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 D8.6

Description of the Generic Application Framework and its constituents

Due date of deliverable	Month 27
Actual submission date	30-07-2017
Organization name of lead contractor for this deliverable	SIE
Dissemination level	PU
Revision	Final

Authors

		Details of contribution
Authors	**CAF Signalling (CAF)** Carlos Sicre Vara de Rey Manuel Castro Vinas	Section 8, contribution to 1-12
	HaCon (HC) Sandra Kempf Rolf Gooßmann	Section 10, contribution to 1-12
	SIEMENS (SIE) Stefan Wegele	Sections 1-7, 9, 11,12 contribution to 1-12
Contributor(s)	**Ansaldo STS (ASTS)** Gian Luigi Zanella Matteo Pinasco	Sections 1- 12
	AZD Praha s.r.o (AZD) Martin Bojda Michal Zemlicka Martin Ruzicka	Sections 1- 12
	Bombardier Transportation (BT) Martin Karlsson Roland Kuhn	Sections 1- 12
	Thales (THA) Jean-Jacques Rodot Jean-Yves Friant	Sections 1- 12

Executive Summary

This document provides a description of the Generic Application Framework and is directed at the software architects and developers of Traffic Management applications in the Railway industry.

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

There are three In2Rail Work Packages relating to Intelligent Mobility Management (I2M). In the first 12 months of the project requirements on Traffic Management Systems (TMS) from the recent tenders, national and international projects were collected in D7.1 and D7.2. In deliverable D8.1 requirements on Integration Layer and D8.4 requirements on Application Framework were specified.

Starting working on the Integration Layer and having several prototypes done, the participants came to a conclusion to use the Integration Layer as a communication platform for the functionality of the Application Framework. The design and specification of the Integration Layer will be published in D8.4 nine months after this deliverable. In this document assumptions about the Integration Layer were made, which could be subject of minor changes during this period.

First the experience with the Integration of Applications in different projects, and the historical approaches for software integration as well as currently active projects for cloud-intensive IT industry were analysed. The conclusion of this analysis was that the Application Framework shall be able to express the desired state of deployment and applications states on the Integration Layer and observe the current state on some other topics on the Integration Layer. The specific implementation of the desired state is the responsibility of the future product “Application Framework”.

The second step comprised the identification of the information required in the “desired state”. The detailed specification of the data structures describing desired and current states is provided in D8.7.

In the second part of the document (chapter 10) an important question about Integration of User Interfaces provided by multivendor-applications was analysed. Although the current trend in IT industry is moving to rendering of the application content by web browser and providing HTML5/CSS3 based content from the cloud, it was assumed that in the near future dedicated User Interfaces with high performance graphics will still be the main use case in control centres.

TABLE OF CONTENTS

Glossary, Abbreviations and Acronyms

** Definition extract from §Common Glossary of the* [IN2RAIL D7.1] *deliverable of In2Rail.*

Term	Description
AF	Application Framework
API	**A**pplication **P**rogramming **I**nterface
Application Framework	It is a programming framework allowing plug-and-play integration of multivendor applications into TMS.
COTS	Commercial off-the-shelf
DDS	**D**ata **D**istribution **S**ervice – an OMG standard.
ESB	Enterprise Service Bus
IL	**I**ntegration **L**ayer
IM *	**I**nfrastructure **M**anager: 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 *	**I**ntelligent **M**obility **M**anagement: information developed as a strategically critical asset: ▪ A standardised approach to information management and dispatching systems enabling an integrated Traffic Management System (TMS). ▪ 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. An advanced asset information system with the ability to 'nowcast' and forecast network asset statuses with the associated uncertainties from heterogeneous data sources.
IMDG	In Memory Data Grid: a software installed on one or several nodes and provides reliable access and change notifications for key-value pairs in presence of failures.
LCC	**L**ife **C**ycle **C**osts
Message semantic	Definition of the meaning of every message attribute
Message encoding	Representation of the message content in bits and bytes.
Node	Computing entity: PC, server, Virtual Machine, Cluster.
QoS	Quality of services
RCP	Rich Client Platform
Reliable communication protocols	A set of messages to be exchanged between several partners which ensures defined reliability properties in presence of failures and message losses.
RTTP	**R**eal **T**ime **T**raffic **P**lan: the timeframe of the Daily Timetable transferred from the IM planning department to the Traffic Management Department.

Term	Description
RU *	**R**ailway **U**ndertaking: bodies such as train operating companies and freight operating companies, which are responsible for the operation of passenger and freight trains.
TCO	**T**otal **c**ost of **o**wnership
TOC *	**T**rain **O**perating **C**ompany: a company with access rights to operate passenger trains on the railway network.
TMS *	**T**raffic **M**anagement **S**ystem: a traffic control-command and supervision/management system, such as ERTMS in the railway sector.
WP7	**W**ork **P**ackage **7**: System Engineering of Intelligent Mobility Management (I²M) of In2Rail.
WP8	**W**ork **P**ackage **8**: Integration Layer of Intelligent Mobility Management (I²M) of In2Rail.
WP9	**W**ork **P**ackage **9**: Intelligent Mobility Management (I²M) - Nowcasting and Forecasting
UI	**U**ser **I**nterface

1 Background

This document represents the next step in system development after requirement specification in D8.5 “Requirements for the Generic Application Framework” in the framework of the project entitled “Innovative Intelligent Rail” (Project Acronym: In2Rail; Grant Agreement No 635900). This step is typically referenced as “High Level Design” in V-Model software development (see Figure 1.1).

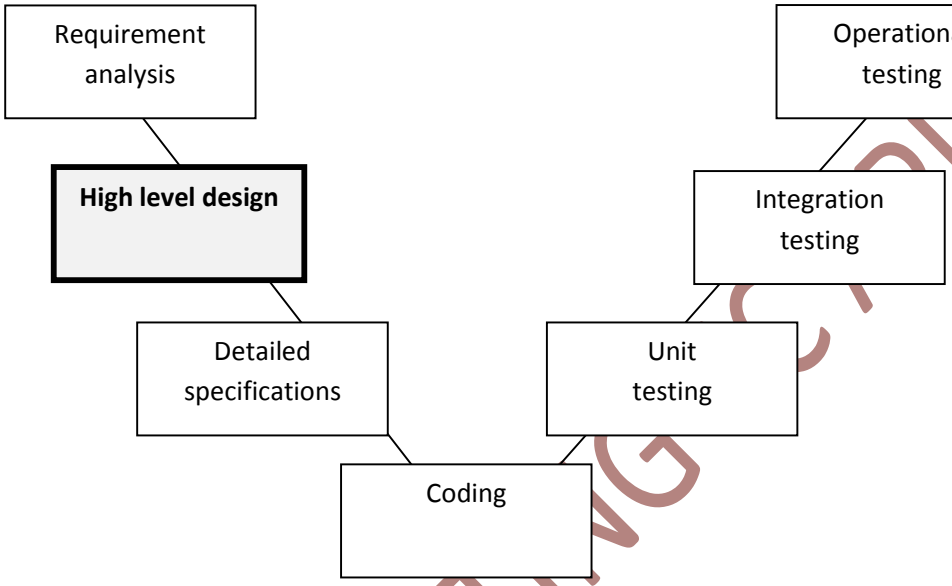

Figure 1.1: Location of this document in the Software-Development process

The WP8 activity is integrated into the development process with WP7 (see Figure 1.2).

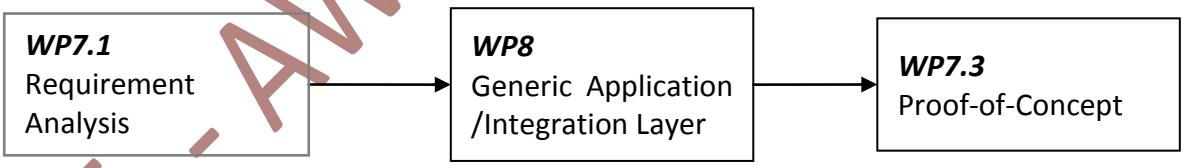

Figure 1.2: Simplified view on integration of WP8 and WP7

The WP8 consists of two main parts the Integration Layer and the Application Framework. The Integration Layer represents the communication platform for applications to be integrated into TMS. It will specify the API to access data (publish and subscribe) as well as the data structures persisted and distributed by the Integration Layer. The set of data structure specifications for Integration Layer is called “canonical data model” according to the ESB term for common message formats.

The canonical data model will provide an additional specification to allow easy extensions for future applications in TMS and railway industry in general.

2 Objective / Aim

As part of the In2Rail project, the work package "Intelligent Mobility management – Integration Layer" shares two major objectives:

1. Reducing Life Cycle Costs (costs for development, deployment, execution and maintenance of the software) for Traffic management systems,
2. Enabling new intelligent functions for overall optimisation of the traffic process.

To achieve the cost reduction two main cost drivers must be considered:

- Integration efforts – integration of a new software application/module currently requires the development of interface gateways introducing costs for:
  - Interface clarification,
  - Software for message conversion and aggregation, object id mapping etc.,
  - Development/establishment of proprietary reliable communication protocols
  - Testing, versioning, and configuration of the gateways;
- Absence of a standardised development platform requires each application vendor to repeatedly solve "high availability" and "high performance" issues. This reduces possible vendor market and often requires extensive amount of hardware, as each application is delivered by a vendor with redundant hardware components.

As a consequence of both cost drivers preventing fine modularisation of the software big monolithic applications developed by few vendors cover most of the TMS market.

The first cost driver is addressed by the Integration Layer which provides standardised Interfaces (API), Messages (semantic and serialisation), reliable communication protocols.

The second cost driver is covered by the standardised Application Framework, which provides additional services for:

- Resource management to solve high availability issue;
- Central monitoring and configuration to allow one "system" view on the deployed components.

The second of the listed above objectives (enabling of new intelligent functions) is addressed by both IL and AF:

- IL allows easy access to standardised information available in TMS, required by future intelligent functions;
- AF allows development of light weight Apps (not full blown applications) managed by AF-services.

Both components (IL and AF) are very ambitious from the software point of view. The required functionality can be covered by different existing software solutions on the market to a large extent. Unfortunately, neither the existing solutions provide a standardised API nor does an agreed set of data structures for TMS applications exist. Therefore, the overall objective of WP8 is to provide a specification of API and the Canonical Model (see Figure 2.1).

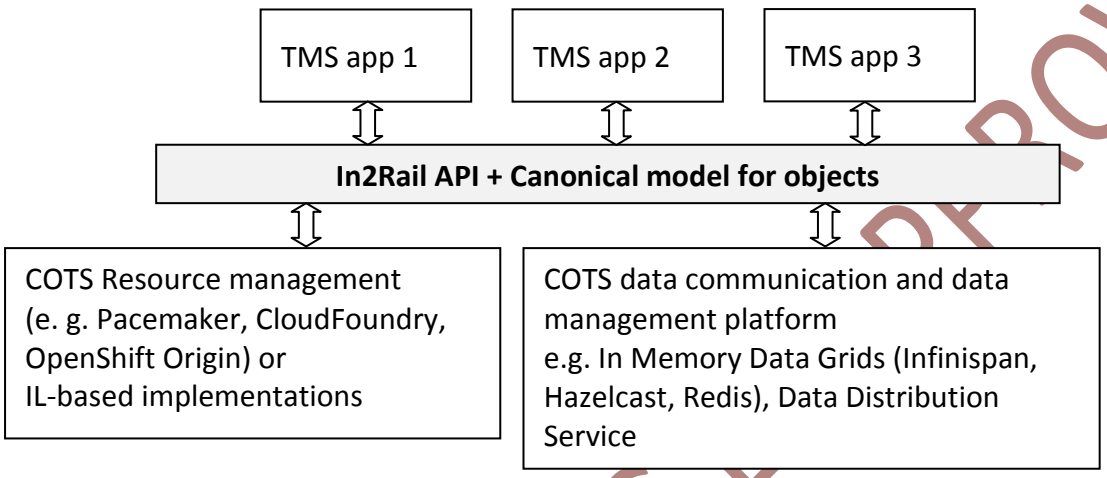

Figure 2.1: Integration of TMS Applications into In2Rail platform (IL and AF) providing a Wrapper to COTS-software and Canonical Data model

The specification is split into two modules:

- The Integration Layer specifies:
  - API for accessing the communication platform and
  - Canonical data model for communication with TMS applications.
- The Application Framework specifies only the canonical model for resource management.

To give an idea about the functional separation, an existing application server in Java-domain can be considered:

- The IL corresponds to the JMS (Java Messaging Service) of the application server;
- The AF corresponds to all the other parts – Registry, Container-management, Cluster-Management, etc.

As the separation of concepts into Communication and Application Platform already takes place in real world, the same approach seems to be reasonable in the context of the IN2RAIL project.

3 Introduction

The first assumption for the specification of the Application Framework (AF) is, that the Integration Layer (IL) is defined and ready to use. The AF shall be based on the IL and provide additional services with the aim of offering a plug-and-play platform for light weight applications.

The Application Framework provides a "system view" on a set of light weight applications originating from different vendors. It allows systematic:

- Deployment;
- Configuration;
- Monitoring;
- management including fail-over, maintenance, load balancing, etc.;
- Removal;

of the applications managed by the AF.

In the following, the combination IL + AF is referenced as the IN2RAIL-platform.

Let us consider the process of evolution from current TMS-applications to the IN2RAIL-platform-based future. Current TMS applications are full blown software products containing their own proprietary resource management, therefore the first step would be integration of existing applications by means of IL (see Figure 3.1).

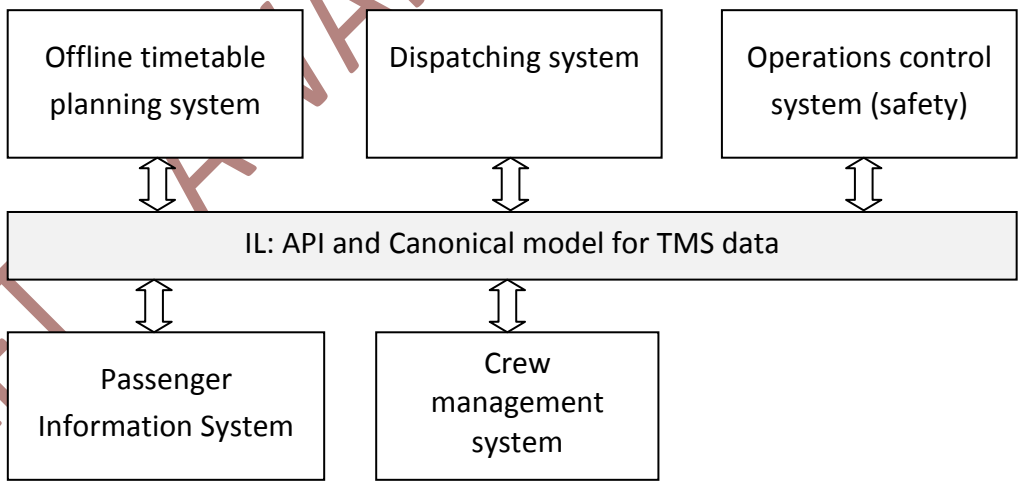

Figure 3.1: Integration of legacy systems by the mean of IL only

As long as the cooperating systems manage their availability themselves and support different departments with their own system administrations, a central management, monitoring and configuration of the entire system is often not required. In this case the Application Framework can be omitted.

In the future it is assumed that TMS will be created by many Apps representing a form of microservices design pattern and originating from different vendors (see Figure 3.2). In this case the usage of an Application Framework is inevitable.

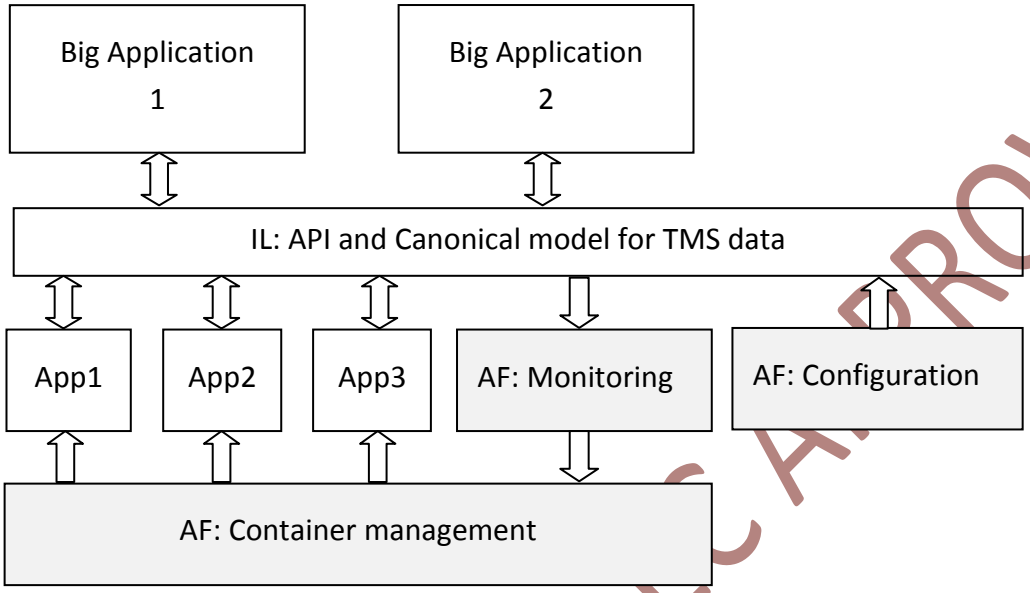

Figure 3.2: Integration of light-weight Apps by the means of AF

In such a deployment the Application Framework manages the light-weight Apps and provides central monitoring and configuration functionality.

4 Application Framework requirements

To identify the requirements of the Application Framework (AF) first the applications that shall be managed by the AF and used integration patterns were analysed.

4.1 Structure of TMS applications

The TMS applications are structured around the data in real time. The major data blocks are shown in Figure 4.1.

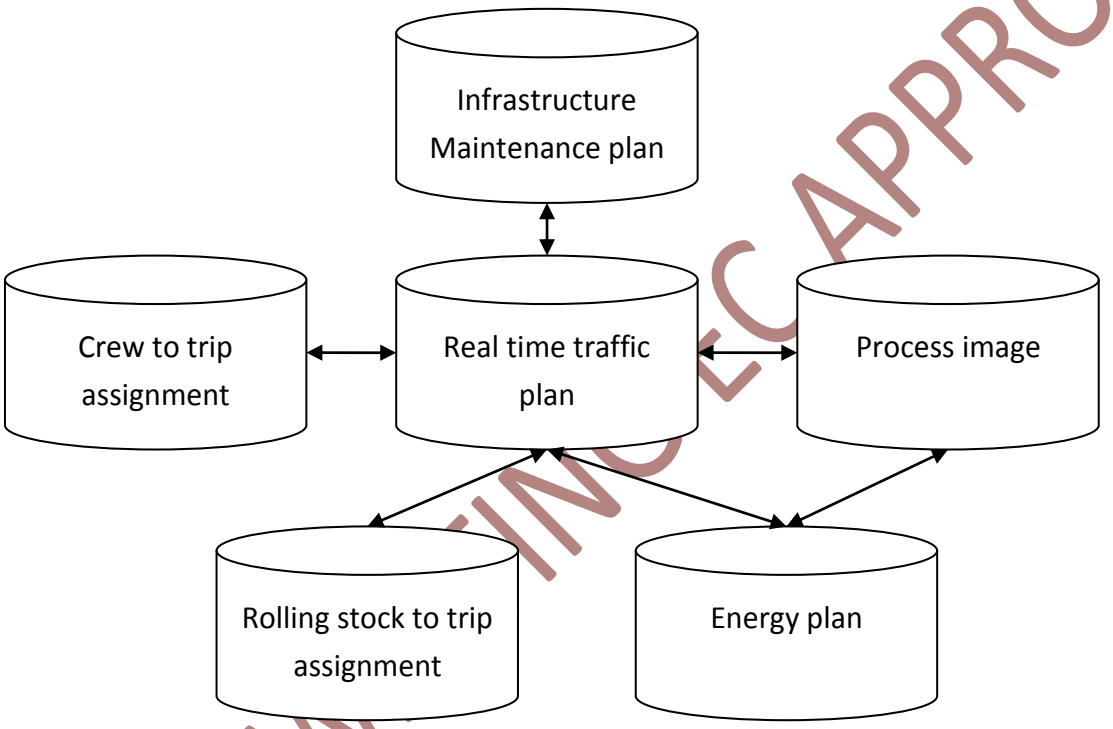

Figure 4.1: Main data blocks in a TMS

The TMS applications provide following general functionality:

- Represent the data from the above modules to the users (10-50 per control center);
- Receive user input (routing commands, timetable modifications, workflows with RUs etc.), implement input validation by calculating forecasts and provide decision proposals.

The representation of such large amount of data in the frame of high performance graphics on (often) eight monitors currently requires installation of a client computer with sufficient hardware resources (e.g. Intel i3-processor with 16 GB RAM). The client hardware often supports some part of safety functions, e.g. providing safe track view.

The server functionality is typically divided into safety relevant and timetable related parts. The number of servers is constant during the day.

In many existing installations the operators are working on the shared Real Time Traffic Plan (RTTP) containing planned train movements, train connections, planned possessions and restrictions and covers between 8 and 72h. This RTTP for a medium sized network is 1-2 GByte. Any modifications of RTTP shall be automatically analysed by a forecast algorithm and the result of such analysis shall be represented to the users.

Even with this simple case the amount of network communication is quite large: one shared forecast with about 1-2 GBytes shall be distributed to 30-50 workstations during less than one second. Often basic analysis is done on the operator's workstation and the detailed forecasting is done after application of the changes on the shared RTTP on one powerful server, managing RTTP.

To be able to manage such amounts of data each workstation is typically provided with a cache of the shared RTTP and only delta-information is transmitted.

To be able to represent the current field state the operator's workstation is provided with a cache of the Process Image (train positions, switch positions, currently active restrictions, possessions etc.).

In future applications (e.g. in recent tender from BaneNOR) the operators will be able to analyse their modifications in a sandbox, i.e. each operator will have her own "copy" of RTTP and shared Process Image. With current architecture of the operator's workstation the AF shall be able to manage not only "anonymous" hardware executing server functions, but also the dedicated nodes for each workstation.

A general architectural pattern for TMS-services is shown in Figure 4.2. The TMS-service observes a set of Topics (1-3) in advance, receives its requests from one of the topics and publishes its results on some other topics.

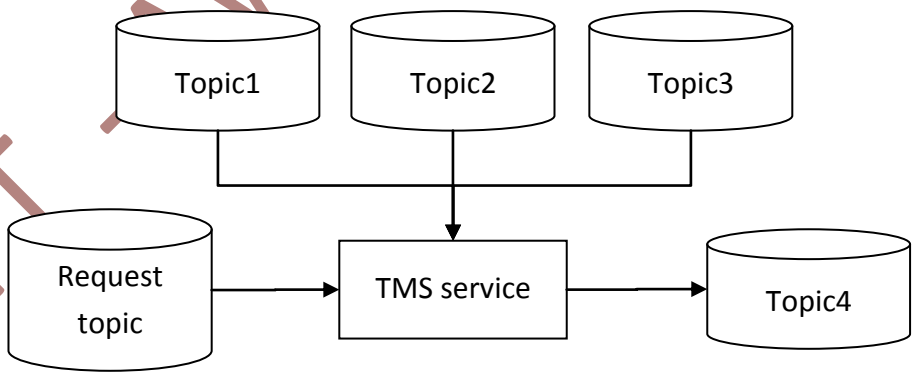

Figure 4.2: General architecture of a TMS service

4.2 Integration patterns in Application Framework

As the second step the integration patterns available for the applications were analysed.

4.2.1 Integration by API

This kind of integration provides the highest performance – instead of sending messages over the localhost-interface direct programming calls are applied. Especially for small messages the performance improvement is by a factor of twenty.

Historically one of the main concerns during specification of CORBA [OMG CORBA] was the transparency between local and remote object invocation: the client application calls a method of an object without knowing, if it runs in the same process, on the same node or on a remove machine. This allows the flexibility for allocating the modules on different nodes (remote), starting as independent programs (remote) or loading as dynamic libraries (local) per configuration. This logic is still used in several Application Servers, as they use CORBA protocols for internal communication.

A similar implementation is provided by ZeroC-Internet Communication Engine (ICE) with the module IceBox (see Figure 4.3).

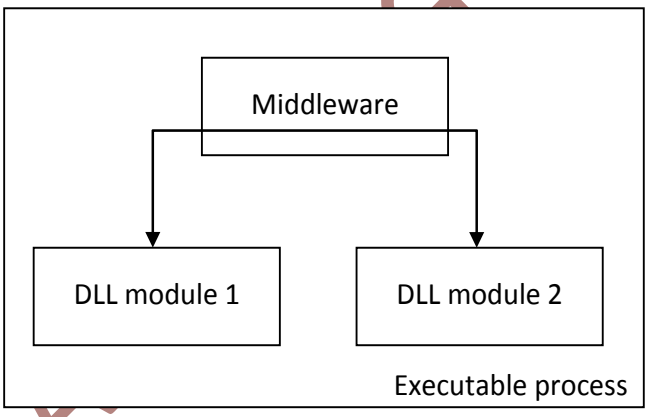

Figure 4.3: Collocated invocations inside of one executable in ZeroC-ICE

The IceBox loads dynamic libraries and communicates with them using “remote” protocol. All modules loaded into one IceBox communicate with each other by means of local calls.

The DDS implementation OpenSplice provides a similar way for integrating services.

In TMS applications a typical use case is the run time calculation module (see Figure 4.4).

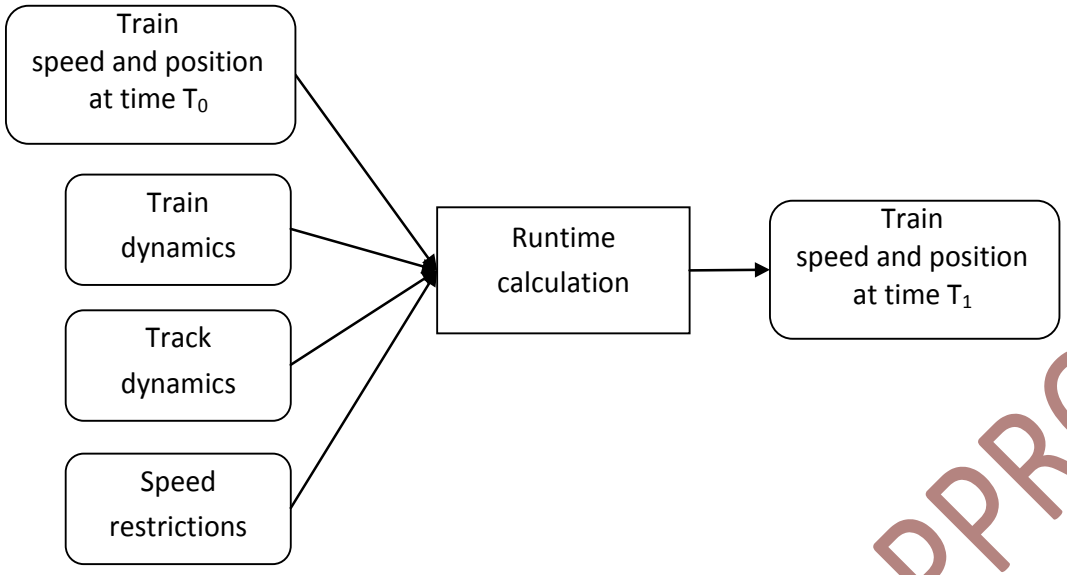

Figure 4.4: Integration of Runtime calculation service by API

Often each Railway company has its own train dynamics database with preferred run time calculation algorithms, which often already exists a dynamic library.

It is assumed that the API based integration is limited to special cases like the example shown above. The Integration Layer will provide enough bandwidth to enable Integration of general TMS applications. Therefore the API-based integration was excluded from the Application Framework.

4.2.2 Integration of Executables

An executable can be represented as an archive containing all required artefacts to start an application from an operating system by one command. For deployment of such applications packet managers are used, e. g. apt (dpkg) in Debian based Linux.

It is assumed that creating a Container from an Executable is a relatively easy task. In comparison to bare executable the container technology provides so many advantages, especially a much better isolation of the concurrently running applications in CPU, RAM and Network usage, that it was decided to exclude Integration of Executables from Application Framework.

4.2.3 Integration of Virtual Machines

At the current state of IT the Virtual Machines represents a highly reliable technology for isolation of Applications. Assuming a multivendor TMS the Infrastructure Manager can have a full control on the software modules running in his data centre.

It is assumed the VM to be in the near future the most used way for integration of TMS applications, therefore it shall be included into the Application Framework.

4.2.4 Integration of Containers

Containers represent a light weight virtualisation technology, where the software in containers is running under the same operating system. The isolation of the processes is implemented by means of the host operating system. In the context of Open Container Initiative the interfaces for the management of containers will be standardised. Several platforms on the market support the management of Container Integration [Openshift 2017], [CloudFoundry 2017], [Kubernetes2017] having different approaches of integration.

It is assumed that in the future the innovative TMS applications with short deployment cycles will be integrated by Container Virtualisation. Therefore they should be included into the Application Framework.

4.3 Requirements

It is assumed that TMS applications will be provided either as Virtual Machines or as Container therefore the word Application means an application delivered as Virtual Machine or Container.

It is assumed that one TMS application can contain one or more TMS services.

The requirements from TMS applications are:

- The TMS services are stateful with a state of 1-3 GB in RAM. They must be provided with caches before they can start to handle user requests;
- The required failover time in some projects is smaller, than the time needed for cache transmission. Therefore if required the services must be pre-started in hot-standby mode;
- It must be ensured, that only one instance of a service is able to write to specific topic on IL to ensure that decisions are only made in one place. This functionality can be implemented on the services themselves or provided by Application Framework.

The responsibility of the Application Framework is:

- Deploy TMS application on some execution environment;
- Manage start on demand by observing “request topics” and by timer;
- Manage service failover by means of hot and warm standby;
- Manage service leader: from several running instances of a service with identically configured inputs and outputs only one is allowed to write to the output topics. A new leader is selected only if the previous leader has crashed;
- Application Framework manages TMS applications on backend servers and dedicated operator’s workstations as well.

4.4 Application Framework Functions and Constituents

The Application Framework is a product which will be provided by an external vendor. The vendor can take any existing implementation on the market as a basis and extend it as needed to be able to fulfil the requirements of the AF.

In the context of this document it is intended to specify only a set of data structures, needed to represent the desired state of the managed TMS services and to observe the current state of TMS services. The responsibility of the Application Framework as a product is to implement the desired state and to provide the current state of TMS in specified data structures on the Integration Layer.

In the AF-data structure the term Node is used, which is either the data centre cluster for running Virtual Machines or Containers, or the operator's workstation.

To provide monitoring functionality independent from used COTS-Container-Management and allow tight integration into TMS, the states of the Nodes, Applications, and Services shall be published on Integration Layer using standardised data structures.

5 Private cloud management software on the market

Assuming that in the near future the TMS applications will be integrated into Application Framework as Virtual Machines the Application Framework shall be able to distribute and manage them on a cluster of nodes – cloud. The cloud could be private in the case where the nodes are located and belonging to the Infrastructure Manager (IM) or the IM could rent a cloud from an external cloud provider.

There are several solutions on the market for building and managing private clouds:

- System Center Virtual Machine Manager (VMM) as a management tool for Hyper-V from Microsoft [MS VMM 2017];
- vCloud Suite from VMware [VMW 2017];
- CloudPlatform from Accelerite [ACCP 2017] (former Citrix product);
- CloudForms from Red Hat [RH CF 2017].

In addition to these tools there are numerous third-party products that provide additional capabilities, such as multi-platform support, the ability to reclaim wasted space, and virtual machine monitoring for optimal performance. Examples are as follows:-

- VMTurbo Operations Manager Cloud Edition [VMT 2017];
- Embotics vCommander Enterprise Cloud Management Software [EMB 2017];
- Solarwinds Virtualization Manager [SWVM 2017].

Considering the volume of different tools it is assumed that the AF-Vendor would be able to create a small wrapper, which is able to use the desired state represented on the Integration Layer to create commands on the Command Line Interface provided by most of these cloud management tools.

6 Container management on the market

Containers include the application and all of its dependencies - but share the kernel with other containers, running as isolated processes in user space on the host operating system. As an application can be provided in form of container the next sections show the container management software on the market.

6.1 Cloud Foundry

The open source project CloudFoundry (CF) provides a Platform as a service (see Figure 6.1) [Winn 2017]. It is located on top of an "Infrastructure as a service"-provider and simplifies container management. The system integrator can push a container into CF, configure connections to CF-services (like databases, middleware, etc.) and define how many instances of the container shall run. The CF ensures that the configuration is implemented even in case of failures (hardware or software).

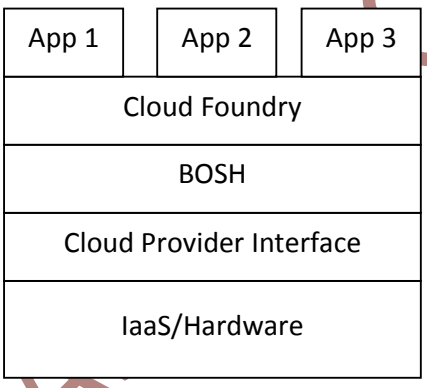

Figure 6.1: Cloud Foundry layers [Winn 2017]

CF defined an interface to Infrastructure as a Service provider (Cloud Provider Interface), which allows usage of many different hardware providers as well as dedicated hardware. The BOSH module manages virtual machines (creating, movement, deleting) and installation of the CF-components as distinct software (e.g. Databases, User management, etc.).

Cloud Foundry supports the software development process including building, deploying, monitoring, managing user access and authentication, logging, failover etc. The applications can be "pushed" to the CF either as a source code or as a container.

The CloudFoundry requires from the Application a set of requirements – the twelve factors contract. Some of them are listed here:

- Stateless – containers are not allowed to store anything on container disk;
- Logging must be implemented through standard streams;
- Configuration of the container at runtime is done by environmental variables.

CloudFoundry provides Applications with additional services using environment variables:

- Data base management systems (Redis, MySql, Postgresql);
- Message Bus (NATS);
- In Memory Data Grid (e.g. Hazelcast on Pivotal CloudFoundry).

Additional services can be integrated by CF-provider as required. The services are started and managed on dedicated Virtual Machines by the Bosh module. The Applications implement only the business logic – the data management modules are provided by the CloudFoundry.

6.2 Openshift

Openshift is a similar project by Redhat Inc [Openshift 2017]. It has several subprojects:

- Openshift online with commercial platform provided by Redhat;
- Openshift origin is an open source project.

It is based on Container management software Kubernetes by Google. It tries to make container creation and management transparent to the software development. The developer configures her source code project and manages it in the Openshift project in the same ways as with versioning system (see Figure 6.2). Openshift even uses Git as a part of the interface.

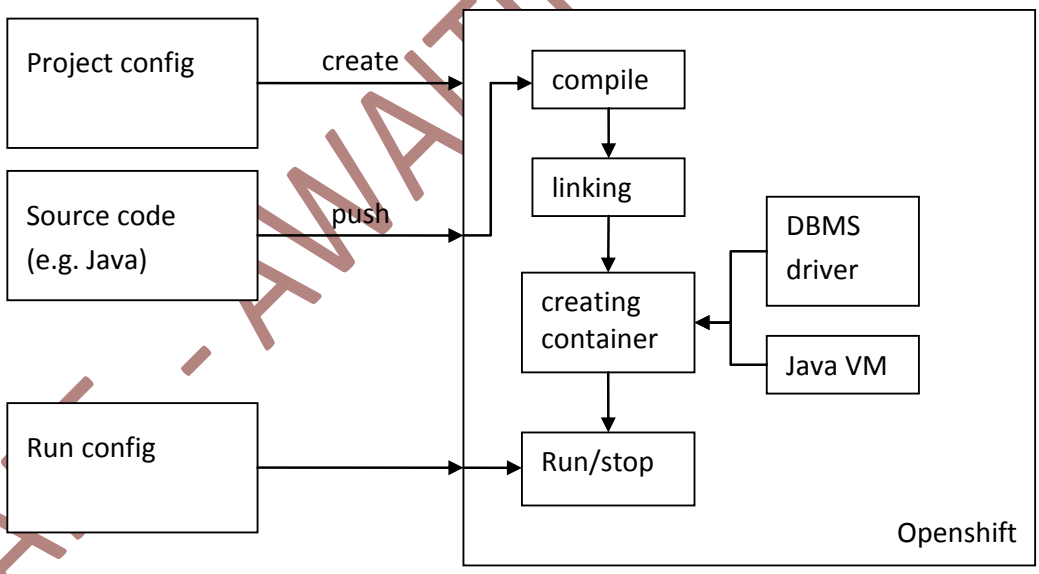

Figure 6.2: Schematic overview of Openshift

In addition it provides commands for monitoring, archiving/backup, movement of projects between development, test and production environments.

6.3 Docker Universal Control Plane

Docker Universal Control Plane (UCP) is an add-on component of the product Docker Enterprise Edition. [Docker 2017]. It provides an enterprise-grade cluster management with the following functionality:

- Monitoring of the cluster nodes (CPU load, memory, services, containers, etc);
- Centralised cluster management (adding/removal nodes);
- Deployment, management, and monitoring of applications and services.

Access to the UCP is controlled by build-in authentication mechanism or by integration of LDAP services.

7 Architectural decisions

It is not the aim of IN2Rail project to specify and develop a new container management within the Application Framework. The objective is to use existing implementations as much as possible and provide a narrow wrapper to separate them from “long-term” applications in the railway domain.

Architectural decisions are derived using the following approach:

- Identification of required functions and grouping them to “functionalities”;
- Identify information flows required by the functions (Topic structures);
- Define data structures needed for implementation of the required functions.

7.1.1 Identification and grouping of functions

The single functions of the Application Framework can be grouped together (see Figure 7.1).

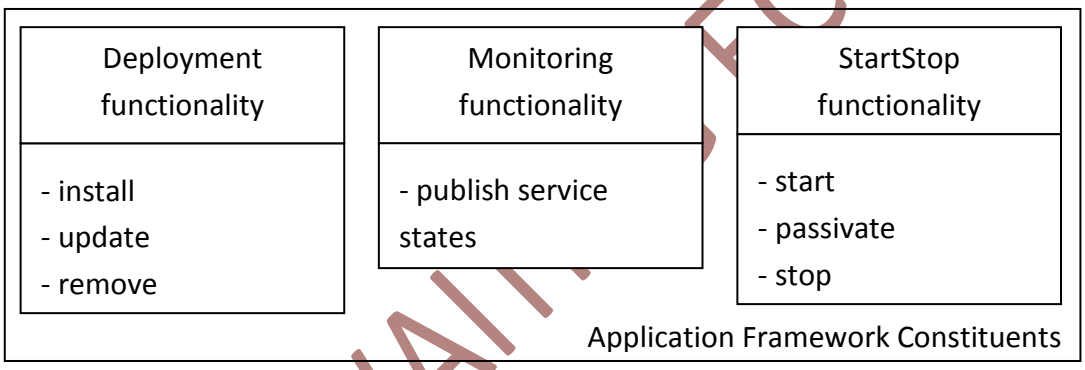

Figure 7.1: Application Framework functionalities

In the context of IN2RAIL the following decisions were made:

- Integration Layer covers all communication aspects between TMS and Application Framework;
- The required data structures (messages) are specified as part of Canonical Data model.

In this sense the AF is an optional add-on to the IL-specification.

7.1.2 Relevant data objects

In the existing projects different words may refer to similar things. Therefore the used terminology shall be introduced before the functionality description.

App: as already mentioned in Section 4.2.3 and 4.2.4 as integration entity either a Virtual Machine or a Container is considered. The name App is used as abbreviation of Application under the meaning VM or Container.

Service: is a functionality which reads and/or writes to Topics on the Integration Layer. It is assumed that one App provides one or more services.

Node: an execution unit – can be a computer (e.g. an operator's workstation), or the entire data centre, if AF is managed by existing cloud based projects (CloudFoundry, Openshift, Kubernetes, etc.).

7.1.3 Relevant object states

App

The concept of App has two aspects:

- Deployment: create an accessible copy, which can be started on a Node;
- Start/Stop: actual activation of an App, which is similar to starting/stopping of an executable.

Service

As the service is a part of an App, it does not require special handling for deployment. The only relevant states are:

- Started;
- Stopped;
- Passive – ready to start writing to Topics (hot standby).

Node

To be able to handle cloud infrastructure it is assumed, that the Application Framework is able to start and stop nodes according to some logic for the sake of

- Energy saving;
- Cost saving in the cloud environment;
- Hardware maintenance;
- Tests.

In the following the functionalities required from Application Framework were analysed. As a result data topics on Integration Layer needed for implemention of the Application Framework were provided.

In the following many the states and objects are noted by their identifier. The "CamelCase" notation is used for combing words to an identifier; e. g. the Topic with identifier equal to DeploymentState contains data structures representing the deployment state of the applications.

7.1.4 Deployment functionality

The deployment function has the following responsibilities:

- Ensuring that the configured Applications are installed on the configured Nodes from configured repositories;
- Handling of the changed software version shall depending on the configuration:
  - if required, the old version is uninstalled before installing the new version,
  - the new version of an application shall be installed,
  - if required, the old version is uninstalled after installation,
  - concurrent usage of two versions in one Application Framework must be supported;
- Uninstalling the software first, requests to stop all running services, observation of this behaviour, and if finished, removal/archiving of the old version.

The deployment function shall observe the "desired deployment state" on the node and compare it with the actual node state (see Figure 7.2). As soon as a discrepancy occurs, it shall implement changes in each node. A central logic is not required. Therefore, on each node a Deployment process can be started.

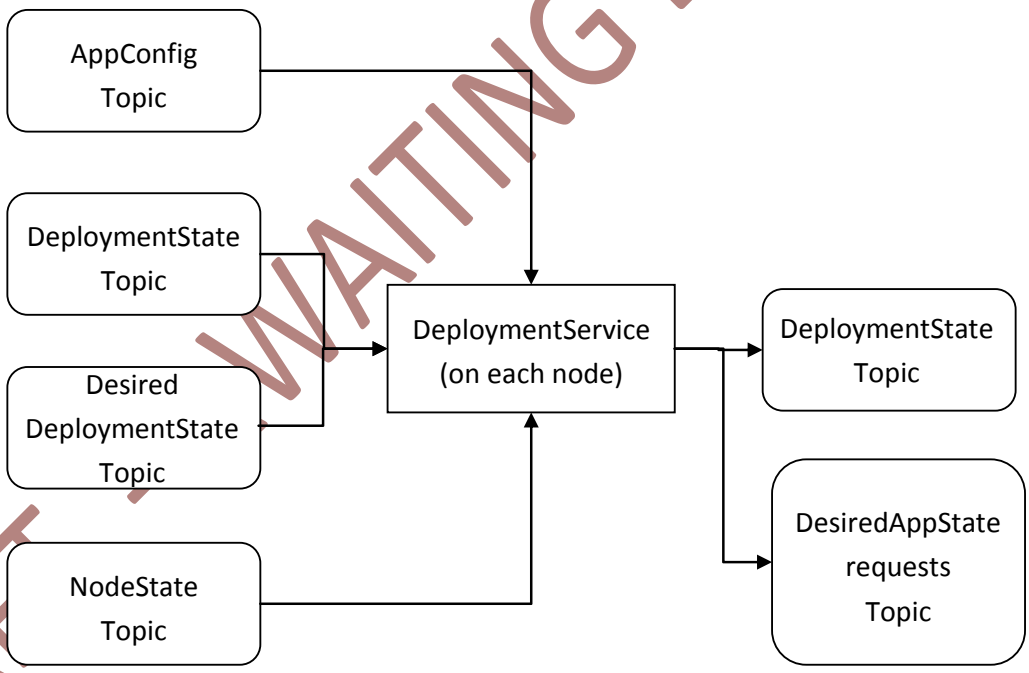

Figure 7.2: Information flow for Deployment Service

The DeploymentService is running on each node and subscribes to the following Topics:

- AppConfig Topic contains configuration of the App, like required OS, RAM, CPU, provided services etc.;
- DeploymentState Topic contains currently installed Apps assigned to Nodes;
- DesiredDeploymentState Topic contains the next deployment state to be achieved by AF;

- NodeState Topic contains the load and life statistics of each Node managed by AF (running since DateTime, current CPU load, free RAM, etc.)

The DeploymentService writes to:

- DeploymentState Topic to publish the newly installed/uninstalled Apps on the node managed by DeploymentService-instance;
- DesiredAppState-Requests Topic is used to publish request to stop Apps on specific nodes before uninstalling Apps on it.

The detailed use cases are specified in D8.7.

7.1.5 App-StartStop functionality

The App-StartStop function shall start and stop deployed Apps depending on:

- Desired AppStates requests coming from outside (user command, deployment service etc.);
- Data published on configured topics;
- Timeouts.

In case of problems (e.g. missing access rights, wrong data types, etc.) the App publishes the errors on App-StartStop-Errors topic.

A crucial functionality represents a “singleton”-pattern – the ServiceStartStop function shall ensure that only one instance of a service is able to publish in IL, if configured so. In the case of a program crash it shall decide on the node where the function shall start.

A typical use case for a singleton is a service collecting timetable change requests from operators, validating them and putting them into a unique sequence. Two such active services could create different “sequences” as they receive modification requests asynchronously.

Potentially singleton function can be implemented by a distributed consensus algorithm inside of the Apps [Cachin et.al.2006]. To simplify the implementation it is assumed, the logic behind StartStop-Decisions will be active in only one central instance (StartStop Controller). The central instance will publish its decisions (see Figure 7.3), which will be implemented by special processes on each Node (AppStartStop Service) (see Figure 7.4).

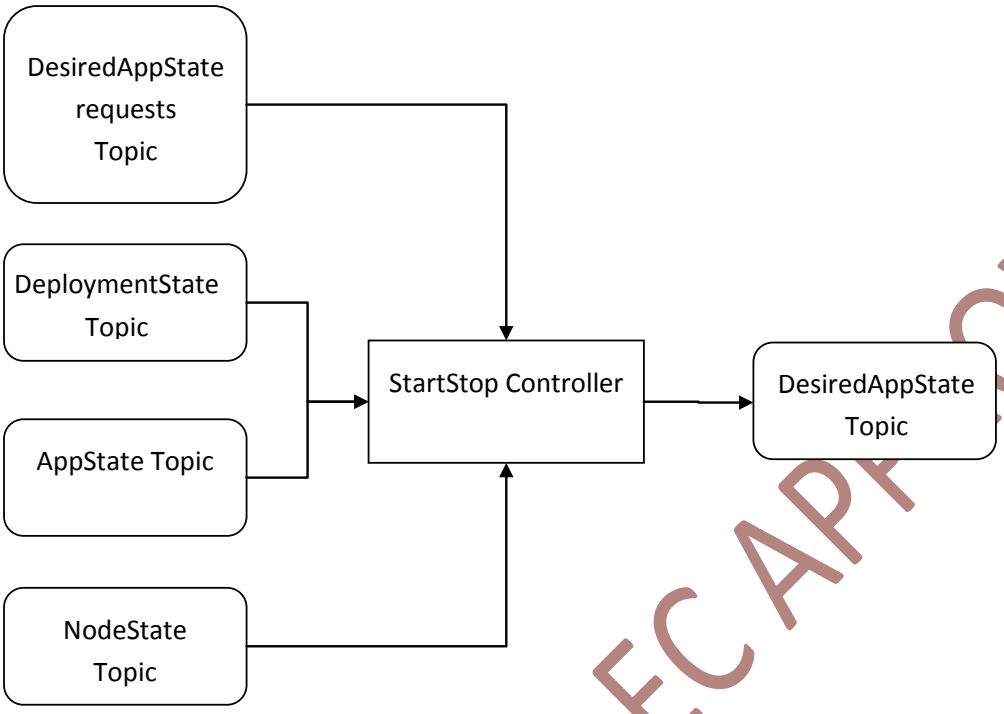

Figure 7.3: Information flow for StartStop Controller deciding which App and service shall run on which Node

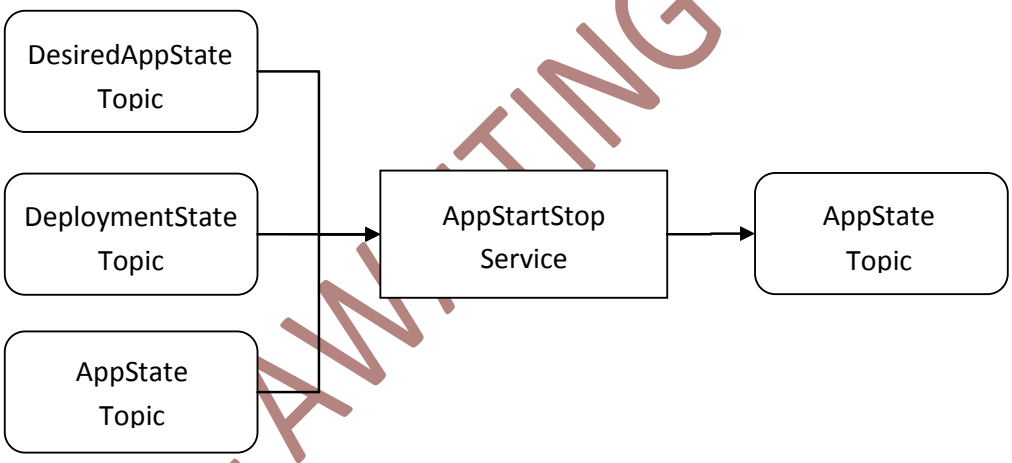

Figure 7.4: Information flow for AppStartStop-Service implementing the Leader-Decisions on each Node

Only configured Apps will be managed by AppStartStop-Function. All the others can be managed independently and cooperate with each other by mean of the IL only.

The services specified here represent only one possible implementation. In the context of this document only a specification of the Information-Topics and Message-Types is required.

7.1.6 Monitoring functionality

Monitoring function publishes states of nodes and services on IL. It shall be independent from the existence of other AF-Functions. Therefore:-

- Apps states shall be published by Apps on AppState-Topic.

Publishing other states:

- Node resource states (Performance monitoring: CPU, Bandwidth, RAM) shall be published by NodeManagement-service;
- DeploymentStates shall be published by each instance of DeploymentService.

7.1.7 Canonical Data Model for AF

The Canonical Model has two main aspects:

- It provides a way to navigate the model by composite relations (child-parent);
- It specifies the attributes in detail to be used for serialisation and data management.

The AF-Part of the Canonical model has its root class "AF" – abbreviation for Application Framework.

It manages Nodes and Apps using configuration and state-data (see Figure 7.5).

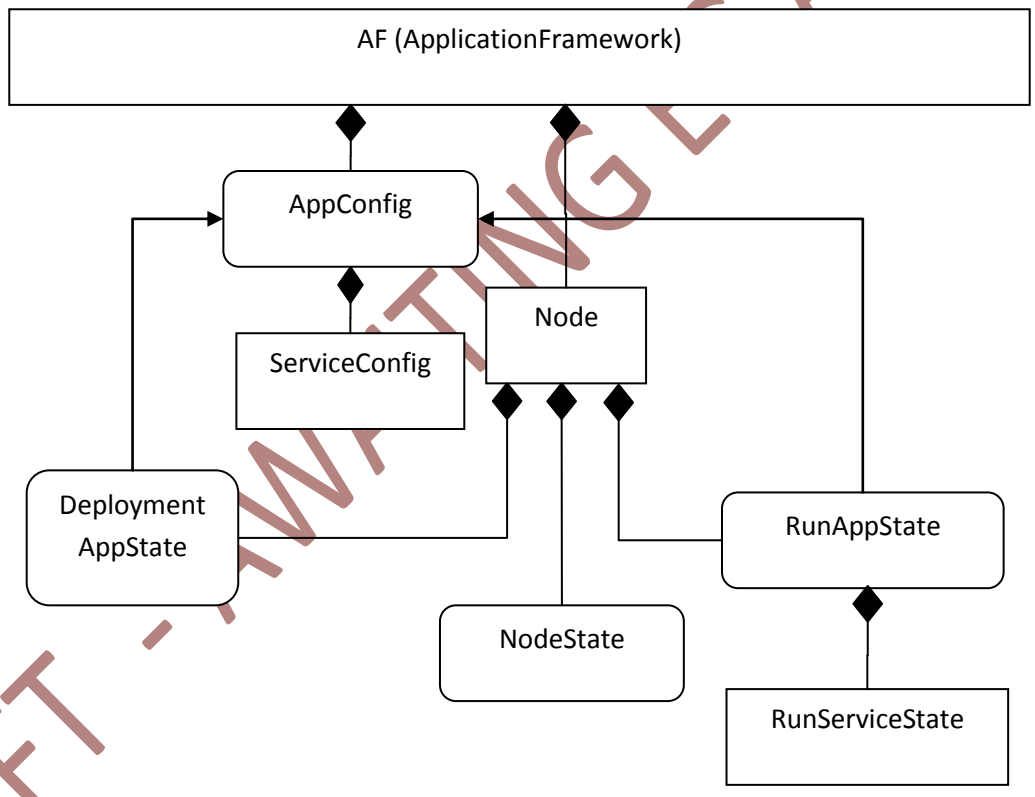

Figure 7.5: UML Class diagram for Deployment, StartStop and Monitoring services

In the class diagram of Figure 7.5 the classes DeploymentAppState, RunAppState and NodeState participate in two composition relations to each Node:

- As currentXXXState and
- As desiredXXXState.

That means the Node has e. g. an attribute with name currentRunAppState and desirecRunAppState referencing objects of the same Class RunAppState. The same is valid for NodeState, DeploymentAppState.

This reduces the number of required classes and simplifies the services comparing desired and current states.

In the following the classes in the AF-CDM will be specified in detail.

7.1.7.1 AF.AppConfig

For an Apps different attributes shall represent in CM:

- Version;
- Vendor;
- Available services (s. ServiceConfig);
- Runtime dependencies from other services;
- Deployment dependencies from other Apps (typically not needed, as an App shall be able to operate alone);
- Node requirements (OS, RAM, CPU, Bandwidth);
- Application UUID.

7.1.7.2 AF.AppConfig.ServiceConfig

A service represents a functional unite which reads from and writes to Topics in the Integration Layer.

A ServiceConfig shall specify:

- Topics which it is able to listen to including Quality of services (QoS);
- Topics which it is able to write to including QoS;
- Starting strategy: singleton vs. non-singleton. Typically it shall ensure that only one instance is able to write to a specific topic – as long as “writing” topics are different, several instances of “singleton”-services can be started;
- Load-balancing strategy: any, max networks, max CPU, max RAM or weighting factors for these aspects;
- Logging level.

The ServiceConfig can reference Topics specified in the Integration Layer (standard topics) or specify its own Topics as well as data types.

7.1.7.3 Node.desiredDeploymentAppState

The NodeConfig represents the desired state of the Node and shall contain:

- List of Apps to be deployed with the deployment state (installed, archived).

7.1.7.4 Node.currentDeploymentAppState

It represents the installation status of an App on a Node. Possible values are installed/installing/deinstalling/archiving/archived.

7.1.7.5 Node.currentRunAppState.ServiceState

It represents the current state of the service: running/stopped/passive. The state can be extended by:

- runningState: running/stopped/passive;
- Load % (activeTime/totalTime);
- Active/Inactive;
- Communication statistics (per Topic);
- Current Logging level.

7.1.7.6 Node.desiredRunAppState.ServiceState

It represents the desired service states assigned to nodes – which service shall be in which state at which node. The values are coming either from NodeManagementLeader or e.g. an administrator UI/command line interface.

Using this class independent from the Node allows a request for new Nodes to be started in a Cloud, if such functionality is required.

7.1.7.7 Node.currentNodeState

It represents a load state of the Node to be used by NodeManagementLeader for load balancing. It shall contain:

- CPU load, %;
- Available RAM, GB;
- Available persistence space (for logging/deployment);
- Used bandwidth MB/sec;
- Available bandwidth MB/sec.

7.1.7.8 Node

Besides the mentioned composite-attributes it contains only the ID and allows specification of simple keys /AF/node[238]/nodestate. The main purpose is to simplify subscription/management of the key-values in IL.

8 Integration of TMS Applications in one user interface

8.1 Objective of User Interface Integration

The main functionality of AF is container management. Integration of TMS-Application into one user interface does not belong directly to the container management. The missing approach for UI integration can prevent the splitting of TMS-functionality into modules and thereby making IL and AF obsolete. In recent tenders and projects the IMs insisted on having one TMS-system with shared menus and navigation bars, consistent navigation between views and a common look and feel.

Considering the current market, the TMS functionality is split in bigger modules – each covering all requirements of the corresponding user of the module: operator, dispatcher, timetable planner, RU-portal, maintenance manager, etc. In many cases, these roles require non-overlapping functionalities; therefore, they can be managed by software products from different vendors with a slightly different look and feel.

It is assumed that the future TMS-Applications will concentrate not only on traffic control, but also on traffic optimisation. Supported by optimisation algorithms the human users will have to consider more aspects in their decisions. More aspects automatically mean having more integrated applications in one UI, e.g. dispatching decisions could depend on the current state of the energy system requiring UI-integration of both applications.

In the following sections possibilities and requirements of integrating TMS applications into one UI were analysed. The issue of integration is not restricted to TMS – as the entire IT industry has been faced with it for the last 30 years. One of the well known solutions is COM/ActiveX-technology integrated in Microsoft Windows since 1996. It is not intended to solve this issue in this deliverable with a solution covering several programming languages, running on different operation systems and covering rising use of cloud technology to provide remote UIs.

The objective is to provide an optional, basic integration mean for UI integration of future TMS-applications.

8.2 Existing application integration patterns

An application is usually specified in three layers: the data layer, the business logic and the user interface layer.

The integration of applications in general can be realised in three different levels (see also Figure 8.1):

a. Integration on the data source level, this means the data sources are integrated and a common business logic and user interface is developed;

b. Integration on the business logic level, this means an integration of the business logic and on top of the integration a common user interface is developed;
c. Integration on the user interface level, this means an integration of several user interfaces.

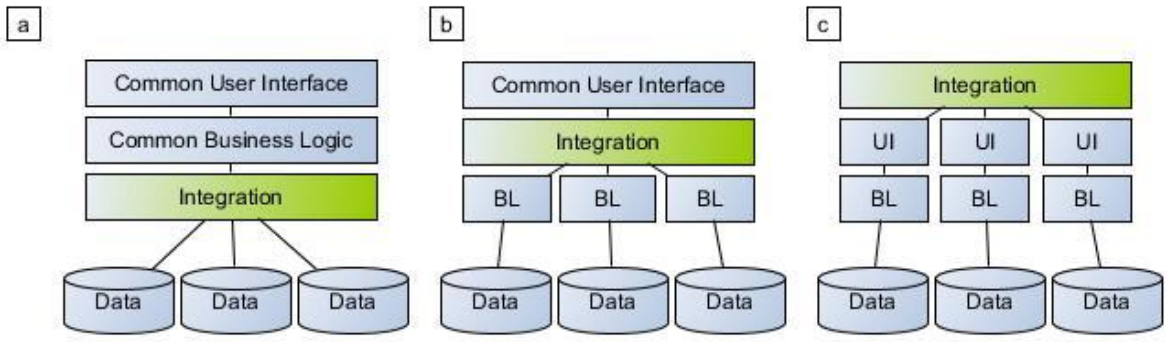

Figure 8.1: Different levels of integration

Regarding the TMS integration, the following solutions can be mentioned.

8.2.1 Approach a

Integration on the data source level is implemented by IL already as all applications use shared data source provided by it.

8.2.2 Approach b

Integration of different business logics into one common user interface seems to be the next step in evolution of TMS applications, where existing "big" applications with Rich UIs are extended by Apps coming from different vendors (see Figure 8.2). Especially decision support functions could be implemented in this way.

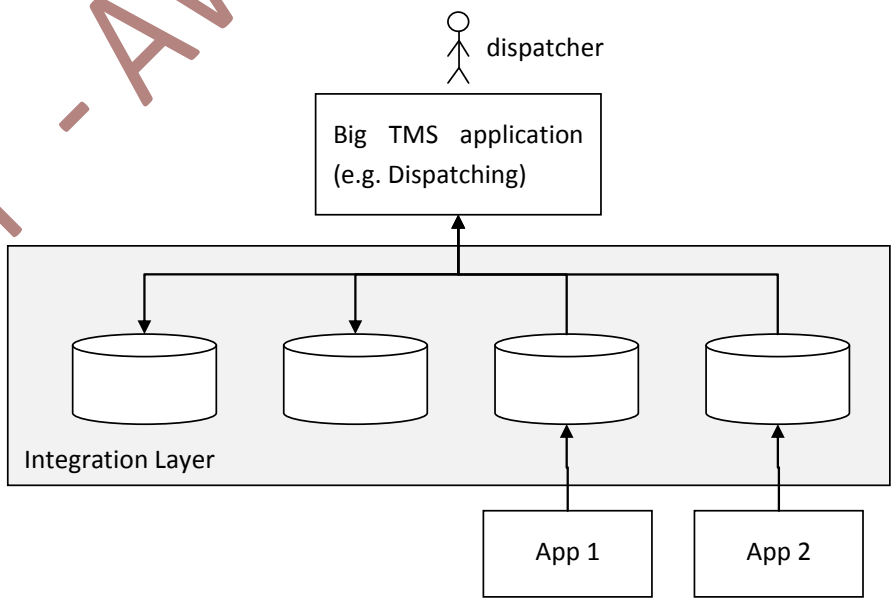

Figure 8.2: Integration of "blackboxes" providing business logic only

A reasonable way could be the specialisation of some vendors to deliver only UI and keeping all business functions separated running in the AF (see Figure 8.3). To enable this kind of integration no additional specifications are required:the IL provides Topics with data to be represented and Topics for modification requests. The UI application shall be able to represent the first and to write to the second Topics. The information transformation between these Topics can be done by backend applications without UI. This approach is possible because many Views in TMS are more or less de facto standardised as follows in the next sections.

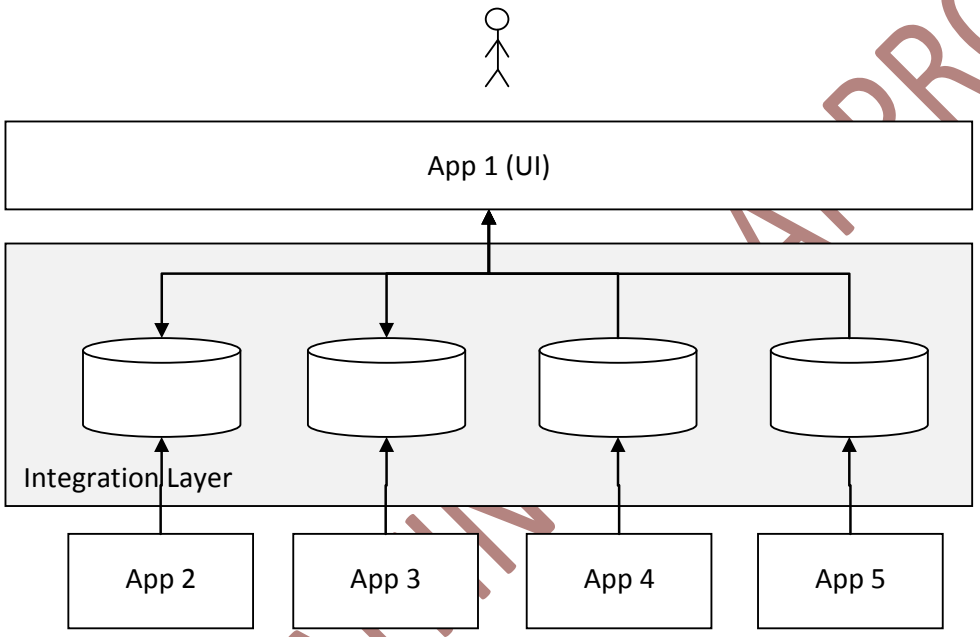

Figure 8.3: UI-App to represent topics and publish requests

8.2.3 Approach c

The last kind of integration is the most challenging one. To ensure a full UI integration the IT industry created the concept of a RCP (Rich Client Platform). Existing implementations are limited to one programming language enabling the integration on the binary level: one module calls another module synchronously using binary calls. It is not intended to standardise such infrastructure, but only to provide additional topics on IL to allow asynchronous information exchange. This restriction influences possible UI integration patterns quite strongly.

In the following common views in TMS to identify required integration patterns are shown.

8.3 Common views and screens in TMS

Within Task 7.2 of In2Rail the Standardised Operators Workstation has already been specified. During this study the required hardware of a future TMS as well as the views of the user interface for specific roles have been addressed.

In deliverable 7.3, it is proposed that the standard TM workstation consists of three screens, named:

a. Overview screen;
b. Close-up screen;
c. Portable screen.

In which every screen has its own layout structure.

8.3.1 Overview screen

The overview screen is the main screen to display available information. The layout of views is pre-defined for each role, but the layout of the views is configurable.

8.3.2 Close-up screen

The close-up screen is a touchscreen and, next to keyboard and mouse, it should be used to control the overview screen. It could be used to configure the size of the views and the information displayed in the views. As in the overview screen the views are pre-defined per role.

8.3.3 Portable screen

The portable screen is an addition to the close-up screen. It always shows the same information as the close-up screen. If the portable screen is docked in the docking station the display is disabled. Only when the user needs to carry information for discussion, the portable screen can be taken away to display the same information as the close-up screen.

8.4 The views that shall be available for a standardised operator's workstation are also described in D7.3 and part of it shown in the Table 10.1 in Appendix A. The full table is shown in D7.3 Chapter 4.10.4 "Details of Views". The views within one screen can be available as single user interfaces from different vendors, but they should have the same look and feel.Required UI integration patterns

To analyse integration patterns two concepts shall be separated:

- **UI Toolkit**, like SWT, Qt, JavaFX, and Motif with the main objective to draw graphical primitives like Buttons/Text/Widgets/etc. on the screen and 2D and 3D contents;
- **Rich Client Platform** providing all building blocks for creating an application from plugins. Typically, an RCP uses the UI Toolkit to harmonize UI representation of integrated Plugins.

To identify the required integration patterns one of the most successful RCPs on the market has been analysed – Eclipse. Other UI-Toolkits enable structuring bigger applications in

modules (plugins) as well, but only Eclipse is a basis for hundreds of independent projects integrated into many applications.

The architecture of Eclipse RCP is shown in Figure 8.4.

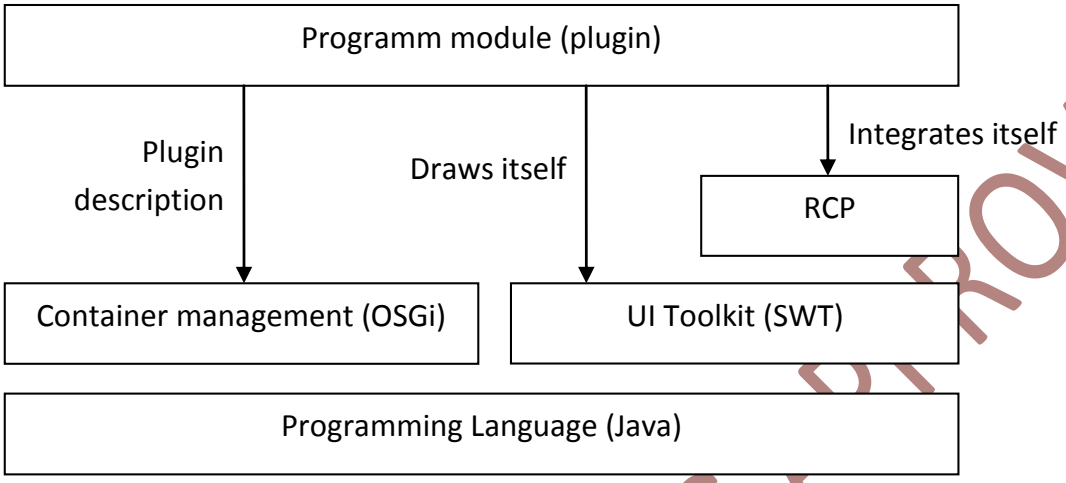

Figure 8.4: Architecture overview of Eclipse RCP

Eclipse is based on a common programming language enabling direct program calls, information hiding by “Interface”-concept, and synchronous logic. The plugins are delivered in form of a JAR-Archive and called a “bundle”. They use UI Toolkit directly for management of their representation.

The basic integration pattern in Eclipse is quite simple:

- One module provides extension points;
- Other modules provide extensions;
- The extension points and extensions are specified as textual documents in XML-format, enabling application creation by configuration.

The main UI integration pattern represents the concept of Action.

An action is a class containing:

- Label (to be presented in Menus);
- Icon (to be presented in Buttons/Tool bars);
- Description text to provide Help/Tooltip information;
- State – enabled/disabled.

In order to provide the State, Actions are often connected to the concept of Selection. Each Window has its own Selection-Object, which is a list of currently selected objects. An Action can subscribe to it and decide if it is enabled or not. For instance an action showing detailed information about a signal observes selected elements and is active only if a signal is selected for which it has detailed information.

In Eclipse there is a distinct plugin representing an Application, provided by the application vendor. It reserves sections for further extensions in its:

- Menu bars;
- Tool bars;
- Context menus.

and publishes them as extension points. Other applications provide extensions in the form of Actions-sets.

Actions from extending applications are provided to the user in Menus and Toolbars. If selected, the RCP calls the function Action.run. Here, the extending application decides what to do:

- Ask the RCP to open a new View or Editor;
- Change selection in one of the previously open views;
- Do some calculation, etc.

A UML class diagram in Figure 8.5 represents the integration pattern.

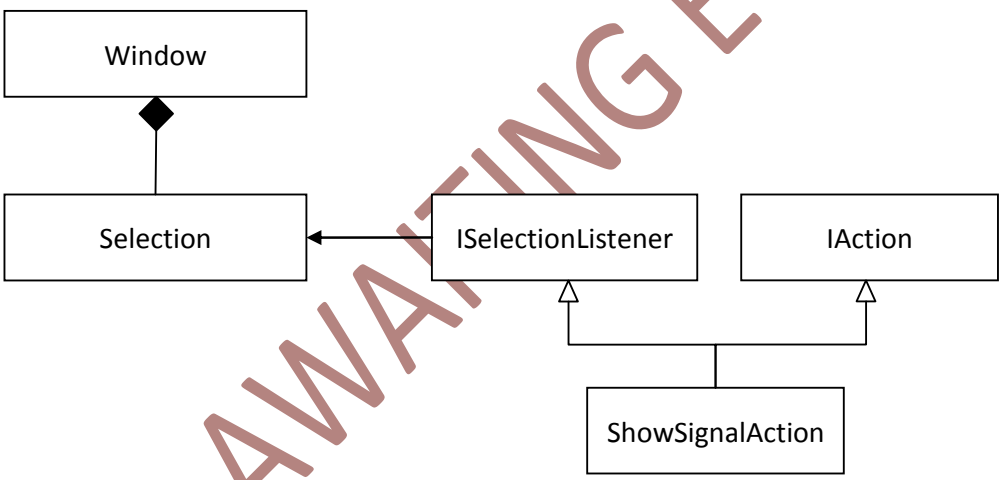

Figure 8.5: UML Class diagram for UI Integration in Eclipse

In the next section the mentioned Eclipse Integration patterns are mapped into the infrastructure provided by AF and IL.

8.5 UI integration approach

The most crucial aspect missing in IL/AF in comparison to Eclipse is a common RCP. In Eclipse the modules use it for:

- Integration of their content into one Layout;
- Representation of shared Buttons, Icons and Text fields;
- Exchange of context information, e. g. selections.

The first two elements are provided by the UI Toolkit used by RCP as a basis. In case of Eclipse it is SWT for desktop applications, and some other libraries for web applications.

In case of In2Rail-Platform it cannot be assumed to use a common UI-Toolkit because:

- The modules can be developed in different programming languages and;
- UI Toolkits are programming language specific;
- Risk of short life time.

Some kind of exception represents the ActiveX-approach, where the Controls provided by Windows can be used directly by applications, replacing the UI-Toolkits to some extent. But this interface seems to be outdated on the one hand and is supported only with Windows on the other.

Due to the decision to use IL as a communication platform, the UI-Modules can communicate by publishing their states on some Topics and Subscriptions.

A possible architecture of UI integration is shown in Figure 8.6. The cylinders in the figure **represent Topics on the Integration Layer.**

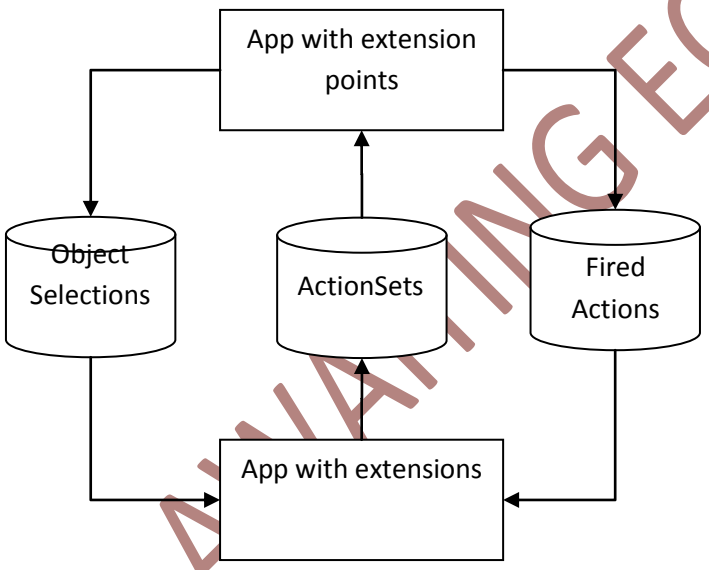

Figure 8.6: Integration of Applications into UI by means of Actions and Selections (metaphor: Eclipse RCP)

Here is the detailed description of the constituents:

- App with extension points: any Application can provide a UI as one or several Windows;
- Selection topic publishes the list of selected objects. A selected object is represented as an address specified in the Canonical Data model, e.g. /tms/signals[s8231];
- ActionSet – is a list of Actions the application with extensions is able to apply now. It is assumed that the App will update this list 10-30 ms after each change of Selection. The App with extension points have to ensure that there is this time interval between the change of selection and construction of context menus, so that it can represent the most current state of actions.

It is assumed that an Action to be represented by two objects:

- Static containing Label, ToolTip text, and Icon as a binary string in PNG or SVG format,
- Dynamic object representing the current state (enabled/disabled) and assumed selection objects. The Application with extension points is able to check, if the current selection fits to the assumed selection in the action and skip actions with selection mismatch;

- If the user activated an Action by selecting a menu or pressing the tool button, the Application with extension points publishes:
  - the Id of the action,
  - the current copy of the selection,
  - Window position on the screen,
  - Mouse position on the screen to enable reactions close to the requested Window,
  - Widget-handle where the app expects the extension to draw its content;

  The App which provided this Action reacts according to its logic.

- In many cases, the Applications will be independently developed, therefore the Application with extension points will not know, how many Views the application issuing the Action represents on the Activation Event. Therefore it is assumed that in most cases Applications will represent their content in distinct Windows. This fact is represented by ApplicationWindow-Class in the class diagram.

The next UI-integration aspect is related to relative positions of the applications on the workstation. To handle it an additional service shall be started on each Node with UI: UI controller (see Figure 8.7).

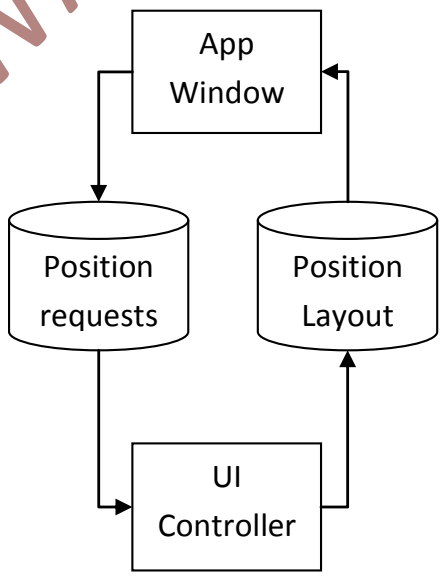

Figure 8.7: Synchronisation of windows locations on the workstation with help of UI controller

The TMS-tenders often require configurability for positioning of Windows on the Desktop. They comprise:

- Prescribed areas for each Application-Window in pixels;
- Prevention of overlapping for specific windows;
- Store/Restore the size and position of all Windows.

All these functions are implemented by the Window Layouter concept. Each Window before starting its construction looks if an appropriate configuration already exists and if not publishes its request with App-Id, required size of the view. The Windows Layouter uses its own configuration to calculate and publish the position and size for the Window. Often some of these functions are implemented by Window Manager provided as part of Operating System. To enable platform independence an explicit modelling of these concepts in IL is needed.

Focus management – is done by UI Controller. Each application requiring focus requests it from UI Controller, who decides if the focus change is allowed.

As always, the In2Rail-Platform shall provide only specifications for Topics and the data structures in the Topics. For details see D8.4.

8.6 Use cases for UI integration

8.6.1 Selection usage

In TMS domain several examples where Selection-pattern can be used:

- A restriction is selected in the restriction editor and because of that the restriction should be centred in the topology view;
- A station is selected in the map view and because of that the station is shown in the CCTV view;
- A train is selected in the time-distance diagram and the corresponding train driver is shown in the train driver communication view.

An interesting aspect could be the activation of Actions by an external logic (see Figure 8.8).

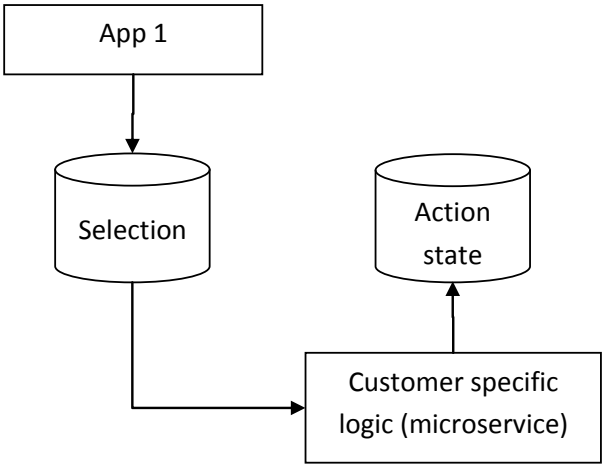

Figure 8.8: Separation of UIs by additional customer specific logic

If App 1 publishes a new selected object (e.g. a station) some additional logic can observe the delay level in this station and decide to start CCTV-application in case of overcrowding. As a consequence both applications are totally unaware of the existence of the other.

Often, the business logic behind this process depends on the kind of selection. Therefore the Selection shall provide the kind of selection as well: clicked, double clicked, menu-selected, key-selected. In this case the "receiving" application can decide how to react on the selection.

Further content of the Selection data structure is the issuing Window-ID and the selected element's position on screen.

8.6.2 Lazy starting

One of the features of Eclipse RCP is its lazy loading of plugins. The developer can put many plugins into one "Application", but only the "Application"-Bundle is started at the beginning. The others are loaded later as soon as their instantiation is required. Typical triggers for instantiations are Actions:

- A plugin provides its extensions, which are used by connected plugins with extension points to be represented in menus;
- As soon as an action is activated (i.e. a method run() is called) the RCP loads the Action implementation and gives the further execution to it.

This kind of application loading enables quick starts and high scalability.

Starting rendering Activities (local plugins) can be implemented by the means of AF: the starting command for the application belongs to the static part of an Action (see Figure 8.9). The Node observing the "Action"-Activation topics start an appropriate bundle as soon as an Action requires that bundle and it is not yet started.

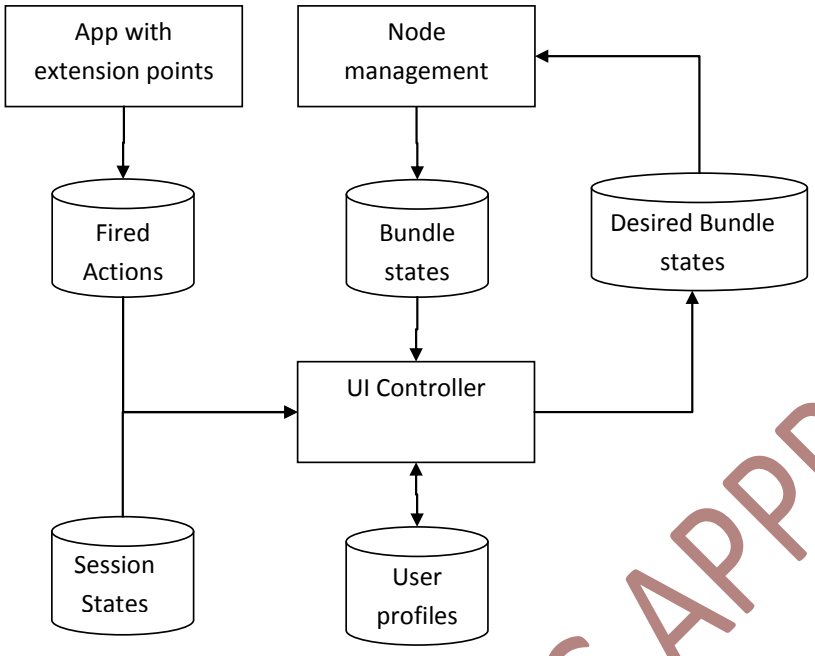

Figure 8.9: UI Controller providing lazy starting functionality

The UI Controller is responsible for observing the Topic "Fired Actions", identifying the required bundle states, and if the specific bundle is not active (not installed), asking the Node management for it.

The logic behind the UI Controller could be in a wide range from pre-starting all bundles at the beginning of their static positions according to the configuration of the User Profile to fully dynamical actions based on the Usability restrictions like available monitor resolution, other started applications, available space on desktop, etc. The implementation details of this logic are not part of the In2Rail platform.

User profiles make it easy for the user to start the whole TMS in the desired configuration. In traffic management departments several roles cooperate with each other, e.g. dispatcher, supervisor, etc. All existing roles are already mentioned and described in D7.2 and D7.3 (see chapter 4.10.7.1 "Roles" in D7.3 and chapter 4.2.1 "Actors" in D7.2). For each of the roles the screen needs to be adapted to their needs, with different views and arrangements (see D7.3 chapter 4.10.5 "Layout of Views"). Even within one role, e.g. for the dispatcher, the views or the arrangement of the views may be different. To reload such preferences of one user, user profiles are saved.

A User profile contains:

- Configured workstation layout;
- The actual workstation layout.

Workstation layout contains the definition of represented Windows (started bundles) incl. Screen-selection and positioning.

8.6.3 Rendering of remote content

Nowadays, popular approach to provide UI is the Web-based clients. In this case, the integrating Application can provide its own view, which renders the Web-based representation of the remote logic.

Web applications are not the only way to provide the content remotely. Some other techniques on the market are:

- RDP (Remote desktop protocol);
- VNC Remote Framebuffer protocol;
- Nomachine NX Protocol;
- ICA protocol (Citrix);
- X-Protocol.

The configuration information to start such a UI shall be part of the Action-configuration: As soon as an Action is started the integrating Application provides a rendering View and starts the communication library supporting the rendering engine.

As the communication between UI-Applications is done per IL, no additional communication infrastructure for “content”-based integration of the local and the remote application is needed. They can use the Action/Selection patterns presented in previous sections.

9 Conclusions

The critical point for opening the TMS markets and driving innovations represents the modularisation of the TMS software. This step allows many companies to provide small functionalities solving specific TMS issues.

On the other hand splitting the TMS software in many modules introduces new issues:

- The reliability of the TMS as a whole must be ensured;
- Each small TMS application shall be deployed, versioned, tested for integration, configured;
- The small TMS applications must be orchestrated (started, stopped, dynamically configured);
- The small TMS applications must create a consistent User Interface on the operator's workstation;
- The requirements on hardware shall not be (much) higher than for conventional TMS software.

The Application Framework shall solve these problems.

The long life cycle of the TMS applications with 20-25 years prevents direct use of the existing integration platforms on the market. On the one hand there are several projects/products competing so their life time is currently unpredictable. On the other hand the interfaces and functionalities of the projects are different, so the substitution of the solution could require considerable effort from the IMs.

This document together with deliverable D8.7 provides a narrow wrapper, separating actual implementation from the tools using it in TMS. This allows simple exchange of the basis technology without involvement of IM and ensures long life time of the integrated TMS software.

10 Appendix A

View Name	Summary
Widget View	This view offers different widgets, connected to the internet, which provide to the user complementary information that may be useful depending on the use case/operational scenario.
Map View	This view provides an animated map containing static and dynamic information. Layer-concept shall be used to group different kinds of information, such as: information related to crew management, consist management, train operation, evacuation facilities, etc.
Topology View	A schematic view of the railway is provided, allowing the user to concentrate on railway operation when needed
CCTV View	This view shall offer CCTV images of the different cameras installed in the area being controlled at the TMS
Telefony-View	This view shall allow the operator to connect with the train drivers and other staff by voice communication utilising GSM-R and other communication means.
Alarm and Events View	List of alarms and events that are being processed by the TMS displayed as a tabular view
Planning View	This view shall provide schedule and regulation information of the railway operation
Settings View	View for allowing the user to customize the workstation
Actions View	This is a dynamic view, automatically activated when the user wants to send a command to a certain element or group of elements
Detailed View	A view for displaying details of a certain element. The details vary depending on the selection done
Assets View	View for displaying assets information in real time that can be used by maintenance personnel
Logistics View	View for displaying logistic information associated with the maintenance of the railway
Reports and stats view	View for creating and displaying reports and stats of the TMS. The view provides support for both “operation of the railway” information and maintenance information
Customer information View	View for graphical representation of the customer information infrastructure.

Table 10.1: Details of Workstation Views [D7.3 chapter 4.10.4.]

11 References

[ACCP 2017] https://accelerite.com/products/cloudplatform/

[AWS 2017] Docker Enterprise Edition on the AWS Cloud, retrieved from https://s3.amazonaws.com/quickstart-reference/docker/latest/doc/docker-datacenter-on-the-aws-cloud.pdf

[Cachin et.al.2006] Introduction to Reliable and secure distributed programming. C. Cachin, R. Guerraoui, R. Rodrigues, Springer, 2006

[Chappell 2004] Enterprise Service Bus: Theory in Practice; D. A. Chappell, O'Reilly, 2004.

[CloudFoundry 2017] https://cloudfoundry.io

[Daniel et al 2007] Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. F. Daniel, M. Matera, Jin Yu, B. Benatallah, R. Saint-Paul, F. Casati 2007. Retrieved from http://www.floriandaniel.it/papers/DanielIEEEIC07.pdf

[Docker 2017] https://docs.docker.com/engine/swarm

[EC2 2017] http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

[EMB 2017] http://www.embotics.com/solutions-cloud-governance

[Hazelcast 2017] Hazelcast IMDG, retrieved from https://hazelcast.org/

[Kubernetes 2017] https://kubernetes.io/

[MS VMM 2017] https://docs.microsoft.com/en-us/system-center/vmm/overview

[IN2RAIL D7.1] State-of-the-Art and High Level Requirements. IN2RAIL

[IN2RAIL D7.2] Consolidated functional and non-functional requirements. IN2RAIL

[IN2RAIL D7.3] Specifications of the Standard Operator Workstation. IN2RAIL

[OMG DDS 2017] Data Distribution Service™ (DDS™), retrieved from http://www.omg.org/spec/DDS

[OpenShift 2017] https://openshift.io/

[OSGi 2017] The OSGi Alliance: OSGi Core. Retrieved from https://osgi.org/download/osgi.core-7.0.0-early-draft-2017-03.pdf

[Paulheim 2009] Ontologies for User Interface Integration, H. Paulheim, Retrieved from http://www.heikopaulheim.com/docs/iswc_2009.pdf

[RH CF 2017] https://www.redhat.com/en/technologies/management/cloudforms

[SWVM 2017] http://www.solarwinds.com/virtualization-manager

[VMT 2017] https://turbonomic.com/solutions/projects/private-cloud-management/

[VMW 2017] https://www.vmware.com/products/vcloud-suite.html

[Winn 2017] Cloud Foundry: The definitive guide, D.C.E. Winn, O'Reilly, 2017