Horizon 2020
European Union Funding
for Research & Innovation

In2Rail

# 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 D8.2
# Requirement for Interfaces

| | |
|---|---|
| Due date of deliverable | Month 27 |
| Actual submission date | 31-07-2017 |
| Organisation name of lead contractor for this deliverable | THA |
| Dissemination level | PU |
| Revision | FINAL |

# Authors

| | | Details of contribution |
|---|---|---|
| **Author(s)** | **Thales (THA)**<br>Jean-Yves Friant<br>Jean-Jacques Rodot | Coordination and Document structure of D8.2<br>Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| **Contributor(s)** | **Ansaldo STS (ASTS)**<br>Gian Luigi Zanella<br>Matteo Pinasco | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **AZD Praha s.r.o. (AZD)**<br>Martin Bojda<br>Michal Žemlička<br>Martin Růžička | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **Bombardier Transportation (BT)**<br>Roland Kuhn<br>Martin Karlsson<br>Zbiewniew Dyksy | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **HaCon (HC)**<br>Sandra Kempf<br>Rolf Gooßmann | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **Thales (THA)**<br>Jean-Yves Friant<br>Jean-Jacques Rodot | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **CAF Signalling (CAF)**<br>Carlos Sicre Vara de Rey<br>Manuel Castro Viñas | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |
| | **Siemens (SIE)**<br>Stefan Wegele | Contributions to Sections 1-7 and Appendix 1<br>Review of contributions from other partners |

# Executive Summary

The overall aim of the S2R program/In2Rail (Lighthouse to S2R) is to set the foundation for a resilient, standardized cost-efficient, high capacity System architecture and Data structure for Traffic Management System linked with other Rail Business services.

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

WP8 specifies the Architecture, the data-structure model and Interfaces of the Integration Layer based on the needs of the different internal and external "clients". The availability of various rail asset status data via one common communication layer will improve the efficiency of the decision processes of all linked rail business services to meet their future targets for capacity growth, reliability improvement and cost reduction.

This document provides the high-level requirements for interfaces needed for collaboration of the TMS with other rail business services and the constraints exported from automated communication processes and includes the tools to exchange data in a publish/subscribe pattern to the Interfaces.

D8.1 ("Requirements for the Integration Layer"), D8.5 ("Requirements for the Application Framework"), inputs from ongoing works to specify the Integration Layer (D8.3) and a set of Use cases which are presented in this document form the base of requirements for the Interface requirements that have been determined.

The analysis and the evaluation of a comprehensive set of requirements has been jointly conducted from the partners of WP8.

This document is a key input to the proceeding activities of the X2RAIL-2, WP6 ("Traffic Management Evolution") and IMPACT-2 WP7 ("Integrated Mobility") projects.

# TABLE OF CONTENTS

# Abbreviations and acronyms

** Definition extract from Common Glossary of the* [In2Rail D7.1] *deliverable of In2Rail.*

| Term | Description |
|---|---|
| AMS | Asset Management System |
| AoR | Area of Responsibilty |
| API | Application Programming Interface |
| BPM | Business Process Model (from BPMN but without the indication of notation) |
| BPMN | Business Process Model and Notation |
| CDM | Common Data Model : The static hierarchy of classes that represent data and messages that can be exchanged between TMS modules and applications. |
| COTS | Component Off The Shelf : A (often software) component readily available. The original term means Commercial Off The Shelf, but is often extended (as here) to free software. |
| DDS | Data Distribution Service (Standard from OMG) |
| EU * | European Union |
| ICD | Interface Control Document |
| GUI | Graphic User Interface |
| IM * | Infrastructure Manager: anybody or undertaking that is responsible for establishing and maintaining railway infrastructure. This may also include the management of infrastructure control and safety systems. The functions of the infrastructure manager on a corridor or part of a corridor may be allocated to different bodies or undertakings. |
| I²M * | Intelligent Mobility Management: information developed as a strategically critical asset:<br>• A standardised approach to information management and dispatching systems enabling an integrated Traffic Management System (TMS).<br>• An Information and Communication Technology (ICT) environment supporting all transport operational systems with standardised interfaces and with a plug and play framework for TMS applications.<br>An advanced asset information system with the ability to 'now-cast and fore-cast network asset statuses with the associated uncertainties from heterogeneous data sources. |
| IMDG | In Memory Data Grid: A data structure entirely residing in RAM (Random Access Memory) and distributed within a cluster of computers. The core of the data structure is one of several dictionaries of values accessed from their key. |
| OMG | Object Management Group (American association maintaining various standards including UML, DDS and BPMN) |
| RU * | Railway Undertaking: bodies such as train operating companies and freight operating companies, which are responsible for the operation of passenger and freight trains. |
| TMS * | Traffic Management System: a traffic control-command and supervision/management system, such as ERTMS in the railway sector. |
| UC | Use Case |

| Term | Description |
|---|---|
| UI | User Interface |
| UML | Unified Modelling Language |
| WP7 | Work Package 7: System Engineering of Intelligent Mobility Management (I²M) of In2Rail. |
| WP8 | Work Package 8: Integration Layer of Intelligent Mobility Management (I²M) of In2Rail. |
| Key/Value Pair | Written with capital initials, it corresponds to an object (class instance) in the CDM. By extension it also corresponds to a XPATH like string which corresponds to the initial part of the XPATH like string of an actual object of the CDM. |

# 1 Objectives

The overall objective of WP8 is to specify the architecture, the data-structure model and interfaces of the Integration Layer based on the needs of the different internal and external "clients". The availability of "now" – and "forecasted" status data of various assets involved in rail operations in one communication infrastructure (Integration Layer) available for the different clients will improve the efficiency of the decision processes of all linked rail business services to meet their future targets for capacity growth, reliability improvement and cost reduction.

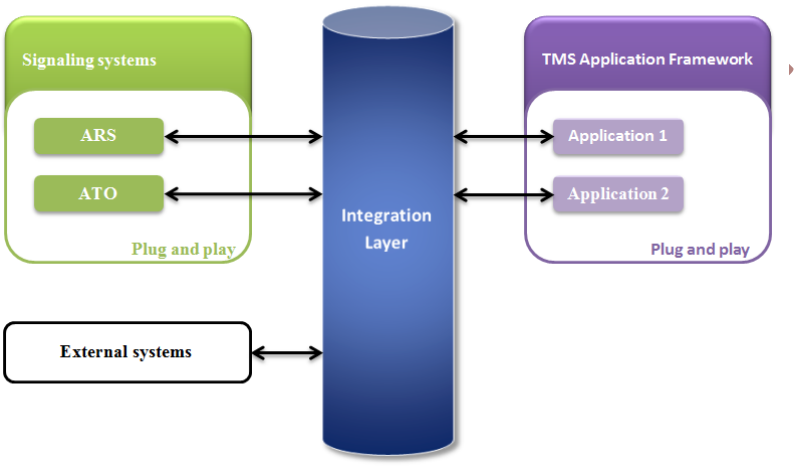


**Figure 3.1: High Level system view**

In2Rail is the lighthouse project to S2R IP2_ X2RAIL-2, WP6 ("Traffic Management Evolution") and S2R_CCA_IMPACT-2, WP7 ("Integrated Mobility") and is the first milestone in the roadmap of the design of a future Traffic Management System which is able to deliver the contributions expected from the S2R program:

- OPEX and CAPEX reduction up to 10% *;
- Reduction of Delay Minutes up to 10% *;
- Increase of Capacity up to 10% *.

(* *compared to a network with same size and complexity of traffic operated under current conditions*)

D8.2 will present the high-level requirements for the Interfaces of the Integration Layer to other business services or functional applications. The main goal is to secure a defined Communication/Data exchange between TMS applications and other multimodal operational systems allowing the seamless exchange and integration of data.

Deliverable 8.2 is the first step towards standardized Interfaces of the Integration Layer and will be followed from proceeding activities in S2R_IP2_ X2RAIL-2, WP6 and S2R_CCA_IMPACT-2, WP7. All results will be delivered to these proceeding projects.

# 2 Background

The present document constitutes the first issue of Deliverable D8.2 “Requirements for Interfaces” in the framework of the Project titled “Innovative Intelligent Rail” (Project Acronym: In2Rail; Grant Agreement No 635900).

The overall aim of the S2R program/In2Rail (Lighthouse to S2R) is to set the foundation for a resilient, standardized cost-efficient, high capacity system architecture and data structure for a future Traffic Management System which is linked with other Rail Business services through a single communication infrastructure.

WP8 together with activities in WP6, WP7, WP9 and WP10 fulfil the criteria to be a “Cross-Cutting Activity” hence the I²M sub-project of In2Rail is the Lighthouse project to CCA WA4.2 “Integrated Mobility which integrates elements of S2R_IP2_TD9 with S2R_IP3 and S2R_IP5.

Works proposed for WP8 have been extracted from the S2R_IP2_TD9 “Traffic Management Evolution” scope with the focus on the generic design parameters of the Integration Layer and a “standardized” framework for applications, which allows for Plug-and-Play installation of SW modules.

The proposed activities in WP8 Task 1 cover the evaluation and description of the overall system architecture and data structure of the Integration Layer (D8.3), the interfaces to other business services (D8.2) and to external clients via WEB-IF (D8.4)

D8.2 “Requirements for Interfaces” represents a key part in the concept of design of the system architecture of the Integration Layer.

In2Rail WP7 Task 3 (“Proof of Concept”) has been originally part of WP8 but has been assigned finally to WP7, but will use D8.2 as input for their design of the “Prototype-Interfaces”.

WP7 Task1 “Requirement Analysis” focussed on integrating all the state of the art requirements from recent and past customer and research projects for Traffic Management as further input to WP8 and applicable requirements have been integrated into D8.2.

D8.1 (“Requirements for the Integration Layer”), D8.5 (“Requirements for the Application Framework”) and the current state of the design process for D8.3 (Description of the Integration Layer” are inputs to the definition of the Interface requirements.

# 3 Boundaries

API Interfaces to integrate TMS modules into a framework are not in the scope of this document and will be presented in D8.7 “Interface Control Document (ICD) for Application-specific Interfaces”.

D8.4 “Interface Control Document for Integration Layer Interfaces, external/Web interfaces and Dynamic Demand Service” will define the Interface to external clients via WEB including for general Data exchange management and the support of “Dynamic Demand Service” messages from external clients to Rail services via the same Interface.

The evaluation and presentation of requirements for Interfaces between Integration Layer and Command and Control System (CCS) in the field (Interlocking, ATO Trackside, RBC and other field elements) are not in the scope of WP8 and will be specified in the proceeding project S2R_IP2_X2RAIL-2 in WP6.

# 4 Methodology of Works

## 4.1 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 D8.2 Requirement for Interfaces was coordinated by Thales.

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

All partners provided assigned input and conducted reviews to all chapters of the document.

Telephone conferences and periodic progress meetings are held to align the inputs and to clarify difficult technical topics.

## 4.2 Presentation of Requirements for Interfaces of the Integration Layer

For the description of the requirements of the Interfaces following approach has been chosen:

- overview of the different interfaces described in this document (Chapter 5);
- Interaction of different Interfaces with the Integration Layer applying rules and principles of a Canonical Data Model (Chapter 6);
- Low level Interface Requirements presented in a synthetic table format (Appendix 1).

# 5 Definition of Interfaces

## 5.1 Overview

Figure 5.1 shows the interfaces that are specified by the In2Rail platform, the different components that must be provided by the In2Rail platform, and examples of modules and applications connected to the platform.

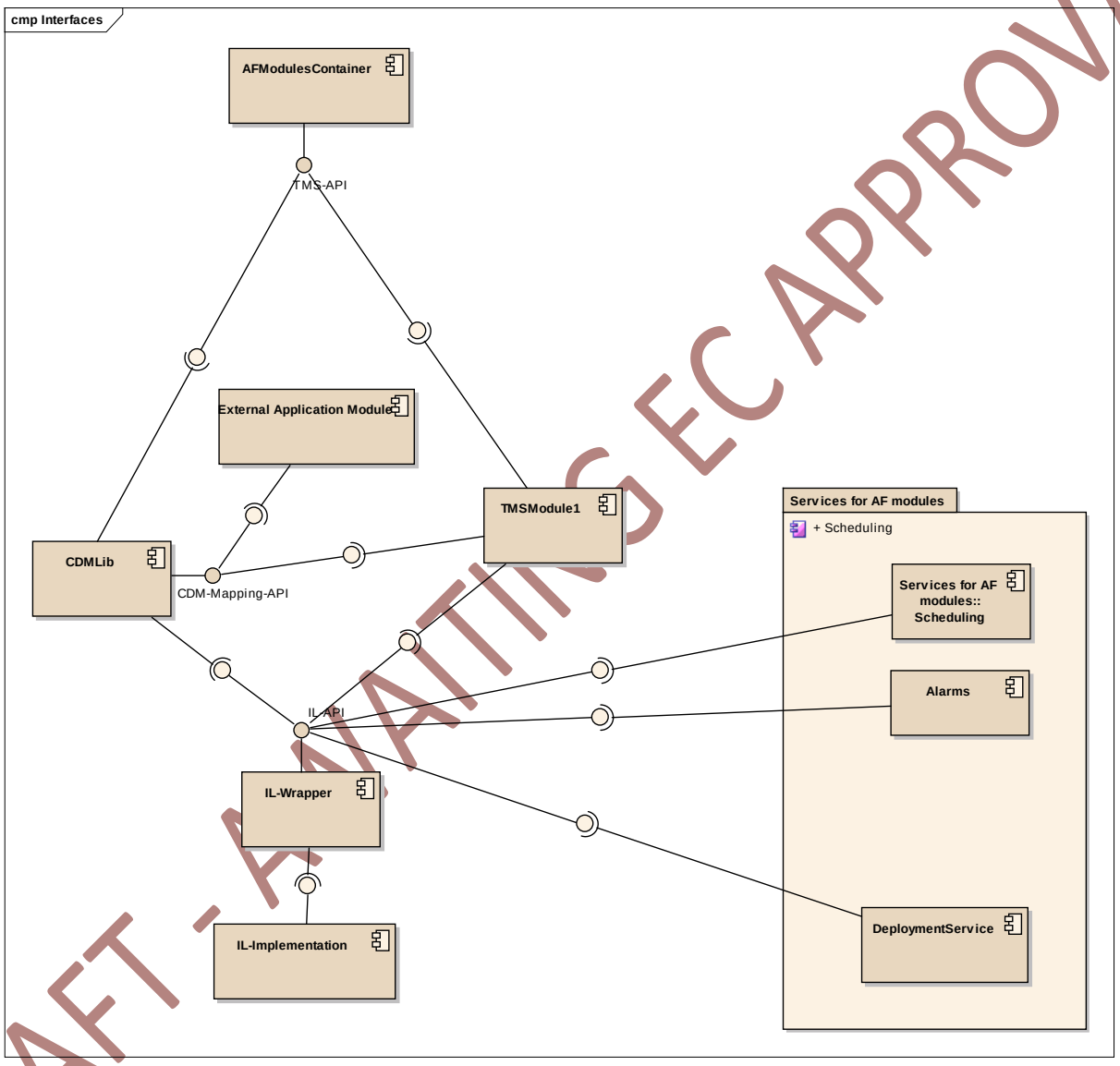


**Figure 5.1: Interfaces of the In2Rail platform**

## 5.2 Interfaces of the Integration Layer for Business Services and Applications

IL-API is the lowest level interface standardised by the In2Rail platform to access data in the Data Warehouse.

CDM-Mapping-API is a high-level interface to modules, external applications and business services to provide access to data which are available on the Integration Layer following the rules and principles of the Common Data Model (structures, attributes, unions, etc).

TMS-API is an interface used by internal modules of the In2Rail platform.

## 5.3 Services, Implementations and Applications

Services of the AF module uses the IL-API (or if it is possible the CDM-Mapping-API) to communicate with other modules. All services apply read/write/publish/subscribe processes.

IL-Implementation is the implementation of the data warehouse in a specific In2Rail platform deployment (COTS). There are no requirements on it, except that it should be possible to develop a IL-Wrapper for this implementation.

IL-Wrapper is a plug-in to a specific IL-Implementation needed for the IL-API.

CDMLib is a component of In2Rail platform, which provides the CDM-Mapping-API based on IL-API and the definition of the CDM. Its purposes are to serialise/deserialise data from an object/attributes form to a binary encoded structure. This serialisation/deserialisation process should be based on code generation or CDM description file dynamic inspection.

AFModuleContainer implements the TMS-API interface to provide the life cycle services to internal modules of the In2Rail platform.

Components of the Services for AF modules are implementations of the different services of the In2Rail platform.

TMSModule1 is one example of TMS module hosted in the In2Rail platform.

External Application Module is an example of an external application connected to the In2Rail platform (for example, an AMS application).

# 6 Interfaces in a Canonical Data Model

## 6.1 Canonical Data Model (CDM)

A standardized Data Structure enables different services to share data in a publish/subscribe pattern, thereby making full use of all available information. In addition, the various applications with their specific data coexisting in the Integration Layer force a common language - the Canonical Data Model (CDM).

There are several standardised data models already existing in the railway sector – RINF, TAF/TAP TSI and ETCS language to name a few. What most of these have in common is that they were designed to solve a specific problem. RINF describes the infrastructure, TAF/TAP TSIs are for data exchange between RU and IM, ETCS defines data needed by an on-board ATP. The result is that they have quite different views of the same reality. For example, all three models need to refer to geographic positions in the railway topology. But they do it in different ways, making them incompatible with each other. Identifying every position as a distance from a balise group is very practical for the ATP, but useless as a basis for an infrastructure register.

When defining the In2Rail/S2R CDM, the goal was to take a different approach. Start by a structured break down of the entities of the railway system (including physical entities like wayside assets and vehicles as well as logical entities, like time tables and traffic restrictions). Then add to each type of entity whatever data attributes that may be of interest to any application in the overall system context.

In this way, the data model will be structured according to real world objects and relations, instead of to the needs of the foreseen services. Based on the CDM, data provider services know where to place their output, and data consumers know where to find it.

Providers and consumers become loosely coupled, as they do not need any knowledge of each other's internal data representation. This simplifies maintaining and upgrading applications independently. One application may "think" in RINF and another in ETCS, but over the Integration layer, they both "speak" CDM.

The following chapters describe how the different types of interfaces are implemented/connected to the CDM.

## 6.2 Process to define a Pattern

Use cases are selected and analysed for possible constraints. Then patterns are defined which covers these constraints. The defined patterns will be used to validate the requirements.

*Note: Requirements reflect constraints in a formalized way*

## 6.3 Requirements for the Interfaces imposed from CDM

### 6.3.1 Low-level interface IL-API and CDM

| Requirements # | Description |
|---|---|
| REQ_IRS_1 | The In2Rail Platform permits applications and modules to access (read or write) individual Key/Value Pair content in their binary serialised form. The lowest level interface usable by module or application to access Key/Value Pair content is the IL-API. |
| REQ_IRS_2 | The In2Rail Platform allows applications and modules to create and delete individual Key/Value Pair in the data grid. Type (class in CDM) must be specified when a Key/Value Pair is created. |
| REQ_IRS_3 | The binary serialised form of any Key/Value Pair contains the serialised data of a complete Key/Value Pair including (recursively) all subparts, with the special case that accessible Key/Value Pairs which are part of the Key/Value Pair are serialised as an "address" which allows to retrieve the actual content of the accessible Key/Value Pair. |
| REQ_IRS_4 | The "address" (UUID or XPATH like string) of an accessible Key/Value Pair B which is included in a Key/Value Pair A, should be stable in time at least during the lifetime of both A and B. This means that as long as A and B both exist (none of them has been deleted), then the address's (UUID or XPATH like string) validity is guaranteed. |
| REQ_IRS_5 | The IL-API is transparent to the serialised data of Key/Value Pair content. This means that in no treatment, the implementation of IL-API need to deserialise the Key/Value Pair content. |
| REQ_IRS_6 | The binary serialisation method (i.e. mapping between the CDM and serialised data) should be the same for all type of Key/Value Pairs of the CDM and for instances of Key/Value Pairs, within a single deployment of the In2Rail Platform. This requirement does not need to be respected when the serialisation is changed; in that case, and during a non-production time, all persistent data can be de-serialised and re-serialised using a different method, and during that process, the serialisation method is not uniform. |
| REQ_IRS_7 | Two different implementations of the In2Rail Platform can use different serialisation methods. Bridges are then used to connect these In2Rail PLATFORM implementations. |
| REQ_IRS_8 | Serialisation method allows to represent at least the following basic data types: Variable length string, 8 bits signed integer, 32 bits signed integer, 128 UUID, Boolean, Simple (32 bits) and double (64 bits) precision floating point (IEEE 754). The following basic types, unsigned integer 8 bits, 32 bits and 64 bits can be used in the CDM to indicate that the negative values are not authorised, but the MSB can never be set. This means that they are serialised as their signed counterparts |
| *Note: the subset of basic data types above is restricted to corresponds to what exists in all target languages, especially Java which does not admit unsigned integers)* | |
| REQ_IRS_9 | Serialisation method allows to serialise enumerated values. |
| REQ_IRS_10 | Serialisation method allows to serialise structures (sequence of named values of any serializable data type, which are called attributes). |

| Requirements # | Description |
|---|---|
| REQ_IRS_11 | Serialisation method allows to serialise unions (one value chosen among several serializable data type). The chosen data type of each value is part of the union serialisation, it does not require a separate attribute. |
| REQ_IRS_12 | Serialisation method allows to serialise sequence of data type (serialisation of arrays). The serialisation method should handle in the sequence the number of elements (not necessarily using a counter, stop markers or other methods are possible too). |
| REQ_IRS_13 | Serialisation method should allow the presence of non-valued data (example, attribute not valued in a structure, or element not valued in a sequence, but present to maintain indexes value). |
| REQ_IRS_14 | Serialisation method should allow structure extension mechanism, at least through the non-valued data attribute representation. |
| REQ_IRS_15 | The In2Rail Platform allows modules to be notified of values changes (change of the serialised data) of given Key/Value Pairs. Monitored Key/Value Pairs can be specified individually or by filter (regular expression on the key). |
| REQ_IRS_16 | The In2Rail Platform allows modules to be notified of apparition and suppression of Key/Value Pairs. The module indicates which are the Key/Value Pair of interest either by the type (class in CDM) or by a regular expression applied to the path of Key/Value Pair or by a combination of both. |

**Table 6.1: Low-level interface IL-API and CDM**

### 6.3.2 High-level interface (CDM-Mapping-API)

| Requirements # | Description |
|---|---|
| REQ_IRS_17 | For the part which is dependent on the CDM, the implementation of the CDM-Mapping-API should be generated (using a code generator) from the file defining the CDM. |
| REQ_IRS_18 | For each of the following language, C, C++, Java, C#, if a module of AF or IL is using the CDM-Mapping-API, then the implementation must be generated (using a code generator) from the XML file defining the CDM. Writing implementation by hand of data structures defined by the CDM is forbidden for these languages. |
| *Note: This should be automatically the case for C++ and Java, if protobuf serialisation is chosen, as these languages have a generator in the default implementation.* | |
| REQ_IRS_19 | For each of the following languages, C, C++, Java, C#, the representation of structures, enumeration, attributes of structures and their presence, sequences, unions, basics data types and reference to other Key/Value Pairs should be the same (but it can be different for two different languages) for all implementation of the In2Rail Platform. The representation should be specified by the document D8.4. |

*Note: Of course, the different representations for different languages concern the High Level API, note the binary (serialised) content, which should ensure possibility of communication between modules/application.*

***Table 6.2: High-level interface(CDM-Mapping-API)***

## 6.4 Command Pattern

The following use cases are (arbitrarily) chosen to evaluate the pattern for commands:

- one TMS module sends a command to another module (internal or external) for immediate application;
- an external application (AMS) requests computation of forecasted data (number of forecasted switch movements);
- an operator, through the external application of operator workstation, requests to insert a temporary restriction.

### 6.4.1 Internal Command

This is the simplest case of a command. One module (module A) is sending a command and waits for the command to complete. Another module (module B) sees that there is a command to execute which is its responsibility, it executes the command, and stores the result. Module A fetches the result, and once everything is finished, it cleans the data store.

### 6.4.2 Request of a Forecast

Differences to Use Case 8.4.1:

- Module A is an external application;
- The data should be kept for “some reasonable time”;
- There could be a special authorisation from an external application attached to the request.

Defined process:

- Module A searches available forecast data for switch movements for a given date, browsing the available data;
- Module A reads the available data (maybe there are rights and price attached to the reading of already available data) or requests the computation of new data;
- Module B will compute the data, and make it available. These data are associated with some expiry date. When the expiry date is reached, the data can be discarded (by the data store or by IMDG wrapper);
- Module A reads the result. At some point, requests for computation or end of computation or reading the result can be applied.

*Note: “Querying available data” (1st sub-case) and “Reading available data” (2nd sub-case) are not directly related to the pattern for commands.*

### 6.4.3 Temporary Restriction inserted by Operator

This use-case is a of the rapid reaction type, if there is no simulation of the impact in some sandbox before applying the restriction. The operator requests an immediate insertion of a restriction with minimal impact analysis. This use case can be decomposed with the following high level steps:

- An operator places a command for inserting a temporary restriction from its GUI (external application);
- Operator rights are checked (he has the rights in this use case) for this command;
- The result of the command is logged as operator action with some execution status
- The GUI application fetches the result of the command and presents it to the operator.

What this use case demonstrates, is that rights are not attached only to applications but also to operators and that commands are potentially logged differently.

### 6.4.4 Command Pattern

Each sub node (Key/Value Pair) of the Network root should have its own Key/Value Pair for commands. In this Key/Value Pair, there can be groups of commands. At this level, the different groups of commands are just logical groupings. Groups can be nested. In a group, there can be inner groups or a command directory. A command directory represents the full semantics of a command and contains all instances of this command.

In a command instance, there are several Key/Value Pairs:

- Key/Value Pair for parameters;
- Key/Value Pair for result;
- Key/Value Pair for execution status;
- Key/Value Pair for abortion request;
- Key/Value Pair for sandbox where command can be prepared (can be directly the sandbox, or a reference to a sandbox being elsewhere in the data store).

*Note: A sandbox has generally the meaning of a partially isolated environment where the actions carried inside the sandbox have limited and controlled effect on the outside of the sandbox. For example, a sandboxed environment can be used to test a future release, while being isolated from the actual production environment. In this document, the term sandbox is used for a part of the data store which is isolated from the actual production environment. Key/Value pairs have the same structure in the sandbox and in the actual production data store, and sandboxes have each their own current time. Sandboxes can be used to prepare commands to be applied "immediately" (their current time is now) or to prepare/evaluate commands to be applied sometimes in the future (their current time is the planned time of application). When the sandbox is "applied", it should be completely applied (there is no partial application concept, or human interaction in the middle of the sandbox application).*

The Key/Value Pairs described above can be represented by the paths as shown in Table 6.3.

| Path | Content |
|---|---|
| /network/Infrastructure/cmd | Path for all commands concerning infrastructure. This path is NOT a key with value. |
| /network/Infrastructure/cmd/tempRestriction | Path for all commands concerning temporary restrictions. This path is NOT a key with value. |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley | Path for all the instances of commands to immediately put a temporary restriction. This path is NOT a key with value. |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid> | Path for one instance of command |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid>/parameters | Key/value for the parameters of one instance of command |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid>/result | Key/value for the result (or reference to the value?) Having the actual value here allows keeping the data until they expire easily. |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid>/status | Key/value for the status of the command. Value could at least be “preparation”, “submitted”, “running”, “completed”, “failed”, “aborted” |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid>/abortionRequest | This key/value is created by requester when it wants to cancel the command |
| /network/Infrastructure/cmd/tempRestriction/placeRestrictionImmediateley/<uuid>/sandox | This can be a path root for the sandbox or a key/value with the uuid of a sandbox located elsewhere. |

**Table 6.3: Paths representing Key/Value Pairs**

*Note: A pattern can be represented by a dedicated application/service. This service can also subscribe to the issued commands and can update the status of information.*

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_20 | The In2Rail Platform permits applications and modules to issue commands to other applications and modules without prior knowledge of the module or application that will handle the command. The lowest level interface usable by module or application to access commands is the IL-API. |
| REQ_IRS_21 | There can be several instances of the same command running at the same time in the system. The interface gives access each of these instances individually. |
| REQ_IRS_22 | The IL-API allows querying all instances of a given command (type). |
| REQ_IRS_23 | The IL-API allows creating new instance of command. A new command is by default in “preparation” state. |

| Requirements # | Description |
|---|---|
| REQ_IRS_24 | The status of each instance of command can be queried (and set) individually. Status includes at least the following values: Preparation, Submitted, Running, Completed, Failed and Aborted. |
| REQ_IRS_25 | For each instance of command, CDM allows storing parameters attached to a key, separate from the command's other keys. The parameters of each type of command are defined by the Canonical Data Model. |
| REQ_IRS_26 | The result of a command can be read/write independently (separate key/value) from the rest of the command. |
| REQ_IRS_27 | The result of a command is given an expiration date. The In2Rail platform keeps the result in the data store at least until the expiration date is reached or some module (issuer of the command) explicitly removes the value of the result. After expiration date, the In2Rail Platform should remove the expired result (and command) to limit memory usage. |
| REQ_IRS_28 | The requester of a command can also request to cancel the command (abortion request). The module executing the command is NOT forced to take abortion request into account. |
| REQ_IRS_29 | A sandbox is by default attached to each command instance. The sandbox contains data which can be required to run a BPM triggered command and scenario study in the frame of the execution of the command. The sandbox should be removed when the command is complete (Completed, Failed, or Aborted). |
| REQ_IRS_30 | For each command (type), access rights can be defined. Only module and or Persons having the right access can issue the command. |
| *Note: Access rights are configured via Topics.* | |
| REQ_IRS_31 | When a module or person does not have the right access tries to issue a command, the denied access is logged and the command is not issued. |

**Table 6.4: Command Pattern - Requirements on the low-level interface**

## 6.5 Triggers for Workflow (BPM) and associated Sandbox

The workflow represents an oriented graph consisting of connected business activities and their results. Figure 6.1 represents a workflow by means of a Petrinet in accordance with ISO/IEC 15909-2:2011.

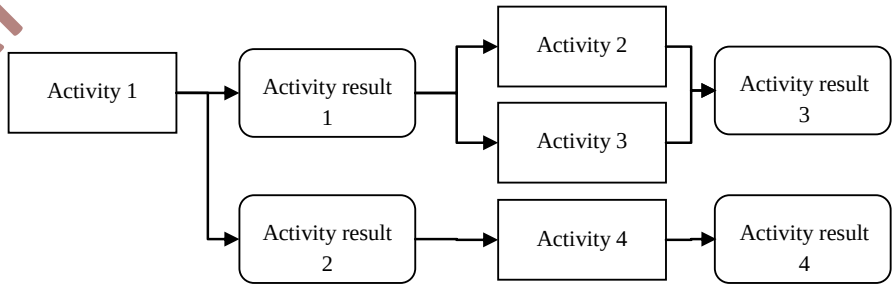


**Figure 6.1: Example of a workflow specification**

The sequence of an activity implemented depends on the result of the previous one and can be concurrent (Activity 4 and Activity 2) or alternative (Activity 2 and Activity 3). Typically, workflows involve cooperation of several persons on one task representing different steps of modifications. The intermediate results of such modifications are not intended to be published to the neighbour systems and sandboxes model is used to manage this part of the workflow.

Following use-cases have been selected:

- Management of very short term request;
- Management of updates of train information provided by RU;
- Negotiations about train rerouting/cancellation due to major disturbances with RU;
- Concurrent modifications of the timetable with side effects to the neighbour areas.

### 6.5.1 Short Term Requests

A railway undertaking (RU) can require modifications of the current production timetable:

- Train path cancellation total or partially;
- Change of the train route (e.g. to join additional wagons/passenger cars);
- Change the schedule of a train route;
- Change planned connections between trains (e.g. to enable crew change);
- Request for a new trip.

The RU requests new slots to operate trains on the network. The impact on the actual production timetable situation is analysed by an operator in a sandbox. The operator can apply modifications in several cycles to the request of RU to evaluate the best result.

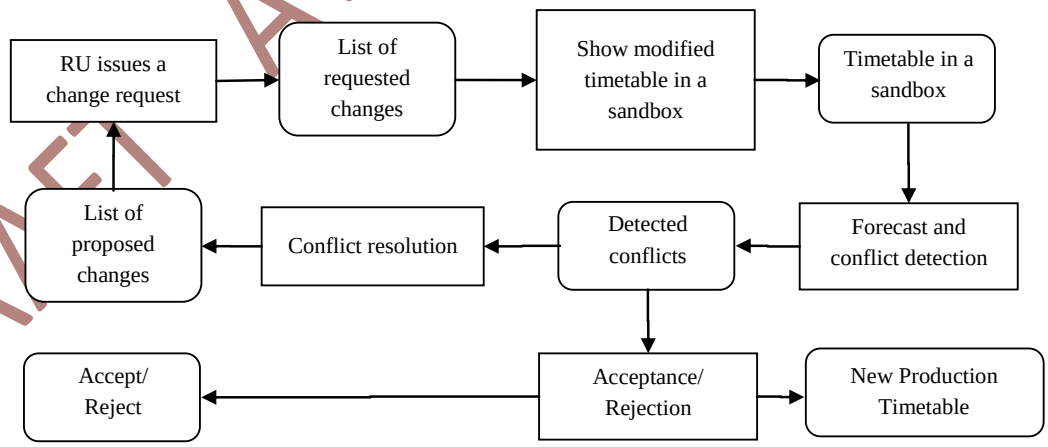


**Figure 6.2: Workflow overview for short term requests from RU**

### 6.5.2 Modification of Production Timetable

Typically, the country wide railway network is divided into areas of responsibility (AoR) assigned to different operators. If one operator starts modification of trips inside of his area, the consequences inevitably arrive at neighbouring areas, causing potential conflicts as entrance times of the trains into the neighbour area and even their sequences may change.

Depending on the business processes of Infrastructure Managers (IM) the cooperation between operators can be organized such that:

- Areas of responsibilities are virtually isolated – the neighbour modifications are explicitly shown and require explicit application in the local AoR or;
- any modification of the timetable is done in a sandbox. Before the sandbox is accepted, all the involved neighbours must acknowledge it.

Despite the different processes to manage changes of the production timetable "very short term requests" shown in Figure 6.2 seem to be very similar.

- A Railway Untertaking (RU) requests a modification to the production timetable a "change set" is prepared by an operator involving his AoR and;
- workflows are needed which allow "sandboxes" to be exchanged between involved persons/systems.

This use case demonstrates that the CDM has to support efficient configurations of a workflow/process and the monitoring of the workflow/process state by observing timings of activities.

The table below shows the pattern needed to manage the necessary data involved in the process. The implementation/location/path of the data will follow the rules and principles of the CDM.

| Path | Content |
|---|---|
| /network/sandboxes | Path containing sandboxes. They can be used without involvement in workflows for e.g. for what-if-analysis. |
| /network/requests | Path containing further details about requests |
| /network/requests/issued | Path containing a key-value pair representing some object from Canonical model. |
| /network/ requests/workflow | Path containing transitions in the workflow (which activity when started and finished) as key-value-pairs. |

*Note: The involved key-value-pairs shall be removed from the IL during a specified time interval starting after the process/workflow is finished*

**Table 6.5: Pattern needed to manage the necessary data involved in the process**

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_32 | The In2Rail Platform permits to applications and modules to issue change requests to other applications and modules without prior knowledge of the module or application that will handle the change request. |
| REQ_IRS_33 | There can be several instances of the workflow running at the same time in the system. The interface gives access each of these instances individually. |
| REQ_IRS_34 | The CDM allows access to the current state of all workflows. |
| REQ_IRS_35 | The finished workflow is given an expiry date. The AL/IF keeps the workflow intermediate states at least until the expiry date is reached or some module (issuer of the command) explicitly removes the value of the result. After the expiry date, the In2Rail Platform should remove the expired data to limit memory usage. |
| REQ_IRS_36 | The requester of a workflow can also request to cancel it (abortion request). The module executing the command is NOT forced to take abortion request into account. |
| REQ_IRS_37 | A sandbox can be attached to different steps of the workflow either by reference or by value. The sandbox should be removed when the workflow is finished (Completed, Failed, or Aborted). |
| REQ_IRS_38 | For each workflow type, access rights can be defined. Only applications having the right access can issue read the state or contribute to the workflow. |

**Table 6.6: Modification of Production Timetable - Requirements on the low-level interface**

## 6.6 Access Rights

Under the access rights we define the right to read and to write to the data managed by the Integration Layer. The Integration Layer shall manage the data in the form of key-value pairs, where a value represents one object encoded by a serialisation method. The object itself can represent a tree of further child-objects according to a composition relationship. As the Integration Layer does not provide any validation support for the value-content and act as a "black-box", it is not possible to ensure the access rights to the single attributes inside of the object in a value. The access right granularity in the Integration Layer is the value part in key-value. The values are managed inside of Topics, where only specific types of object can be published/subscribed. Therefore, the most appropriate point to define access rights is at the level of a the Topic: each Application can be configured with read and/or write rights to some Topics.

### 6.6.1 Protection against disturbing applications

As soon as a TMS is configured with its business applications, the availability of the vital functions can be violated by modules starting publishing on “wrong” Topics due to:

- bugs in the software;
- message errors;
- message does not correspond to object types specified for the Topic.

### 6.6.2 Protection against malicious applications

Inside of the TMS some commercially sensitive data are located which need to be protected against malicious applications. To define the requirements following selection of “critical” data has been chosen:

- Timetables and train consists of competing RU;
- Diagnostic data for the field equipment from different signalling system suppliers;
- Intermediate results of the root cause analysis for delay.

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_39 | The In2Rail Platform permits to applications and modules to subscribe to only configured Topics |
| REQ_IRS_40 | The In2Rail Platform permits to applications and modules to publish to only configured Topics |
| REQ_IRS_41 | The In2Rail Platform permits to applications and modules to open their own private Topics and configure access rights for other applications. |
| REQ_IRS_42 | The In2Rail Platform support access rights for stream oriented and multicast protocols. |
| REQ_IRS_43 | The In2Rail Platform uses one combined API for getting access to the data on IL and notifies the client application about trials of access violation. |

**Table 6.7: Protection against malicious applications - Requirements on the low-level interface**

*Note: The In2Rail Platform will have to log all failed access trials into a central repository for debugging purposes.*

## 6.7 Historical Data

Under history we define past values for key-value pairs managed in a Topic.

The main objective of this functionality is to separate the life cycle of the information between producers of information and subscribers.

The Topic is configured to indicate the amount/number of available historical patterns embedded and new subscribers can read these historical data.

### 6.7.1 Train Position Report

Most of the subscribers are interested in the history of train position e.g. for calculation of the forecast or triggering announcements in passenger information system or to create the "Time Distance Diagrams"

### 6.7.2 Pattern for Train Position Report

The API provided by the middleware can send the historical Data to the newly connected client using the same API with "historical" timestamps in fast motion. The high level differences of some available COTS products in terms of implementation of messaging of Train Position Reports can be characterized as follows:

- DDS supports this feature directly;
- Hazelcast supports the lifetime of a key-value in maps. As an alternative, it provides a RingBuffer structure, which can be configured with some size, which is the same as history depth;
- Redis supports lists, so that the writer could keep the size of a list according to the history depth. Redis provides an index-based access pattern to the list, so the client could "get" only the last N values it is interested in.

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_44 | A Topic shall be configurable with the number of historical samples (History Depth). This number can be used to limit resource consumption on the IL. |
| REQ_IRS_45 | A subscriber shall be able to configure required History Depth, which can be smaller than the Topic configuration. |
| REQ_IRS_46 | The handling of the history shall not increase the API complexity to IL. |
| REQ_IRS_47 | The IL shall support at least one historical Topic per cluster. |

**Table 6.8: Train Position Report - Requirements on the low-level interface**

## 6.8 Traffic Status Fore-cast

The following use cases were (arbitrarily) chosen to evaluate the pattern for commands:

- One TMS module uses the continuously computed future state going from now to the next 12 hours of each train positions, arrival times and other data used in the Traffic Control process;
- An external application (AMS) request computation of forecasted data (number of forecasted switch movements);
- An operator analyses the impact of inserting a temporary restriction.

### 6.8.1 Forecasted Train Position (near future)

This use case corresponds to a TMS application module which needs continuously the fore-casted position and delays of all trains for the near future to compute some global indicator (average delay, average time of trains stopped out of station).

The forecasted train position is supposed to be always available. TMS applications use this information together with actual and past position of trains to compute the fore-cast status.

That status information is used to update other applications.

Other types of objects/rail assets have a less continuous update model. For example, switches move a few times an hour and use value change principles.

The "Client" can subscribe permanently to the topic representing the status of the object/train of interest or subscribe only periodically to obtain the required data.

At each computation, the fore-cast data is updated and subscribed clients are notified and updated.

### 6.8.2 Temporary Restriction

In this use case, an operator analyses the impact of a temporary restriction that would be activated in two or three weeks. The operator creates a "scenario study" in a sandbox, set the simulated time of the sandbox and then applies the restriction with the specified date in the sandbox scenario. The operator continues then running some global command (e. g. conflict detection) in his sandbox and displays the result. During the global command execution, the sandbox is enriched with the computation results. For a scenario sandbox, the result is kept alive as long as the sandbox exists. The date/time, for which the simulation is supposed to create a forecast scenario is set by the operator. All existing data is available for the simulation.

### 6.8.3 Pattern Traffic Status Forecast

To describe the pattern, this text uses the object referencing principles described in current work on Canonical Data Model. It may need updating when the CDM is finalised.

From the above use cases, we can identify the following families of future status representation:

- Continuous representation of future values: All values related to present time up to the end of the specified "future" are continuously computed and available without any period missing;

  *Note: The end of this "future" will be called present horizon or simply horizon if the context does not require distinguishing the present horizon from a future horizon.*

- Event based representation of changes between present time and horizon: this event based representation includes events that will arrives only once between now and horizon;
- Potential sandboxes whose current time correspond to present time, with event based and/or continuous representation until horizon;
- Commands requesting computation of some future value, this is sub case of command pattern, potentially using a sandbox either around present or some moment in future time;
- Sandboxes whose current time is corresponding to some moment in future time (they have a simulated current time). These sandboxes have their own future horizon, with event based and continuous future values.

To illustrate the possible representation of these concepts with key/values pairs, the text and table below uses the example of train position for continuous values and switch position for event based values.

| **Path** | **Content** |
|---|---|
| /network/TimeTable/Trains/Train/uuid/Position | Key/value for the value of the actual (most recent update) of a train position |
| /network/TimeTable/Trains/Train/uuid/Position/Futures | path for all future positions of this train. This path is NOT a key with value. |
| /network/TimeTable/Trains/Train/uuid/Position/Futures/now-some-time-in-future | Key/value for the list of sampled positions of this train between now and some-time-in-future (example now-22-10-2018-10:15:00). The sampling time is some configuration data of the Position type. |
| /network/TimeTable/Trains/Train/uuid/Position/Futures/time1-time2 | Key/value for the list of sampled positions of this train between two times in the future (example 22-10-2018-10:15:00-22-10-2018-10:30:00). The sampling time is some configuration data of the Position type. This slicing is used to limit the size of values as the now-some-time-in-future must be updated very often for numbers of trains |
| /network/Infrastructure/Operation/Track/Condition/Switches/uuid | Key/value pair for the present state of a switch |
| /network/Infrastructure/Operation/Track/Condition/Switches/uuid/futures | Path for all the events forecasted between now and horizon for this switch (uuid). This path is NOT a key with value. |
| /network/Infrastructure/Operation/Track/Condition/Switches/uuid/futures/time-in-future | Key/value for the forecasted state change of this switch between present and horizon. Time-in-future is a time representation like 22-10-2018-10:12:17 and corresponds to the forecasted time when the event (state change) will occur. |
| /network/sandboxes/uuid/CurrentTime | Key/value for the current time to be used by a sandbox whose current time is not corresponding to present time |

| Path | Content |
|---|---|
| /network/sandboxes/uuid/TimeTable/Trains/Train/uuid/Position/Futures/now-some-time-in-future | Key/value for the list of sampled positions of this train between now and some-time-in-future (example now-22-10-2035-10:15:00) in a studying scenario around /network/sandboxes/uuid/CurrentTime. The sampling time is some configuration data of the Position type. |

**Table 6.9: Example of train position for continuous values and switch position for event based values**

*Note: The proposed keys above do not mean that reading or subscribing to these keys is triggering the computation. It just indicates the used key for storing data. If corresponding future data are published, the key should be as indicated, so that subscribers are aware of the key they should subscribe in a regular way for all implementations. The Key/Value Pair for future are computed normally based on BPM triggers (some may be periodic) or commands.*

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_48 | In2Rail Platform permits to applications and modules to store and query values and events between present time and a moving end of near future called horizon. For each type of Key/Value Pair, the canonical data model determines if the values are continuous or event typed. |
| REQ_IRS_49 | For continuously forecasted value, configuration of In2Rail Platform gives for each type of data the sample period (time elapsed between two contiguous values) and the slice size (the maximum number of samples in the same key/value pair for this data) |
| REQ_IRS_50 | Extent of horizon should be configurable, either statically or dynamically (at runtime). Dynamic configurability of horizon is not mandatory. |
| REQ_IRS_51 | For continuously forecasted value, the slices have fixed boundaries, except the slice which starts at present time (now). Therefore, the first slice, starting at present time, has a variable number of samples. Other slices have fixed number of samples (for the same type of data). Modules can read/subscribe to/write whole slices but not part of slices using the IL-API. |
| REQ_IRS_52 | For event typed forecasted values, only change of values can be read/subscribed to/written, and all events based forecasted value is attached to a time when the value change is forecasted. |
| REQ_IRS_53 | Continuous and event typed forecasted values can also exist in the sandboxes whose current time is corresponding to present time. Nevertheless, some command need to be issued to trigger their computation and therefore their apparition in the data store. |
| REQ_IRS_54 | It is possible to set the current time of a sandboxes to some moment in the future. Their horizon is then moved accordingly. When attaching a new moment to a sandbox, then all now-some-time future should be invalidated/removed. This is not the responsibility of the In2Rail platform to do so. It is the responsibility of the module which changes the sandbox current time. |

**Table 6.10: Traffic Status Forecast - Requirements on the low-level interface**

## 6.9 Event Management

Event logging is a central support function in the system, which has several purposes:

- Alerting users of significant technical and operational events;
- Record device behaviours over time, for statistical purposes;
- Aid in posterior analysis of incidents and unexpected situations;
- Trace application execution, for debugging purposes.

This chapter discusses the specific action of logging in response to an event that has occurred. The term *log event* is used for the result of this action, i.e. the actual log entry. The event may cause other actions from various applications, but that is outside the scope of this chapter.

An event, in the sense of a trigger, happens at a specific point in time and can have a duration which shall be logged. If the log event indicates an error condition in some equipment, it is useful to know not only when the error condition was entered, but also if the condition still persists, and eventually at what time the error was cleared. We define this as a *stateful log event*, whereas the log event without duration is called a *singular log event*.

The following use cases are chosen for evaluating the pattern for log events and alarms.

- An application logs a singular event related to an object;
- An application logs a stateful event related to an object. Later, the application marks the event as cleared/inactive;
- A responsible user views open events for an object, and acknowledges them;
- A user or an application requests a report over historical events.

### 6.9.1 Logging a singular event

Any application can log an event at any time. The receiver(s) of the log event need not be known to the log source. Indeed, the source should make no assumptions of an action being triggered by logging the event. Such actions must be explicitly called upon through other interfaces.

A log event is an instance of a *log event type*. The log event type is a static definition, which identifies and documents the nature of a log event. E.g. a log event type may have the meaning “switch out of control”, while a log event (an instance of this type) means that a specific switch has gone out of control at a specific time. Each application is free to maintain its own set of log event types.

A log event is associated with a *severity* attribute, which governs how it is propagated and presented. The severity is defined by the log event type, out of a fixed set of options.

A log event will normally refer to an *object*, which is the "owner" of the event (the source of the event or the entity mainly affected by it). The object can be any addressable resource in the data set, e.g. a wayside object, a time tabled trip, a temporary restriction.

A log event may optionally carry a set of *properties*, providing additional information about the log event. The type and significance of the properties is defined by the log event type. E.g. for a log event type "user logged in", a property may carry the name of the user.

A log event type may specify that events of this type require acknowledgment from a user. If so, the logging application shall indicate the event as *waiting for acknowledgment*. See further in 0.

An event is associated with a set of *timestamps*. Minimum for the time of event occurrence, when applicable also for the time of user acknowledgment.

Any application can subscribe to logged events, optionally filtered on any of the event attributes. This includes the "big data" system for asset status statistics, described in D9.2. Hence, there is no explicit use case included here for that interface.

A log event may be configured (per log event type) to trigger a workflow. The execution of the workflow is however not covered by this use case.

### 6.9.2 Logging a stateful event

This use case is identical to the previous, except that the logging application indicates the log event as *active*. A log event, which is not active when logged, indicates a singular ("pulsed") event, e.g. "route request issued". A log event, which is active when logged, represents a stateful ("on/off") event, e.g. "switch out of control". The logging application is expected to set a stateful log event to inactive when the condition causing the event is no longer at hand.

A stateful event is associated with an additional timestamp, for the time of deactivation.

### 6.9.3 Use case: Viewing and acknowledging open events for an object

A log event, which is either active or waiting for acknowledgment, is considered as an *open* log event.

Through a UI application, a user with sufficient privileges in relation to a specific object shall be able to retrieve all open log events owned by that object.

A user with sufficient privileges in relation to a specific object shall be able to reset the "waiting for acknowledgement" attribute of any log event owned by that object. Privileges for acknowledgment may differ from the privileges for viewing open events.

### 6.9.4 Historical Events

An application shall be able to retrieve all log events that are persisted by the IL, optionally filtered on any of the log event attributes, e.g. for creating reports. It is up to the retrieving

application to implement sorting, grouping, joining, statistical calculations etc. on the retrieved data.

### 6.9.5 Pattern for Event Management

The use cases above identifie two main classes for the data model:

- "LogEventType", containing static definitions;
- "LogEvent", created dynamically for each event instance and persisted for some time.

The "LogEventType" objects should be owned by the respective applications. For the "LogEvent" objects, the situation is a little more complex. Open events should be accessible through the entities owning them (the "object" attribute), which also controls their access rights. But closed events (as well as open) should also be accessible globally, even after their original owners have ceased to exist.

To handle this, an additional "EventLog" object is introduced, as a "co-owner" of the events. In UML terms, the events are contained in the event log through composite aggregation, while they are contained by their source objects through shared aggregation.

To illustrate the possible representation of these concepts with key/values pairs, see examples in Table 6.11.

| Path | Content |
|---|---|
| /network/controlSystem/prod/applications/uuid1/eventTypes/uuid2 | A log event type definition for the application "uuid1" |
| /network/controlSystem/eventLog | The global event log. This path is NOT a key with value. |
| /network/controlSystem/eventLog/events/uuid3 | A log event instance, accessed through the event log |
| /network/infrastructure/assets/switches/uuid4/events/uuid3 | The same log event, accessed through its owner object |

**Table 6.11: Pattern for Event Management**

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_55 | It shall be possible for applications to publish new log event types (EventType class of the CDM), defining the significance and severity of a type of log event (Event class of the CDM). |
| REQ_IRS_56 | It shall be possible for applications to log events, referencing a log event type, an object (being any identifiable entity according to the CDM) and optionally additional data. |
| REQ_IRS_57 | It shall be possible for a logging system to specify if a log event shall be persisted by the IL or not (Note: persistence should be avoided for low severity log events, in order to avoid congestion). |
| REQ_IRS_58 | It shall be possible for a logging system to specify a limited time to live for a log event, after which it may be dropped by the In2Rail platform. |

| Requirements # | Description |
|---|---|
| REQ_IRS_59 | It shall be possible to configure a system wide maximum time to live for all log events. |
| REQ_IRS_60 | An application which indicates a log event as active when logged, shall set the log event as inactive when the condition causing the log event is no longer at hand. This means that CDM should separate the properties that can be modified and properties that cannot be modified after they have been set. |
| REQ_IRS_61 | A user shall be able to acknowledge a log event, if required according to log event type data, subject to the user access rights in relation to the object indicated in the log event. CDM should make it possible that the application acknowledging the log event has no write access to the log event itself (Acknowledgment and Event should be different classes in CDM). |
| REQ_IRS_62 | The time of creating and modifying a log event shall be recorded, as close to the actual source of the event as possible. This means that CDM should define, for each Event, a timestamp, which is different from the timestamping provided by In2Rail platform for every Key/Value Pair. |
| REQ_IRS_63 | Applications shall be able to subscribe to logged events, optionally filtered on a combination of log event attributes. The In2Rail Platform shall forward incoming log events to relevant subscribers. |

**Table 6.12: Event Management- Requirements on the low-level interface**

Applications shall be able to query the IL for previously logged events, optionally filtered on a combination of log event attributes and time stamps. This includes filtering on object, state and acknowledgement as described in use case “Logging a stateful event”.

## 6.10 GUI interactions

In a Traffic Management System, it is possible that several graphical user interfaces to co-exist, which have been delivered by different suppliers. The Integration Layer is used to connect the different GUIs with each other. For a smooth interaction between the GUIs, information has to be sent between the user interfaces via the IL. In order to understand the required interaction a few use cases are shown below which are used as a baseline in order to specify patterns of user interface interaction.

The following use cases have been selected to evaluate a pattern for GUI interaction:

- Synchronize selected objects:
  - A train is selected by double click in a view of one TMS graphical user interface module and because of this the train is centered within another TMS GUI module.
  - A station is made the current object by mouse double click in one GUI and because of this, the corresponding CCTV view of the selected station is shown within another TMS GUI modules
  - An alarm is selected by keyboard and <Enter> key in the alarm view and the corresponding object is shown in another view of a GUI module.

### 6.10.1 Train selection in time-distance diagram and centring the train in topology view

A train or object is selected and the action is published via the IL. One or more user interface modules may subscribe for this action and adapt their display depending on the type of action and the corresponding object. The action itself is stateless.

### 6.10.2 Station is selected and corresponding CCTV view of station opens

A user double clicks with the mouse on a station in the topology view. The mouse action is published and the subscribing module, in this case the CCTV view module, opens the CCTV view of the corresponding station.

### 6.10.3 Alarm is selected and corresponding object is shown

The operator selects the alarm by keyboard or mouse and this action is published to the IL with the corresponding object information.

### 6.10.4 Pattern for GUI interactions

The synchronisation of selected object views mostly involves required interaction between graphical user interfaces. Consequentially, certain “Actions” should be exchanged between the different GUI modules. A GUI action can be for example a selection of a train in one GUI module.

The GUI actions of a GUI module are only interesting for other GUI modules and not for other applications that contain for example the business logic. Hence, there is only a need for a Key/Value Pair for GUI actions within the user interface part of the CDM. In principle, references to all objects that can be selected and that are of interest for other user interface modules shall be possible.

The Key/Value Pairs for GUI interaction shall be represented by paths like those shown in the table below. The location of the data inside of canonical model (path) gives just a hint for implementation.

Apart from object selection, more detail information on the current operation the operator is working on could be interesting for other GUI in to optimize their internal layout (order of menus, presenting some actions buttons in special areas…). For that, GUI modules will also be able to publish/subscribe to a current Activity. The activity should be a class defined in CDM and containing an activity name (at least). It is not imposed how other GUI should react to a change of the current activity (internal layout may be unchanged), neither how is the activity changed (it can be inferred from last issued command, some expert system, depending on the operator profile or set manually by the operator).

Therefore, the scope of selections can be more or less wide. Selection can be limited to a single view, or a single operator station. To simplify integration, we suppose that the view

should be used to represent global selection on a workstation. It means that is a kind of virtual view.

Finally, it appeared that GUI may distinguish the selection (initiated by the operator) from the highlighting (which can be initiated by the operator, but also by a change in the system independent from operator): Example of highlighting could be that when a train is selected and "monitored", its next operational point is highlighted (automatically as the train moves). As for selection, highlighting may be different for each view.

| Path | Content |
|---|---|
| /network/ViewControllers/uuid | path for all Key/Value Pair concerning one workstation (one ViewController per workstation). This path is NOT a key with value. |
| /network/ViewControllers/uuid/Views[n]/CurrentActivity | path for the current activity on one view. Its content is a CurrrentActivity object which may be used by GUI modules to optimise the interface. |
| /network/ViewControllers/uuid/Views[n]/CurrentSelection | Path for an instance of a selection object (defined in CDM). This object can contain an ordered list of keys of objects, selection mode (mouse or keyboard), weight of the selection (light, medium, strong, for instance strong for double-click), as will be indicated in the CDM object. |

**Table 6.13: Pattern for GUI interactions**

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_64 | The GUI applications shall be able to publish their activity on the integration layer. |
| REQ_IRS_65 | Applications shall be able to subscribe to activities. The IL shall forward incoming activities to relevant subscriber. |
| REQ_IRS_66 | In2Rail Platform permits to applications and modules to issue activities without prior knowledge of the module or application that will handle the action. |
| REQ_IRS_67 | The activities shall not be persisted longer than GUI sessions. |
| REQ_IRS_68 | There can be several instances of the same activity running at the same time in the system (but not for the same workstation in the same view). The interface gives access each of these instances individually. |
| REQ_IRS_69 | The CDM allows querying all instances of a given activity (type). |
| REQ_IRS_70 | The CDM allows creating new instance of activity. |
| REQ_IRS_71 | For each activity (type), access rights can be defined. Only modules having the right access can issue the action. |

**Table 6.14: GUI interactions- Requirements on the low-level interface**

When a module does not have the right access, but attempts to issue an activity, the denied access is logged and the action is not issued.

## 6.11 Sessions

A Session is defined as a mechanism used to encompass sandboxes and reservations created during operations. Sessions are held by session-owners and have a lifecycle and can be voluntary or automatically deleted (after the expiry of associated watchdogs or the expiry of work-shift) or can be handed-over to a different session-owner.

All sandboxes and reservations encompassed by a session follow the lifecycle of the encompassing session.

Sandboxes hold copies of Topics that may be freely changed without impacts to current operations. Changes done on sandbox Topics may be applied to original Topics, or discarded.

Reservations (aka locks) implement a mechanism that limits the access to TMS controlled objects and functions. Reservations are set by applications based on authorization, roles and rules associated with e.g. the end-users using the applications. *How authorization is implemented is not in the scope of this document and therefore would not described here.*

### 6.11.1 Acquisition of AoR and hand-over of Station Control

In this high-level use case, two different actors are involved: the dispatcher, responsible to secure the time table and to take operational decisions for the near future; and the Signaler, responsible for local control of a station.

The dispatcher logs into the system and acquires its AoR and all the TMS resources and functions related to the AoR are reserved in the context of the session associated to the dispatcher.

At some point in time, the dispatcher needs to hand-over the control of a station to a signaller. Following the national regulations, the dispatcher grants the full control to local signaling system to the signaller. After hand-over, the reservations of the local signaling system are moved from the dispatcher to the signaler: the signaler will have full control of local signaling system, and the dispatcher will be unable to interfere with signaller activities.

### 6.11.2 Patterns for Sessions

The use case described above identifies three logical sections for the data model. A fourth section for private Topics will be added.

The data model comprises:

- Meta data, containing the properties of the session itself;
- sandboxes, which can be implemented in different ways in the CDM;
- reservations holding a list of references;
- dynamically created private Topics.

The following table depicts one of the possible structuring in the data set.

| Path | Content |
|---|---|
| /network/users/<uuid1>/sessions/<uuid2>/meta/ | Path for section containing metadata regarding session management. |
| /network/users/<uuid1>/sessions/<uuid2>/meta/SessionName | Key/value pair containing the session name/description in plain text. |
| /network/users/<uuid1>/sessions/<uuid2>/meta/StartTimestamp | Key/value pair containing the timestamp of session start. |
| /network/users/<uuid1>/sessions/<uuid2>/meta/ExpectedDuration | Key/value pair containing the timestamp of the expected session end. |
| /network/users/<uuid1>/sessions/<uuid2>/meta/WatchdogTimeout | Key/value pair containing the watchdog period. This key is to be continuously refreshed by the owner of the session in order to keep the session alived. |
| /network/users/<uuid1>/sessions/<uuid2>/sandbox/<uuid3>/watchDog | Key/value pair containing the watchdog period. This key is to be continuously refreshed by the owner of the session in order to keep the session alive. |
| /network/users/<uuid1>/sessions/<uuid2>/sandbox/<uuid3>/CurrentTime | Key/value pair for the current time to be used by a sandbox which is not attached to present time. |
| /network/users/<uuid1>/sessions/<uuid2>/sandbox/<uuid3>/TimeTable/Trains/Train/uuid/Position/Futures/now-some-time-in-future | Key/value pair for the list of sampled positions of this train between now and some-time-in-future (example now-22-10-2035-10:15:00) in a studying scenario around /network/user<uuid>/session<uuid>/sanbox<uuid>/CurrentTime. The sampling time is some configuration data of the Position type. |
| /network/users/<uuid1>/sessions/<uuid2>/sandbox/<uuid> | Path for section containing the name of the associated sandbox.<br>If this path has no further sub-path, then it is a reference to a sandbox stored in a predefined path (e.g. /network/sandboxes). |
| /network/users/<uuid1>/sessions/<uuid2>/reservations/<uuid3>/ | Path for section containing data for a single reservation. |
| /network/users/<uuid1>/sessions/<uuid2>/reservations/<uuid3>/watchDog | Key/value pair containing the watchdog period. This key is to be continuously refreshed by the owner of the session in order to keep the reservation alive. |
| /network/users/<uuid1>/sessions/<uuid2>/reservations/<uuid3>/nodeList | Key/value pair containing the reference to authorization Key/Value Pairs allocated in a reserved area of the CDM (*not described in this deliverable*). |
| /network/users/<uuid1>/sessions/<uuid3>/privateTopics | Path containing data for a privateTopics. |
| /network/users/<uuid1>/sessions/<uuid2>/privateTopics/<topicName> | Key/value pair containing the topic to be subscribed to by other applications. |

**Table 6.15: Patterns for Sessions**

The following requirements on the low-level interface (applicable to the IL-API and CDM) have been defined:

| Requirements # | Description |
|---|---|
| REQ_IRS_72 | It shall be possible for an application to start one or more sessions after the application is authenticated by the IL. |
| REQ_IRS_73 | Each session shall have a unique identifier which is univocally associated to a specific user. In this context the term user means either an application or an end user. The specific user is referred as the owner of the session. |
| REQ_IRS_74 | A set of reservations may be associated to a given session. |
| REQ_IRS_75 | A set of sandboxes may be associated to a given session. |
| REQ_IRS_76 | A set of private Topics to be subscribed to may be associated to a given session. |
| REQ_IRS_77 | Each session should have an expected duration after which the session is forcibly closed and all of its related objects (e.g. reservations, sandboxes, private Topics) are deleted. Expected duration may be intentionally extended by application(s) before its expiration. Expected duration may correspond, e.g., to the work shift. |
| REQ_IRS_78 | Each session shall have its own watchdog to be refreshed by application(s). On the expiry of the watchdog period the session is forcibly closed and all of its related objects (e.g. reservations, sandboxes, private Topics) are deleted. |
| REQ_IRS_79 | The owner of a session may transfer the entire session to another owner by starting a session hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. |
| REQ_IRS_80 | The owner of a session may transfer part of reservations associated to the session to another session (possibly owned by another user) by starting a reservation hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. |
| REQ_IRS_81 | The owner of a session may transfer part of sandboxes associated to the session to another session (possibly owned by another user) by starting a sandbox hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. |
| REQ_IRS_82 | Each sandbox should have its own associated watchdog and expected duration. If not, they shall inherit the ones from the encompassing session. |
| REQ_IRS_83 | Each reservation should have its own associated watchdog and expected duration. If not, they shall inherit the ones from the encompassing session. |

**Table 6.16: Sessions - Requirements on the low-level interface**

# 7 Conclusion

The chosen method to select use cases, analyse them for possible constraints and formalise them into requirements has been very successful in defining the requirements for the Interfaces which are within the scope of this document.

The overall approach to apply a Canonical Data Model as the “Language of the Integration Layer” has allowed the creation of the requirements for the Interfaces of the IL and the pattern for the different use cases in an effective and unique way.

However, this document is a “living document” and will require periodical updates during the lifetime of In2Rail, and the proceeding projects of the S2R program and will need to incorporate evolving requirements triggered from the subsequent design works on the Integration Layer.

The results of this activity will be integrated into the “Proof of Concept” activities under In2Rail WP7 Task 3, which is in line with the use cases presented in this document.

The “manageable complexity” of the achieved results are supporting a “Standardisation” of the Interfaces of the Integration Layer as soon as the Design of the Data Model and Processes of the communication infrastructure are finalised and agreed within the railway sector.

# Appendix 1: Low Level Interface Requirements for IMDG (IL-API)

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_1 | The In2Rail Platform permits applications and modules to access (read or write) individual Key/Value Pair content in their binary serialised form. The lowest level interface usable by module or application to access Key/Value Pair content is the IL-API. | |
| REQ_IRS_2 | The In2Rail Platform allows applications and modules to create and delete individual Key/Value Pair in the data grid. Type (class in CDM) must be specified when a Key/Value Pair is created. | |
| REQ_IRS_3 | The binary serialised form of any Key/Value Pair contains the serialised data of a complete Key/Value Pair including (recursively) all subparts, with the special case that accessible Key/Value Pairs which are part of the Key/Value Pair are serialised as an "address" which allows to retrieve the actual content of the accessible Key/Value Pair. | |
| REQ_IRS_4 | The "address" (UUID or XPATH like string) of an accessible Key/Value Pair B which is included in a Key/Value Pair A, should be stable in time at least during the lifetime of both A and B. This means that as long as A and B both exist (none of them has been deleted), then the address's (UUID or XPATH like string) validity is guaranteed. | |
| REQ_IRS_5 | The IL-API is transparent to the serialised data of Key/Value Pair content. This means that in no treatment, the implementation of IL-API need to deserialise the Key/Value Pair content. | |
| REQ_IRS_6 | The binary serialisation method (i.e. mapping between the CDM and serialised data) should be the same for all type of Key/Value Pairs of the CDM and for instances of Key/Value Pairs, within a single deployment of the In2Rail Platform. This requirement does not need to be respected when the serialisation is changed; in that case, and during a non-production time, all persistent data can be de-serialised and re-serialised using a different method, and during that process, the serialisation method is not uniform. | |
| REQ_IRS_7 | Two different implementations of the In2Rail Platform can use different serialisation methods. Bridges are then used to connect these In2Rail PLATFORM implementations. | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_8 | Serialisation method allows to represent at least the following basic data types: Variable length string, 8 bits signed integer, 32 bits signed integer, 128 UUID, Boolean, Simple (32 bits) and double (64 bits) precision floating point (IEEE 754). The following basic types, unsigned integer 8 bits, 32 bits and 64 bits can be used in the CDM to indicate that the negative values are not authorised, but the MSB can never be set. This means that they are serialised as their signed counterparts. | |
| REQ_IRS_9 | Serialisation method allows to serialise enumerated values. | |
| REQ_IRS_10 | Serialisation method allows to serialise structures (sequence of named values of any serializable data type, which are called attributes). | |
| REQ_IRS_11 | Serialisation method allows to serialise unions (one value chosen among several serializable data type). The chosen data type of each value is part of the union serialisation, it does not require a separate attribute. | |
| REQ_IRS_12 | Serialisation method allows to serialise sequence of data type (serialisation of arrays). The serialisation method should handle in the sequence the number of elements (not necessarily using a counter, stop markers or other methods are possible too). | |
| REQ_IRS_13 | Serialisation method should allow the presence of non-valued data (example, attribute not valued in a structure, or element not valued in a sequence, but present to maintain indexes value). | |
| REQ_IRS_14 | Serialisation method should allow structure extension mechanism, at least through the non-valued data attribute representation. | |
| REQ_IRS_15 | The In2Rail Platform allows modules to be notified of values changes (change of the serialised data) of given Key/Value Pairs. Monitored Key/Value Pairs can be specified individually or by filter (regular expression on the key). | |
| REQ_IRS_16 | The In2Rail Platform allows modules to be notified of apparition and suppression of Key/Value Pairs. The module indicates which are the Key/Value Pair of interest either by the type (class in CDM) or by a regular expression applied to the path of Key/Value Pair or by a combination of both. | |
| REQ_IRS_20 | The In2Rail Platform permits applications and modules to issue commands to other applications and modules without prior knowledge of the module or application that will handle the command. The lowest level | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| | interface usable by module or application to access commands is the IL-API. | |
| REQ_IRS_21 | There can be several instances of the same command running at the same time in the system. The interface gives access each of these instances individually. | |
| REQ_IRS_22 | The IL-API allows querying all instances of a given command (type). | |
| REQ_IRS_23 | The IL-API allows creating new instance of command. A new command is by default in “preparation” state. | |
| REQ_IRS_24 | The status of each instance of command can be queried (and set) individually. Status includes at least the following values: Preparation, Submitted, Running, Completed, Failed and Aborted. | |
| REQ_IRS_25 | For each instance of command, CDM allows storing parameters attached to a key, separate from the command’s other keys. The parameters of each type of command are defined by the Canonical Data Model. | |
| REQ_IRS_26 | The result of a command can be read/write independently (separate key/value) from the rest of the command. | |
| REQ_IRS_27 | The result of a command is given an expiration date. The In2Rail platform keeps the result in the data store at least until the expiration date is reached or some module (issuer of the command) explicitly removes the value of the result. After expiration date, the In2Rail Platform should remove the expired result (and command) to limit memory usage. | |
| REQ_IRS_28 | The requester of a command can also request to cancel the command (abortion request). The module executing the command is NOT forced to take abortion request into account. | |
| REQ_IRS_29 | A sandbox is by default attached to each command instance. The sandbox contains data which can be required to run a BPM triggered command and scenario study in the frame of the execution of the command. The sandbox should be removed when the command is complete (Completed, Failed, or Aborted). | |
| REQ_IRS_30 | For each command (type), access rights can be defined. Only module and or Persons having the right access can issue the command. | |
| REQ_IRS_31 | When a module or person does not have the right access tries to issue a command, the denied access is logged and the command is not issued. | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_32 | The In2Rail Platform permits to applications and modules to issue change requests to other applications and modules without prior knowledge of the module or application that will handle the change request. | REQ_IRS_20 |
| REQ_IRS_33 | There can be several instances of the workflow running at the same time in the system. The interface gives access each of these instances individually. | |
| REQ_IRS_34 | The CDM allows access to the current state of all workflows. | |
| REQ_IRS_35 | The finished workflow is given an expiry date. The AL/IF keeps the workflow intermediate states at least until the expiry date is reached or some module (issuer of the command) explicitly removes the value of the result. After the expiry date, the In2Rail Platform should remove the expired data to limit memory usage. | |
| REQ_IRS_36 | The requester of a workflow can also request to cancel it (abortion request). The module executing the command is NOT forced to take abortion request into account. | |
| REQ_IRS_37 | A sandbox can be attached to different steps of the workflow either by reference or by value. The sandbox should be removed when the workflow is finished (Completed, Failed, or Aborted). | |
| REQ_IRS_38 | For each workflow type, access rights can be defined. Only applications having the right access can issue read the state or contribute to the workflow. | |
| REQ_IRS_39 | The In2Rail Platform permits to applications and modules to subscribe to only configured Topics | |
| REQ_IRS_40 | The In2Rail Platform permits to applications and modules to publish to only configured Topics | |
| REQ_IRS_41 | The In2Rail Platform permits to applications and modules to open their own private Topics and configure access rights for other applications. | |
| REQ_IRS_42 | The In2Rail Platform support access rights for stream oriented and multicast protocols. | |
| REQ_IRS_43 | The In2Rail Platform uses one combined API for getting access to the data on IL and notifies the client application about trials of access violation. | |
| REQ_IRS_44 | A Topic shall be configurable with the number of historical samples (History Depth). This number can be used to limit resource consumption on the IL. | |
| REQ_IRS_45 | A subscriber shall be able to configure required History Depth, which can be smaller than the Topic configuration. | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_46 | The handling of the history shall not increase the API complexity to IL. | |
| REQ_IRS_47 | The IL shall support at least one historical Topic per cluster. | |
| REQ_IRS_48 | The In2Rail Platform permits to applications and modules to store and query values and events between present time and a moving end of near future called horizon. For each type of Key/Value Pair, the canonical data model determines if the values are continuous or event typed. | |
| REQ_IRS_49 | For continuously forecasted value, configuration of In2Rail Platform gives for each type of data the sample period (time elapsed between two contiguous values) and the slice size (the maximum number of samples in the same key/value pair for this data) | |
| REQ_IRS_50 | Extent of horizon should be configurable, either statically or dynamically (at runtime). Dynamic configurability of horizon is not mandatory. | |
| REQ_IRS_51 | For continuously forecasted value, the slices have fixed boundaries, except the slice which starts at present time (now). Therefore, the first slice, starting at present time, has a variable number of samples. Other slices have fixed number of samples (for the same type of data). Modules can read/subscribe to/write whole slices but not part of slices using the IL-API. | |
| REQ_IRS_52 | For event typed forecasted values, only change of values can be read/subscribed to/written, and all events based forecasted value is attached to a time when the value change is forecasted. | |
| REQ_IRS_53 | Continuous and event typed forecasted values can also exist in the sandboxes whose current time is corresponding to present time. Nevertheless, some command need to be issued to trigger their computation and therefore their apparition in the data store. | |
| REQ_IRS_54 | It is possible to set the current time of a sandboxes to some moment in the future. Their horizon is then moved accordingly. When attaching a new moment to a sandbox, then all now-some-time future should be invalidated/removed. This is not the responsibility of the In2Rail platform to do so. It is the responsibility of the module which changes the sandbox current time. | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_55 | It shall be possible for applications to publish new log event types (EventType class of the CDM), defining the significance and severity of a type of log event (Event class of the CDM). | |
| REQ_IRS_56 | It shall be possible for applications to log events, referencing a log event type, an object (being any identifiable entity according to the CDM) and optionally additional data. | |
| REQ_IRS_57 | It shall be possible for a logging system to specify if a log event shall be persisted by the IL or not (Note: persistence should be avoided for low severity log events, in order to avoid congestion). | |
| REQ_IRS_58 | It shall be possible for a logging system to specify a limited time to live for a log event, after which it may be dropped by the In2Rail platform | |
| REQ_IRS_59 | It shall be possible to configure a system wide maximum time to live for all log events. | |
| REQ_IRS_60 | An application which indicates a log event as active when logged, shall set the log event as inactive when the condition causing the log event is no longer at hand. This means that CDM should separate the properties that can be modified and properties that cannot be modified after they have been set. | |
| REQ_IRS_61 | A user shall be able to acknowledge a log event, if required according to log event type data, subject to the user access rights in relation to the object indicated in the log event. CDM should make it possible that the application acknowledging the log event has no write access to the log event itself (Acknowledgment and Event should be different classes in CDM). | |
| REQ_IRS_62 | The time of creating and modifying a log event shall be recorded, as close to the actual source of the event as possible. This means that CDM should define, for each Event, a timestamp, which is different from the timestamping provided by In2Rail platform for every Key/Value Pair. | |
| REQ_IRS_63 | Applications shall be able to subscribe to logged events, optionally filtered on a combination of log event attributes. The In2Rail Platform shall forward incoming log events to relevant subscribers. | |
| REQ_IRS_64 | The GUI applications shall be able to publish their activity on the integration layer. | REQ_IRS_1<br>REQ_IRS_2 |
| REQ_IRS_65 | Applications shall be able to subscribe to activities. The IL shall forward incoming activities to relevant subscriber. | REQ_IRS_15 |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_66 | The In2Rail Platform permits to applications and modules to issue activities without prior knowledge of the module or application that will handle the action. | |
| REQ_IRS_67 | The activities shall not be persisted longer than GUI sessions. | |
| REQ_IRS_68 | There can be several instances of the same activity running at the same time in the system (but not for the same workstation in the same view). The interface gives access each of these instances individually. | |
| REQ_IRS_69 | The CDM allows querying all instances of a given activity (type). | |
| REQ_IRS_70 | The CDM allows creating new instance of activity. | REQ_IRS_2 |
| REQ_IRS_71 | For each activity (type), access rights can be defined. Only modules having the right access can issue the action. | |
| REQ_IRS_72 | It shall be possible for an application to start one or more sessions after the application is authenticated by the IL. | |
| REQ_IRS_73 | Each session shall have a unique identifier which is univocally associated to a specific user. In this context the term user means either an application or an end user. The specific user is referred as the owner of the session. | |
| REQ_IRS_74 | A set of reservations may be associated to a given session. | |
| REQ_IRS_75 | A set of sandboxes may be associated to a given session. | |
| REQ_IRS_76 | A set of private Topics to be subscribed to may be associated to a given session. | |
| REQ_IRS_77 | Each session should have an expected duration after which the session is forcibly closed and all of its related objects (e.g. reservations, sandboxes, private Topics) are deleted. Expected duration may be intentionally extended by application(s) before its expiration. Expected duration may correspond, e.g., to the work shift. | |
| REQ_IRS_78 | Each session shall have its own watchdog to be refreshed by application(s). On the expiry of the watchdog period the session is forcibly closed and all of its related objects (e.g. reservations, sandboxes, private Topics) are deleted. | |
| REQ_IRS_79 | The owner of a session may transfer the entire session to another owner by starting a session hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. | |

| Id | Text | Not necessary because implied by |
|---|---|---|
| REQ_IRS_80 | The owner of a session may transfer part of reservations associated to the session to another session (possibly owned by another user) by starting a reservation hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. | |
| REQ_IRS_81 | The owner of a session may transfer part of sandboxes associated to the session to another session (possibly owned by another user) by starting a sandbox hand-over; the transfer is completed by the receiving owner upon starting a correspondent take-over. | |
| REQ_IRS_82 | Each sandbox should have its own associated watchdog and expected duration. If not, they shall inherit the ones from the encompassing session. | |
| REQ_IRS_83 | Each reservation should have its own associated watchdog and expected duration. If not, they shall inherit the ones from the encompassing session. | |