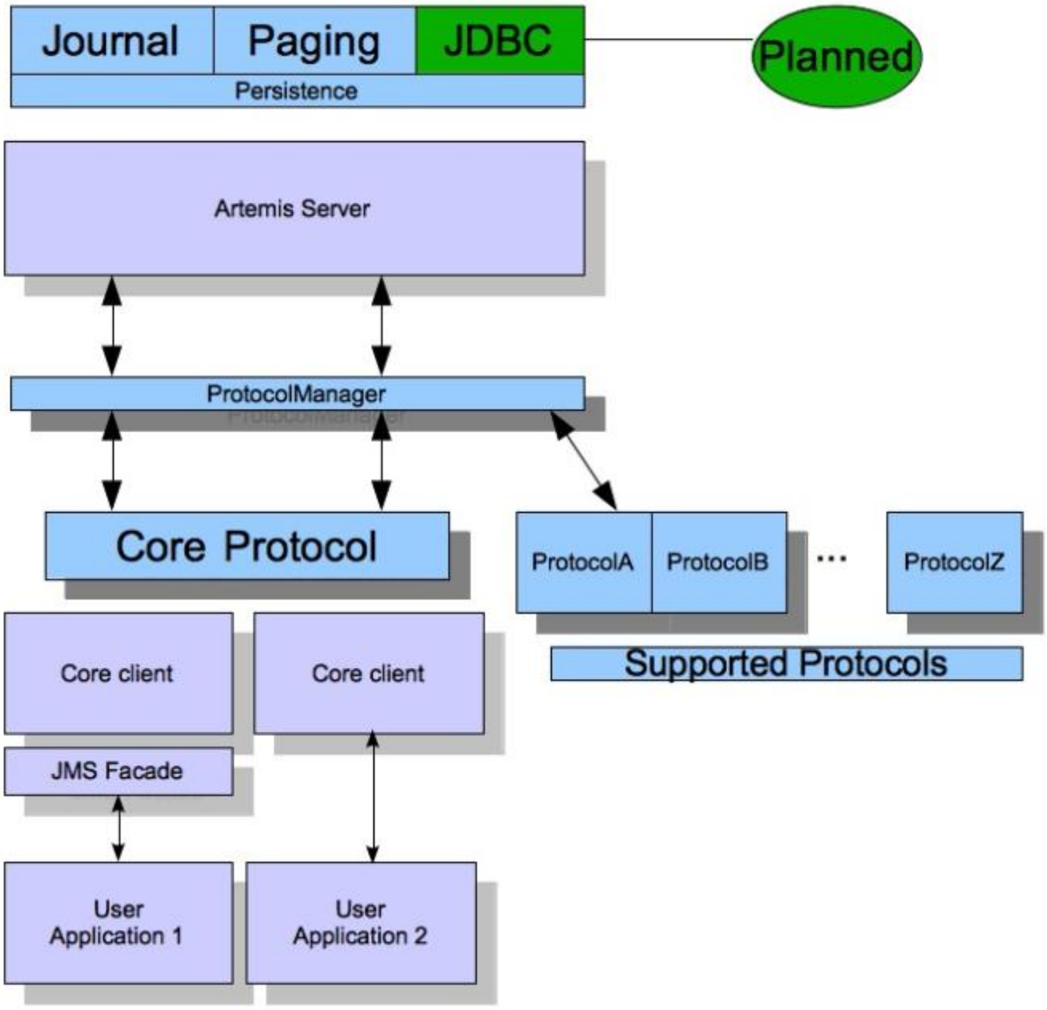


Figure 8.5: ActiveMQ Artemis high level architecture [ActiveMQ Artemis]

On the diagram two user applications interacting with an Apache ActiveMQ Artemis server. User Application 1 is using the JMS API implemented by a facade layer on a client side, while User Application 2 is using the core client API directly.

Apache ActiveMQ Artemis has an extensive management API that allows a user to modify a server configuration, create new resources (e.g. addresses and queues), inspect these resources (e.g. how many messages are currently held in a queue) and interact with it (e.g. to remove messages from a queue). All the operations allow a client to manage Apache ActiveMQ Artemis. It also allows clients to subscribe to management notifications.

There are 3 ways to manage Apache ActiveMQ Artemis:

- using JMX -- JMX is the standard way to manage Java applications;
- using the core API -- management operations are sent to Apache ActiveMQ Artemis server using core messages;
- using the JMS API -- management operations are sent to Apache ActiveMQ Artemis server using JMS messages.

Although there are 3 different ways to manage Apache ActiveMQ Artemis each API supports the same functionality. If it is possible to manage a resource using JMX it is also possible to achieve the same result using Core.

8.3. Infinispan

Infinispan [Infinispan] is a distributed in-memory key/value data store with optional schema, available under the Apache License 2.0.

- available as an embedded Java library or as a language-independent service accessed remotely over a variety of protocols (Hot Rod, REST, Memcached and WebSockets);
- use it as a cache or a data grid;
- advanced functionality such as transactions, events, querying, distributed processing, off-heap and geographical failover;
- monitor and manage it through JMX, a CLI and a web-based console;
- integrates with JPA, JCache, Spring, Spark and many more;
- works on AWS, Azure, Google Cloud and OpenShift.

8.3.1. Runtimes

Infinispan can be used in a variety of runtimes:

- Java SE, started by your application;
- an application server which provides Infinispan as a service (such as JBoss AS);
- bundled as a library in your application, deployed to an application server, and started on by your application (for example, you could use Infinispan with Tomcat or GlassFish);
- inside an OSGi runtime environment (such as Apache Karaf).

8.3.2. Modes

Infinispan offers four modes of operation, which determine how and where the data is stored:

- local, where entries are stored on the local node only, regardless of whether a cluster has formed. In this mode Infinispan is typically operating as a local cache;

- invalidation, where all entries are stored into a cache store (such as a database) only, and invalidated from all nodes. When a node needs the entry, it will load it from a cache store. In this mode Infinispan is operating as a distributed cache, backed by a canonical data store such as a database;
- replication, where all entries are replicated to all nodes. In this mode Infinispan is typically operating as a data grid or a temporary data store, but doesn't offer an increased heap space;
- distribution, where entries are distributed to a subset of the nodes only. In this mode Infinispan is typically operating as a data grid providing an increased heap space;
- scattered, which is like a Distribution mode but is more suitable for write-intensive applications.

Invalidation, Replication and Distribution can all use synchronous or asynchronous communication, Scattered mode is only synchronous.

Infinispan 5 adds the ability to pass a Runnable around the grid. This allows to push complex processing towards the server where data is local, and pull back results using a Future. This map/reduce style paradigm is common in applications where a large amount of data is needed to compute relatively small results (Compute Grids).

Infinispan uses TCP/IP for sending packets over the network (for both cluster communication when using TCP stack or when communication with Hot Rod clients).

8.3.3. Capacity planning

Data in Infinispan is either stored as plain Java objects or in a serialized form, depending on operating mode (embedded or server) or on specific configuration options (store-as-binary). Data size can be estimated using sophisticated tools like Java Object Layout and the total amount of required memory can be roughly estimated using the following formulas:

Total Data Set in library mode:

TotalDataSet = NumberOfEntries · (KeySize + ValueSize + 200b(Overhead))

Total Data Set in server mode:

TotalDataSet = NumberOfEntries · (SerializedKeySize + SerializedValueSize + 200b(Overhead))

Term overhead is used here as an average amount of additional memory (e.g. expiration or eviction data) needed for storing an Entry in a Cache.

Best performance using Cache Stores is achieved by:

- using async mode (write-behind) if possible;
- preventing cache misses by preloading data;
- for JDBC Cache Store:

- using indexes on id column to prevent table scans,
- using PRIMARY_KEY on id column,
- configuring batch-size, fetch-size, etc.

Infinispan can be used also as a database replacement. In such case the Hibernate OGM can replace the RDBMS and store entities and relations directly in Infinispan, interacting with it through the well known JPA 2.1 interface. Hibernate OGM also automates mapping, encoding and decoding of JPA entities to Protobuf.

8.3.4. Summary

Infinispan is an open source data grid platform. It exposes a JSR-107 compatible cache interface (which in turn extends java.util.Map) in which object can be store. While Infinispan can be run in local mode, its real value is in distributed mode where caches cluster together and expose a large memory heap. Distributed mode is more powerful than simple replication since each data entry is spread out only to a fixed number of replicas thus providing resilience to server failures as well as scalability since the work done to store each entry is constant in relation to a cluster size.

8.4. Hazelcast

Hazelcast IMDG (In-Memory Data Grid) [Hazelcast IMDG] provides a convenient interface for developers to work with distributed data structures and other aspects of in-memory computing. In its simplest configuration, Hazelcast can be treated as an implementation of the java.util.ConcurrentMap that can be accessed from multiple application instances of several types such as Java, C++, C#, .NET, Scala, Python, Node.js etc, including Hazelcast instances that are spread out across the network.

8.4.1. In-memory data grid idea

In-memory data grid is data management software that enables:

- **Scale-out Computing:** every node adds their CPU to the cluster;
- **Resilience:** nodes can fail randomly without data loss or significant performance impact to running applications;
- **Programming Model:** a way for developers to easily program the cluster of machines as if it were a single machine;
- **Fast, Big Data:** it enables very large data sets to be manipulated in main memory;
- **Dynamic Scalability:** nodes (computers) can dynamically join the other computers in a grid (cluster);
- **Elastic Main Memory:** every node adds their RAM to the cluster's memory pool.

In-memory data grids are often used with databases to improve performance of applications, to distribute data across servers, clusters and geographies and to manage very large data sets or very high data ingest rates.

8.4.2. Hazelcast characteristics

Hazelcast IMDG has the following characteristics:

- the data is always stored in-memory (RAM) of the servers;
- multiple copies are stored in multiple machines for automatic data recovery in case of single or multiple server failures;
- the data model is object-oriented and non-relational;
- servers can be dynamically added or removed to increase the amount of CPU and RAM;
- the data can be persisted from Hazelcast to a relational or NoSQL database;
- a Java Map API accesses the distributed key-value store.

The primary capabilities that Hazelcast provides include:

- elasticity;
- redundancy;
- high performance.

Elasticity means that Hazelcast clusters can grow capacity on demand, simply by adding new nodes. Redundancy means that have flexibility when configure Hazelcast clusters for data replication policy (which defaults to one synchronous backup copy).

To support these capabilities, Hazelcast has a concept of members – members are Hazelcast instance JVMs that join a Hazelcast cluster. A cluster provides a single extended environment where data can be synchronized between (and processed by) members of the cluster.

The core Hazelcast technology is characterized to:

- be open source;
- be written in Java:
  - supports Java 6, 7 and 8,
  - use minimal dependencies,
  - have simplicity as a key concept.

8.4.3. Hazelcast IMDG Editions

- Hazelcast or Hazelcast IMDG refers to the open source edition of Hazelcast in-memory data grid middleware. Hazelcast is also the name of the company (Hazelcast, Inc.) providing the Hazelcast product;

- Hazelcast IMDG Enterprise is a commercially licensed edition of Hazelcast which provides high-value enterprise features in addition to Hazelcast;
- Hazelcast IMDG Enterprise HD is a commercially licensed edition of Hazelcast which provides High-Density (HD) Memory Store and Hot Restart Persistence features in addition to Hazelcast Enterprise.

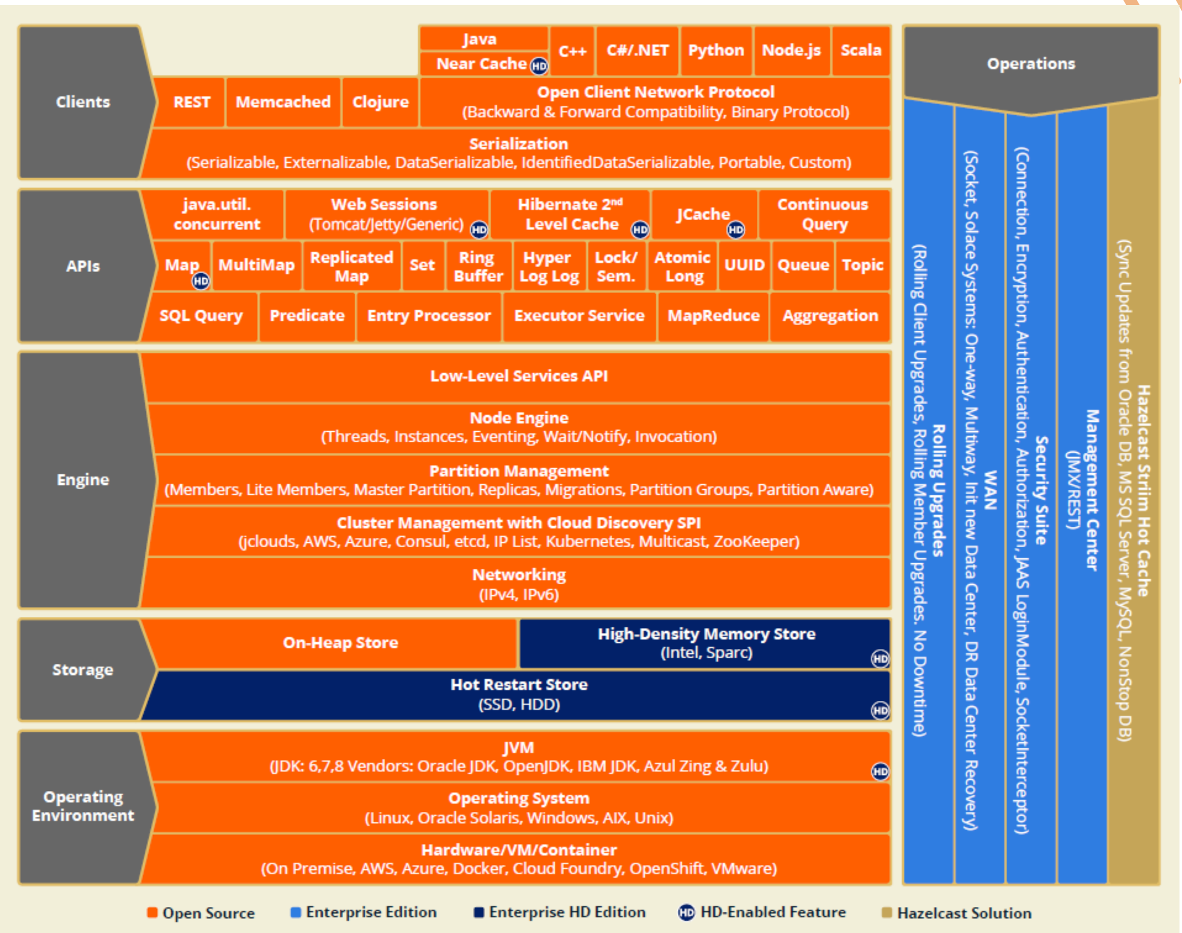

Clients
Java
Near Cache HD
C++
C#/.NET
Python
Node.js
Scala
REST
Memcached
Clojure
Open Client Network Protocol
(Backward & Forward Compatibility, Binary Protocol)
Serialization
(Serializable, Externalizable, DataSerializable, IdentifiedDataSerializable, Portable, Custom)
APIs
java.util. concurrent
Web Sessions (Tomcat/Jetty/Generic) HD
Hibernate 2nd Level Cache HD
JCache HD
Continuous Query
Map HD
MultiMap
Replicated Map
Set
Ring Buffer
Hyper Log Log
Lock/ Sem.
Atomic Long
UUID
Queue
Topic
SQL Query
Predicate
Entry Processor
Executor Service
MapReduce
Aggregation
Engine
Low-Level Services API
Node Engine
(Threads, Instances, Eventing, Wait/Notify, Invocation)
Partition Management
(Members, Lite Members, Master Partition, Replicas, Migrations, Partition Groups, Partition Aware)
Cluster Management with Cloud Discovery SPI
(jclouds, AWS, Azure, Consul, etcd, IP List, Kubernetes, Multicast, ZooKeeper)
Networking
(IPv4, IPv6)
Storage
On-Heap Store
High-Density Memory Store
(Intel, Sparc) HD
Hot Restart Store
(SSD, HDD) HD
Operating Environment
JVM
(JDK: 6,7,8 Vendors: Oracle JDK, OpenJDK, IBM JDK, Azul Zing & Zulu) HD
Operating System
(Linux, Oracle Solaris, Windows, AIX, Unix)
Hardware/VM/Container
(On Premise, AWS, Azure, Docker, Cloud Foundry, OpenShift, VMware)
Operations
Rolling Upgrades
(Rolling Client Upgrades, Rolling Member Upgrades, No Downtime)
WAN
(Socket, Solace Systems: One-way, Multiway, Init new Data Center, DR Data Center Recovery)
Security Suite
(Connection, Encryption, Authentication, Authorization, JAAS LoginModule, SocketInterceptor)
Management Center
(JMX/REST)
Hazelcast Striim Hot Cache
(Sync Updates from Oracle DB, MS SQL Server, MySQL, NonStop DB)
Open Source
Enterprise Edition
Enterprise HD Edition
HD HD-Enabled Feature
Hazelcast Solution

Figure 8.6: Hazelcast IMDG In-Memory Computing Platform Architecture Diagram [Hazelcast IMDG]

Hazelcast IMDG capabilities targeted for enterprise applications are as follows:

- HD Memory: is a Hazelcast Enterprise HD feature. It allows to scale in-memory data storage capacity up to Terabytes of main memory in a single JVM. HD Memory eliminates garbage collection issues by minimizing pauses caused by GC, and delivers scaled up and predictable performance;
- Security: provides standards-based JAAS and interoperable encryption, authentication, and access control checks to mission-critical applications;

- Management Center: provides a bird's-eye view of all cluster activity, along with configurable watermarks for alerts through a web-based user interface and cluster-wide JMX and REST APIs;
- Hot Restart Store: distributed persistence with high write performance and low restart times with parallel loading of data across nodes;
- WAN Replication: synchronizes multiple Hazelcast clusters in different datacenters for disaster recovery or geographic locality and can be managed centrally through Management Centre;
- Rolling Upgrades: is a Hazelcast IMDG Enterprise feature allows to upgrade cluster nodes' versions without service interruption.

8.4.4. Client APIs

The original API for Hazelcast IMDG is the Java API, but recent versions have added a wide range of other language options.

Officially Supported Languages:

- Java;
- C++/.NET;
- Node.js;
- Python;
- Scala.

In addition, recently the Hazelcast user community has added additional languages to directly access Hazelcast clusters:

- Clojure;
- Ruby (in progress).

Hazelcast includes an Open Binary Client Protocol. This has support for versioning and backwards compatibility, and allows additional APIs to be written in any client language.

8.4.5. Cloud presence

With Hazelcast IMDG 3.8, support for cloud environments and infrastructure technologies has been widened. This has focused on auto-discovery in popular clouds, using a technology called Cloud Discovery Service Provider Interface (SPI).

Supported Cluster Management and Cloud Discovery:

- Apache jclouds;
- AWS;

- Azure;
- Consul;
- Etcd;
- IP List;
- Eureka;
- Kubernetes;
- Multicast;
- Zookeeper.

Supported Containers and Virtual Machines:

- Azure;
- AWS;
- Cloud Foundry;
- Docker;
- OpenShift;
- VMware.

8.4.6. Deployment topologies

Hazelcast IMDG supports two modes of operation, either “embedded member”, where the JVM containing application code joins the Hazelcast cluster directly, or “client plus member”, whereby independent Hazelcast server instances running on same or different host, form the Hazelcast cluster. These two approaches to topology are shown in the following diagrams.

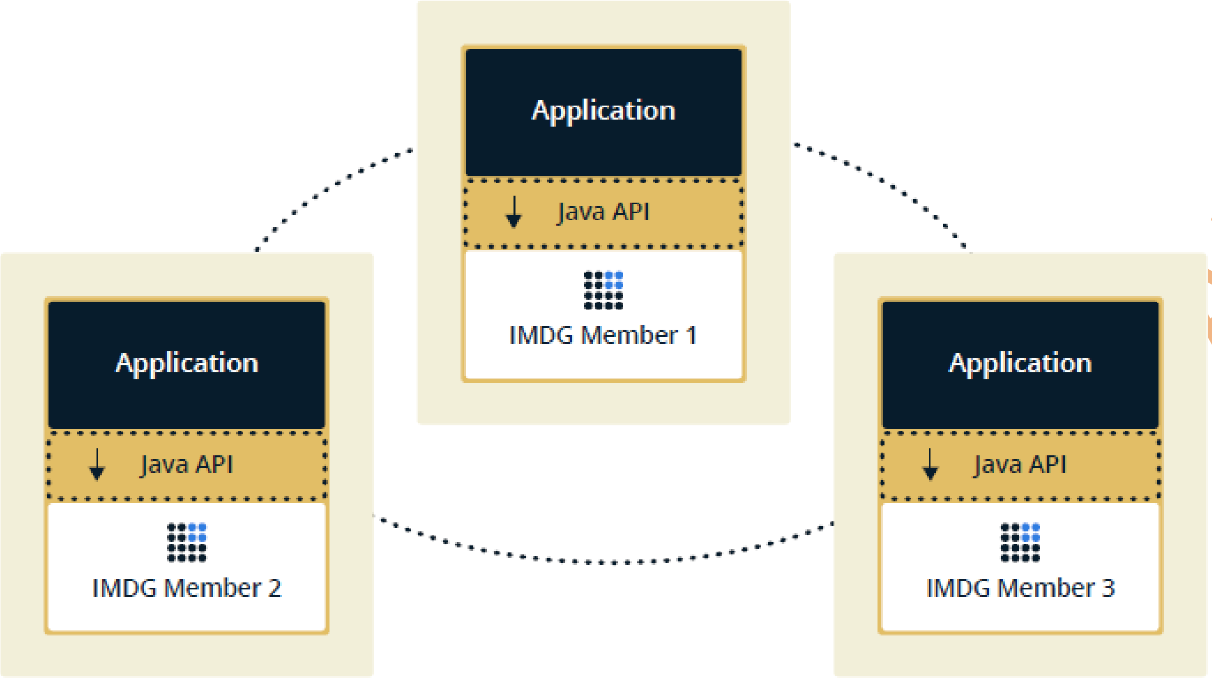


Figure 8.7: Hazelcast IMDG deployment topology - embedded approach [Hazelcast IMDG]

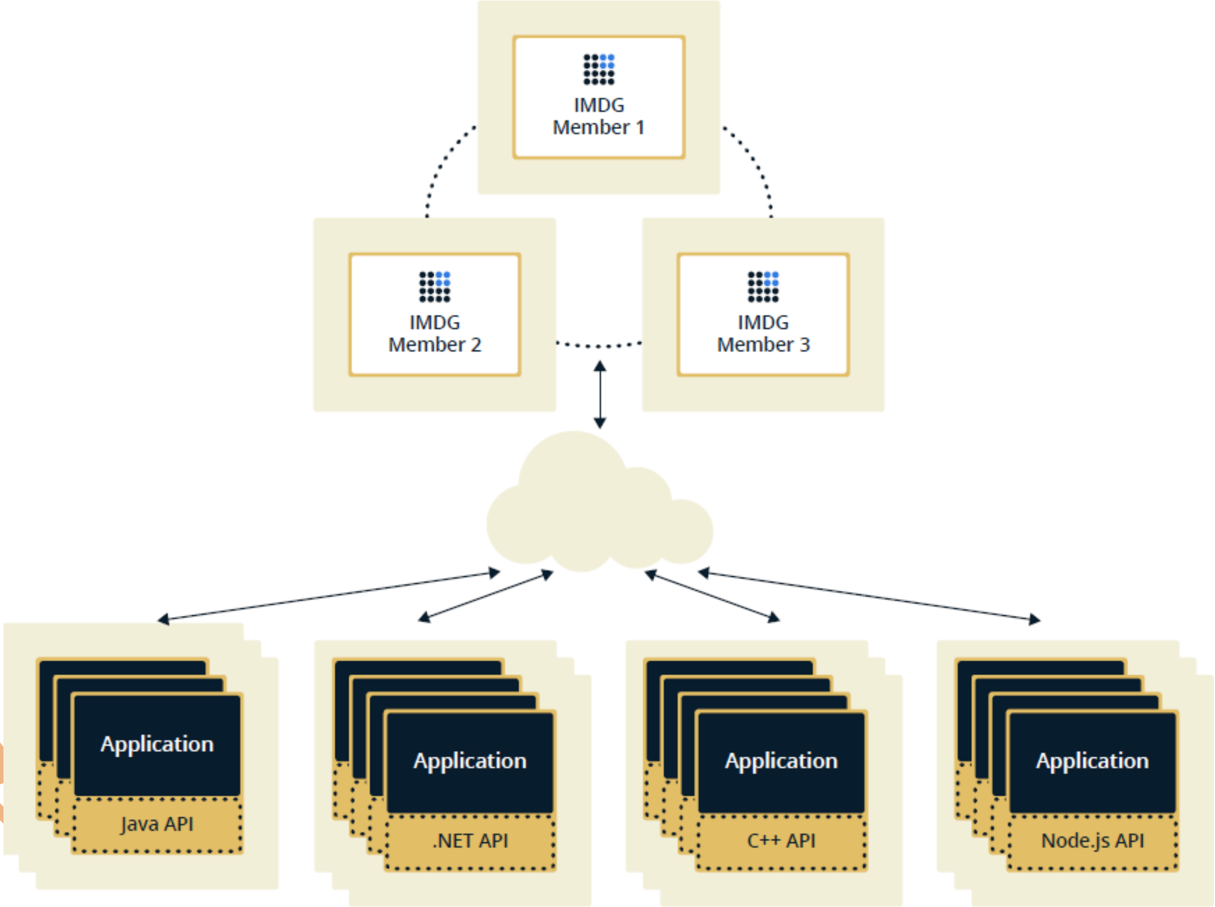


Figure 8.8: Hazelcast IMDG deployment topology - client plus member [Hazelcast IMDG]

Under most circumstances, client plus member topologies is recommended, as it provides greater flexibility in terms of cluster mechanics – member instances can be taken down and restarted without any impact to the overall application, as the Hazelcast client will simply reconnect to another member of the cluster. Another way of saying this is that client plus member topologies isolate application code from purely cluster-level events.

Hazelcast allows clients to be configured within the client code (programmatically), or by XML, or by properties files. Clients have quite a few configurable parameters, including known members of the cluster. Hazelcast will discover the other members as soon as they are online, but they need to connect first. In turn, this requires the user to configure enough addresses to ensure that the client can connect into the cluster somewhere.

8.4.7. Architecture

The below diagram presents example of the client – server architecture of the solution based on Hazelcast IMDG. Main idea of it utilize publish-subscribe pattern which increase decupling modules by using topic bus. Topic bus act as ESB (Enterprise Servise Bus) introducing topic as additional message category.

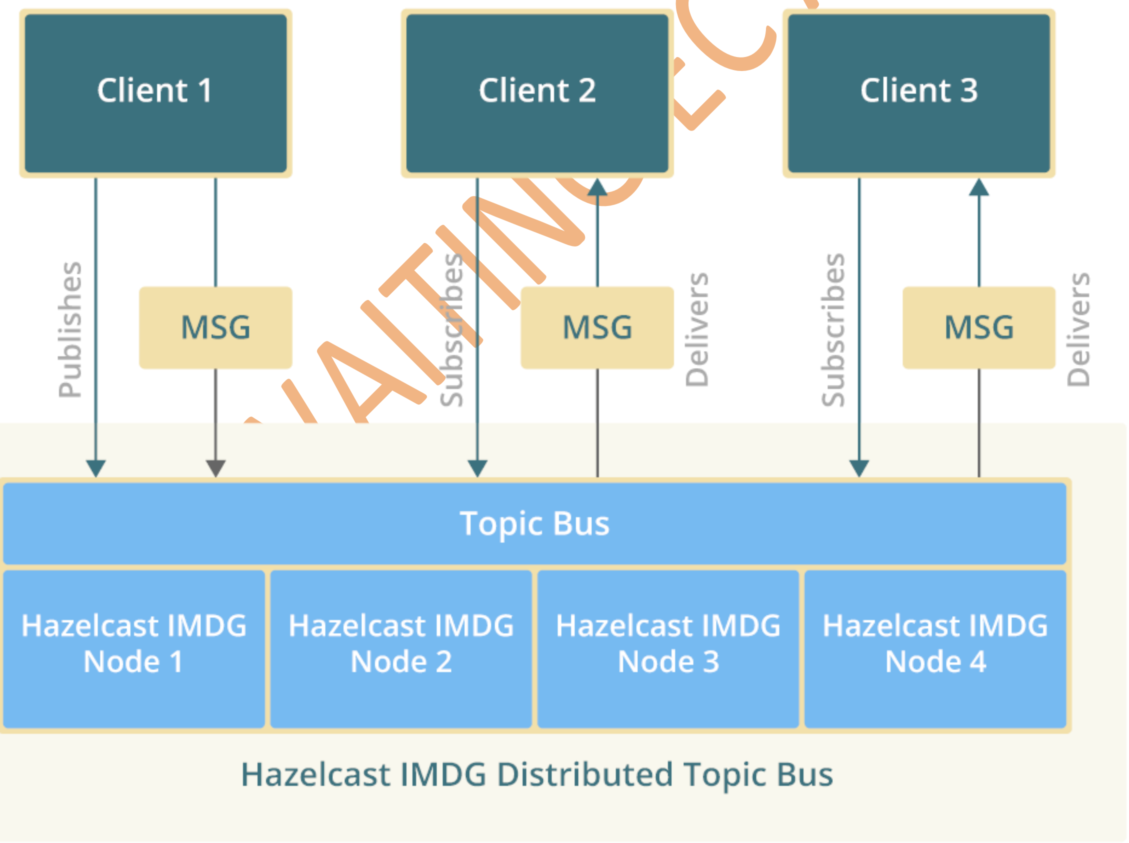


Figure 8.9: Hazelcast IMDG client - server architecture [Hazelcast IMDG]

8.4.8. Replication and serialization

To handle data sets too large to fit into a single JVM, Hazelcast partitions the data into local sets and distributes the partitions as evenly as possible, based on the map key. Hazelcast's elasticity features are enabled by an algorithm that will rebalance the partitions when members join or leave a cluster.

Hazelcast provides a simple naming scheme for controlling which partitions data resides in. This means that the developer can ensure the locality of related data. When Hazelcast's compute capability is in use, this extends to provide the capability of sending a data processing task (often expressed as a Runnable) to the same partition as the data upon which it will operate.

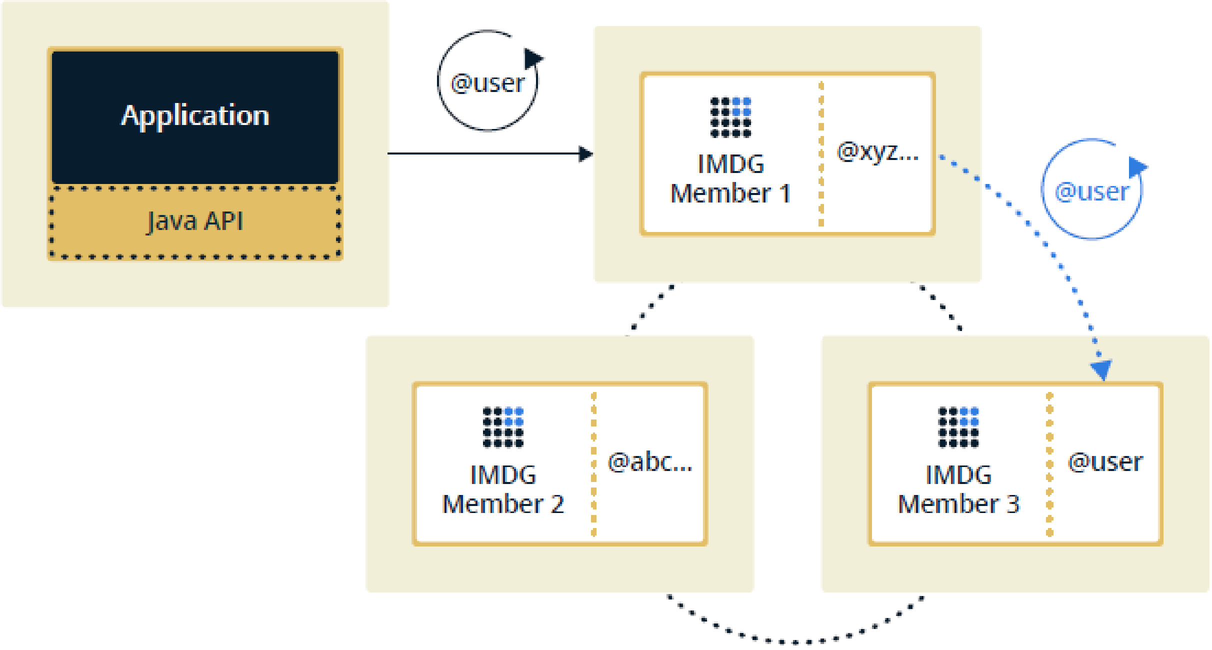


Figure 8.10: Hazelcast IMDG replication [Hazelcast IMDG]

In the case of a hard failure (e.g. JVM crash or kernel panic), Hazelcast has recovery and failover capabilities to prevent data loss. In the worst case, there is always at least one synchronous backup of every piece of data (unless that was explicitly disabled), so the loss of a single member will never cause data loss.

Hazelcast provides a certain amount of control over how data is replicated. The default is 1 synchronous backup. For data structures such as IMap and IQueue, this can be configured using the config parameters backup-count and async-backup-count. These parameters provide additional backups, but at the cost of higher memory consumption (each backup consumes memory equivalent to the size of the original data structure). The asynchronous option is really designed for low-latency systems where some very small window of data loss is preferable over increased write latency.

The key to how replication takes place is Hazelcast's use of serialized Java objects. By default, Hazelcast IMDG serializes objects to byte arrays as they are stored, and deserializes them when they are read. This serialization and deserialization happens only on the client thread. The serialized representation of an object is called the BINARY format. This method shall be used when accessing the data using traditional "get()", "put()" and similar methods.

8.4.9. Summary

Hazelcast IMDG is an open source alternative to other in-memory grid solutions like Oracle Coherence, Software AG Terracotta, Pivot Gemfire.

It provides rich features such as Entry Processing and distributed Execution Services that go beyond caching data and enable programmers to access the distributed processing power of the grid.

8.5. Redis

This section is based on information available on Internet at the following address: https//redis.io, last retrieved on 20 December 2017.

Redis is an in-memory data structure store available under the BSD licence. Redis can be used as a database, a cache, or a message broker.

8.5.1. Terminology

The meaning of some terms used in this section is defined as follow:

- **Client:** see following definition of Redis client;
- **Command:** a command is sent by the client to the server to make operations on elements stored by the server, or to invoke some server features;
- **Element**: a key-value pair stored, managed and provided by a Redis Server;
- **Redis**: the generic name of the product. Depending on the context, it may also refer to the Redis server;
- **Redis client**: an application program that connects to a Redis server and exploits the Redis features;
- **Redis element:** see previous definition of Element;
- **Redis server**: the actual implementation of the Redis Server that stores the Redis elements, and provides other Redis features;
- **Server**: see previous definition of Redis Server.

8.5.2. Properties of Redis elements

Redis stores elements (key-value pairs) in in-memory data sets. Elements can be persisted by dumping the dataset to disk, or by logging each operation.

8.5.2.1. Value - Data structures supported by Redis

Redis stores key-value elements; what makes Redis a data structures server is its support of different kinds of data structures for values. Besides the traditional element made up of a string key and a string value, a key can be associated with value of the following types:

- Binary-safe string: a string consisting of any sequence of characters (bytes);
- List: a linked list of string elements sorted according the order of insertion;
- Set: a set of unique, unsorted string elements;
- Sorted set: a set of string elements, each having a score associated to it. Ranges of elements are retrieved sorted by their score;
- Hashes (maps): a key-value where the value contents is an associative array;
- Bitmaps (Bit arrays): a key-value where the value is handled as a bit array;
- HyperLogLogs: a probabilistic data structure used to count unique objects;
- Geospatial Indexes: a sorted set of geospatial items (longitude, latitude, name), given assuming the Earth is a perfect sphere (worst case error up to 0.5%).

8.5.2.2. Key

Redis keys are binary-safe strings, this means that a key can be either a string of characters (like, e.g. “bozo_age”), or a sequence of bits up to 512MB (e.g. the content of a JPEG file).

8.5.2.3. Expiration

Time out can be set to any element stored by Redis. The time out is associated with the key of the element, and it behaves like a time to live. At the expiration of the timeout, the element is deleted from the in-memory data set. Timeouts are expressed either in second or milliseconds, with a 1 millisecond precision. Expiration time for each element is persisted on disk to enable consistency in case of Redis server stop or restart.

8.5.3. Redis Features (ASTS)

The following are the main features provided by Redis.

8.5.3.1. Commands

Redis commands are issued by a Redis client to perform operations on Redis server.

Commands work on elements (i.e. key-value pairs and the data structures they hold), transactions, Redis configuration, scripting, connections, and clustering operations.

The following are examples of commands on elements:

- set myCode “ATAF1000V” – set the value of the element identified by key myCode to the binary-safe string ATAF1000V;
- get myCode – retrieves the value of the element identified by key myCode;
- expire myCode 10 – set an expiration timeout of 10s on the element identified by key myCode. When the expiration timeout is exhausted, the element is destroyed.

Redis performs the following atomic operations:

- String append: appends a string to an existing element, or otherwise create a new element with the given string;
- Value increment in a hash: increments a specified entry of the associative array stored in the specified element;
- Push to list: pushes the specified items to the list stored in the specified element, or otherwise creates an element holding a list with the given items;
- Intersection, Union, and difference on sets; given two or more elements storing sets, performs the Intersection, the union, or the set difference;
- Selection of set-elements: selects and returns a sorted sub-set from the specified Redis element storing a sorted set in its value.

The above operations don't require a GET-MODIFY-SET sequence on the specified element.

8.5.3.2. Transactions

Transactions allow execution of group of Redis commands in a single step. A transaction has the following properties:

- all commands in a transaction are queued and executed sequentially. During the execution of queued commands, no other commands issued by another Redis client can be executed till the end of the transaction;
- either all the commands of a transaction or none of them are processed. Redis transaction are atomic.

A Transaction can by either successful or faulty. Redis differs from relational database transactions because a faulty command will not interrupt the transaction, and there's no rollback available.

8.5.3.3. Publish/Subscribe (Pub/Sub)

Pub/Sub implements the Publish/Subscribe paradigm. Redis provides channels used by publisher client and subscriber client. Subscriber clients join channels and receive messages sent by publisher clients without knowing the publisher. Publishers are also unaware of subscribers. Each Channel is univocally identified by its own channel name. Subscriber clients can specify can specify more than one channel at subscribe; subscriber clients can also specify a pattern to subscribe to channels whose name matches the pattern. Pattern specification support globing, wild-card and character-classes.

Two reserved channels are used by Redis server itself to notify set, delete, or expiration on any elements to subscribing clients.

8.5.3.4. Key-space and Key-event Notifications

Operations on Redis elements generate two types of event notifications:

Key-space notification - This notification is delivered on a dedicated Pub/Sub channel and it carries two data: the element-key and the command executed on the element

Key-event notification - This notification is delivered on a dedicated Pub/Sub channel and it carries two data: the command executed on the element, and the element-key

A client interested in notifications can subscribe to one or both channels. Notifications basically informs the client that some operation (i.e. command) has been performed on an element, but don't specify the new value of the element.

8.5.3.5. Eviction

When used for caching, Redis provides mechanisms to evict old data. These mechanisms are activated when the threshold related to the maximum size of the in-memory data set is exceeded.

The mechanism, called policies, provided by Redis are:

- Noeviction: when the memory reserved to the data set is exceeded, returns error for each command received by clients;
- Allkeys-lru: evicts Less Recently Used elements to make room for new data;
- Volatile-lru: evicts Less Recently Used elements, among elements that have expiration set;
- Allkeys-random: evicts elements randomly to make room for new data;
- Volatile-random: evicts elements randomly, among elements that have expiration set;
- Volatile-ttl: evicts elements among elements that have expiration set, starting from elements with a shorter time to live.

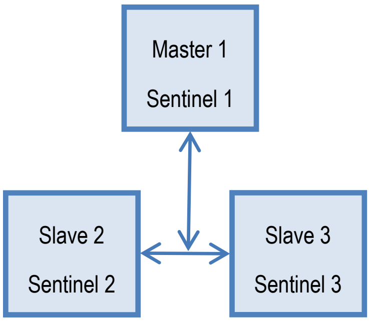
8.5.3.6. High Avilability

High availability is implemented by Redis Sentinel, which controls a group of Redis servers and provides the following capabilities:

- Monitoring: sentinel continuously checks if master and slave instances of Redis servers are working as expected;
- Notification: sentinel notifies one or more recipients (e.g. a system administrator or another computer program) that something is wrong with one of the monitored server;
- Automatic failover: in case of fault, Sentinel promotes one of the slave instance to master and informs the remaining slave instances and the connected clients about the new master instance;
- Configuration provider: sentinel acts a source of authority for service discovery.

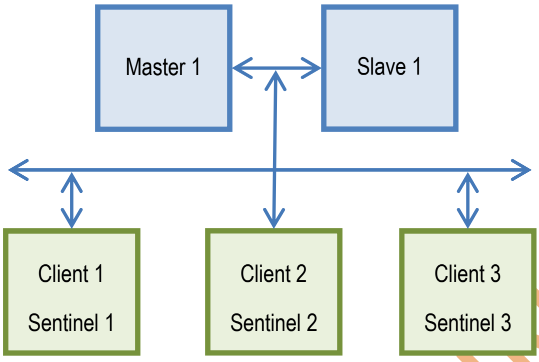
Redis Sentinel, the master instance of Redis server, the slave instances of Redis server, and the connected clients are a distributed system.

Possible topologies for Sentinel are shown in the following figures.

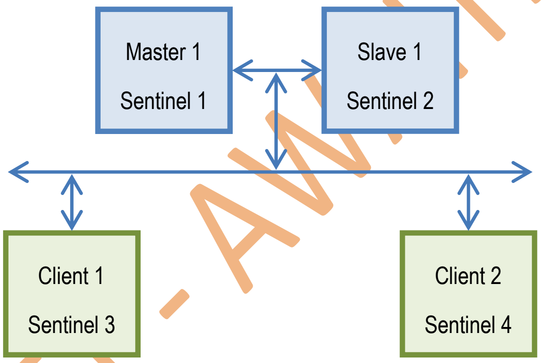


Three nodes (physical computer or virtual machines) running both a Redis server and a Redis Sentinel process.

Preferred topology.



Two nodes hosting Redis servers, one master and one slave, with Redis Sentinel deployed on nodes running clients.



Two nodes hosting Redis servers, one master and one slave, and Redis Sentinel, with additional Redis Sentinel deployed on nodes running clients.

Figure 8.11: Redis Sentinel topologies

8.5.3.7. Automatic Partitioning

Redis Cluster is a distributed version of Redis that scatters elements (key-value pairs) among nodes, enabling horizontal scalability up to 1000 nodes. Redis Cluster design favors performances rather than availability.

Redis Cluster main features are:

- replication and distribution of elements among multiple Redis Cluster nodes;
- continuity of operations in case a subset of the nodes is experiencing failures.

Redis Cluster provides some degree of availability during partitions, providing continuity of operations when some nodes fail or cannot communicate.

8.5.3.8. Extension with loadable modules

Redis provides an API to write loadable modules to add new types of data structures, new commands, or to extend Redis capabilities (e.g. authentication modules, search modules, etc.)

8.5.3.9. LUA scripting

Redis servers embed a LUA interpreter that executes LUA programs (scripts) that bind to stored elements. It is possible to pass parameters at script invocation.

The execution of scripts is not concurrent; scripts are executed one after the other.

8.5.3.10.Programming languages support

At the time of writing, Redis is available for the following languages, or language implementations: ActionScript, Bash, C, C#, C++, Clojure, Common Lisp, Crystal, D, Dart, Delphi, Elixir, Emacs Lisp, Erlang, Fancy, gawk (AWK), GNU Prolog, Go, Haskell, Haxe, Io, Java, Julia, Lasso, Lua, Matlab, mruby (Ruby), Nim, Node.js (JavaScript server side), Objective-C, OCaml, Pascal, Perl, PHP, PL/SQL, Pure Data, Python, R, Racket, Rebol, Ruby, Rust, Scale, Scheme, Smalltalk, Swift, Tcl, MS Visual Basic, VCL.

9. Conclusions

The aim of this report has been to document the path taken to define a scalable and interoperable layer providing the data exchange between internal Rail Operation Services. This includes the integration of the communication between TMS and CCS field infrastructure such as Interlockings, Radio Block Centres and ATO trackside installations. The communication can be extended towards other systems which can utilize exchanged information or providing additional like weather forecast.

Based on the evaluation of requirements of an integrated ICT to link the different Business Services the first important conclusion was that data centric approach is the selected choice. This decision impacts the way of communication and definition of interfaces.

The chosen architecture and the application of a Canonical Data Model allows an easy integration of legacy systems.

As preferable technology for the middleware of such communication infrastructure several products were evaluated and In-Memory Data Grid (IMDG) was selected as most promising technology available (Hazelcast IMDG, RedHat – Infinispan, DDS or others), enabling different implementations depending on client´s choice. The technical evaluation with real systems will be done within X2Rail-2 project.

The important assumption connected with the proposal of data-centric middleware as the backbone of IL is the natural support of publish-subscribe communication pattern. The parallel reading of status indications will enable degraded modes for traffic control and management operations and reduce significant the mandatory handover processes. Important aspects of selected technology are fast machine to machine data exchange, maintaining state of the data with support for a long list of QoS attributes.

A standardized ICT a Canonical Data Model (CDM) has been developed. The Model is scalable and flexible to be extended whenever needed. A CDM is absolute mandatory to ensure the success of the overall Objective to design an integrated ICT for Rail Operations. Based on its importance an overall S2R Data Model has been recommended and is currently under discussion within the S2R Partners.

The discussion around data formats that should be used for data exchange via the integration layer has resulted in the proposal to apply a democratic principle, allowing clients and suppliers to use any format suitable for this type of communication.

The Use cases has shown, that the implementation of an ICT as described in this document will allow new methods of operation as all clients receive simultaneously information of state changes.

10. References

[ActiveMQ Artemis]	https://activemq.apache.org/artemis/docs/2.0.0/
[D7.4]	Definition of the Proof-of-concept, In2Rail 2017
[D7.5]	Evaluation of the proof of concept, In2Rail 2018
[D8.1]	Requirements for the Integration Layer, In2Rail 2016
[D8.4]	Interface Control Document for Integration Layer Interfaces, external/Web interfaces and Dynamic Demand Service, In2Rail 2018
[D8.6]	Description of the Generic Application Framework and its
[D9.1A]	constituents, In2Rail, 2017
[DDS]	Data modelling patterns – Appendix to D9.1
	http://portals.omg.org/dds/
[EIP]	http://www.omg.org/news/whitepapers/Intro_To_DDS.pdf
[EIP Hohpe, Wolf]	http://www.enterpriseintegrationpatterns.com
[EKE]	Gregor Hohpe, Bobby Woolf, Enterprise Integration Patterns, Designing, Building, And Deploying Messaging Solutions, Addison-
[Electronic Design]	Wesley 2003
	https://www.eke-electronics.com/safety-integrity-level-sil-railway-applications
	Stan Schneider „What's the Difference between Message Centric and
[Enterprise Architect]	Data Centric Middleware?" Electronic Design 06 July 2012
[ETCS]	http://www.electronicdesign.com/embedded/whats-difference-
[ETCS Wikipedia]	between-message-centric-and-data-centric-middleware
[EULYNX]	http://www.sparxsystems.com/products/ea/
[infinispan]	http://uic.org/etcs
[JSON]	https://en.wikipedia.org/wiki/European_Train_Control_System
	www.eulynx.eu
[Hazelcast IMDG]	http://infinispan.org/
[Microsoft: Best	http://www.json.org/
Practices –	https://www.w3schools.com/js/js_json_intro.asp
Monitoring and	https://hazelcast.com/products/
diagnostics]	https://docs.microsoft.com/en-us/azure/architecture/best-
[MDA OMG]	practices/monitoring
[MDA – The fast	
guide]	http://www.omg.org/mda/index.htm
	Frank Truyen, The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture (MDA)., Cephas Consulting Corp January 2006
[Publish-Subscribe	http://lsi.ugr.es/~mcapel/docencia/TAMSCT/Cephas_MDA_Fast_Guid
pattern - Wikipedia]	e.pdf
[Protocol Buffers]	https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
	https://developers.google.com/protocol-buffers/docs/overview
[RailEngineer]	https://www.railengineer.uk/2014/05/30/traffic-management-regulation/
[railML]	https://www.railml.org/en/
[Railway Signalling]	http://www.railwaysignalling.eu
[RINF]	www.era.europa.eu/Core-Activities/Interoperability/Pages/RINF.aspx
[RTM]	http://www.railtopomodel.org/en/

[Software maintenance - Wikipedia]	https://en.wikipedia.org/wiki/Software_maintenance
[TAF]	http://www.era.europa.eu/Document-Register/Pages/TAF-TSI.aspx
[TAP]	http://www.era.europa.eu/Document-Register/Pages/TAP-TSI-Technical-Documents.aspx
[UIC]	“Guidelines for the Application of Asset Management in Railway Infratructure Organisations”, UIC, Spetember 2010
[XML]	http://www.w3.org/XML/ https://www.w3schools.com/xml/xml_whatis.asp