RCA

Reference CCS Architecture

An initiative of the ERTMS users group and the EULYNX consortium

Explanation of ARCH Process, Methods, Rules

Document Number: RCA.Doc.60
Version Number: 0.2
Published: 2022-04-26

Publisher:
EUG and EULYNX partners

Technical authors:
None

Classification: None
PUBLIC

Table of contents

Content

Version history

0.1	07.12.2021	First version
0.2	26.04.2022	Still initial version with some updates and changed of format

1 Introduction

1.1 Purpose of this documentation

This document gives a small overview of the context of M&T Cluster and the relation to other clusters in RCA. It is also gives an insight into the ARCH process, methods and rules. To explain how to work with it a theoretical example is given. The full content of he ARCH process, method and rules can be found in RCA.Doc.33.

2 Method and Toolchain (M&T) Setting

The RCA cluster M&T provides all services around tooling, processes, method and rules for currently all other clusters in RCA working on the RCA model. Broader European usage of the M&T services is possible and under discussion. The current work mode setup between M&T and the clusters is presented in Figure 1. The tooling platform is the basis for all modelling work done in the different clusters. The core modellers are included in work groups of different clusters and are there formalising the model together with domain experts. While domain experts have only basic knowledge of the methods and modelling rules, the core modellers are the experts providing method and modelling knowledge as a service. With such a centralised service it is possible to achieve a common, overarching model that can be used to produce a system architecture as part of the architecture cluster work and to elaborate this architecture in discussions with industry. Besides the pure modelling rules, common processes like document management, configuration management, review, etc. need to be defined and will be delivered by the M&T cluster.

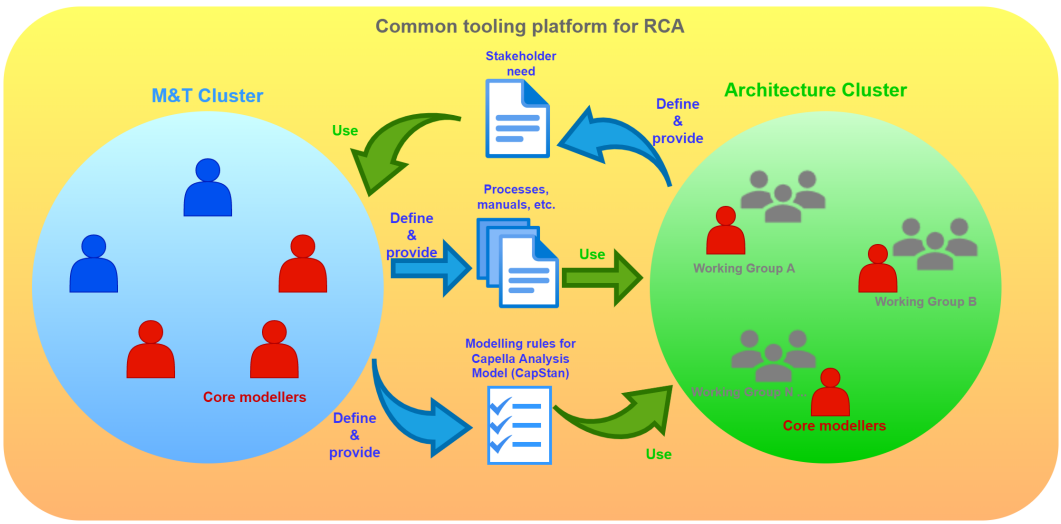


Figure 1:

Figure 1 Setting of M&T as central service for all RCA clusters using the same tooling platform.

3 The ARCH Process and ARCH Methods

One goal of RCA is defining a reference architecture for a railway command-control-signalling (CCS) system, to achieve this goal a process is needed to create an effective reference architecture.

The ARCADIA method already provides certain rules at a high level, but for detailed work, distributed over different groups, more guidance is needed. Especially in the case of RCA, where new stakeholder needs need to be assessed that lead to innovative products with influence on the whole railway system. To break down the complexity of such a new system and to obtain well defined requirements, strict method, rules and a reliable process are key to success. In the case of RCA therefore the ARCH process based on the ARCADIA method was established. It is crucial, to achieve a common understanding of how certain features are expressed in the model. The ARCH process provides a clear work breakdown structure and rules to produce a consistent set of artefacts that support consistency of different approaches and help to get a proper reference architecture.

It aims therefore, that the different groups working on the same model get to comparable results. The method documentation and the modelling rules also serve external experts, such as industry engineers, to exactly interpret the model. In Figure 2 the overall ontology of process and methodology as applied at RCA is shown. The following figures present different levels of the ARCH process and their link to the specific methods and modelling rules as applied in RCA. The whole documentation is currently done in confluence, an easy-to-navigate documentation tool. In addition, a html export (RCA.DOC.33) of the current status was generated to grant access for everyone.

RCA agreed on using the ARCH process already used and provided by DB. Therefore some parts of the process still refer to DB internal information, but nevertheless the overall process is applicable to RCA. Cleaning up the ARCH process to a RCA process is ongoing work.

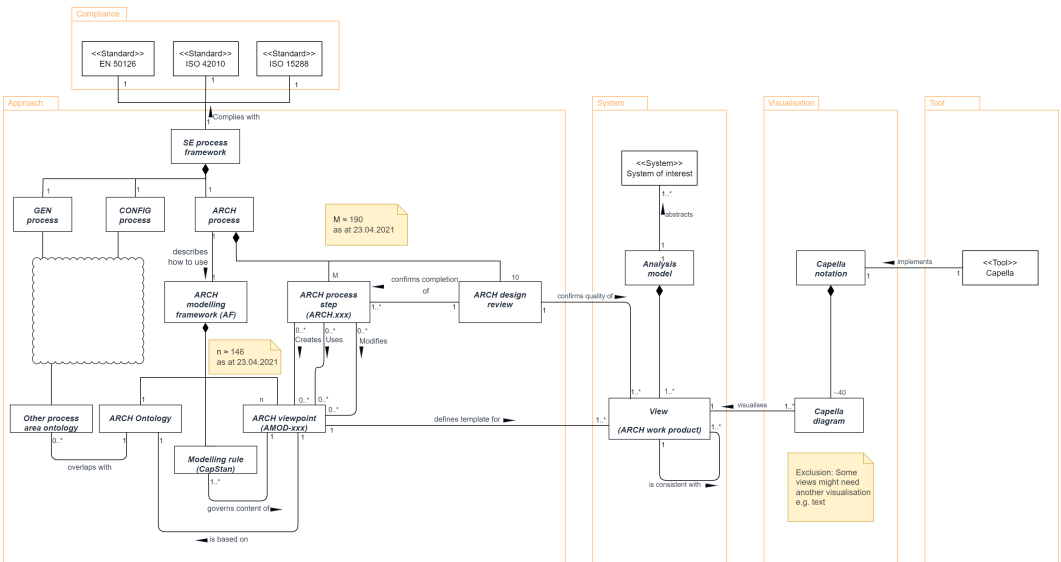


Figure 1: Figure 2 Ontology for process and methodology

- None -

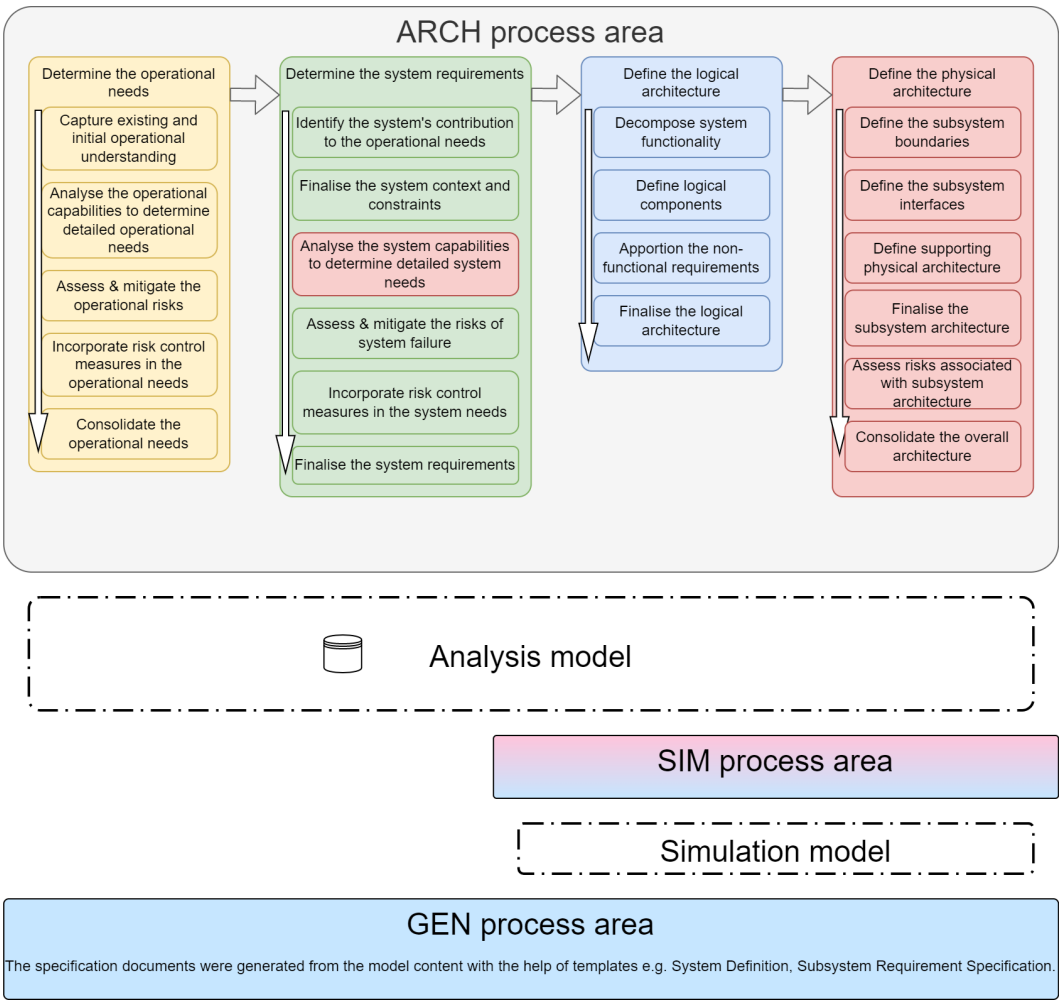


Figure 1: Example: ARCH.913 Analyse the system capabilities to determine detailed system needs

- ARCH.179 Complete the definition of the system capability of interest
- ARCH.088 Define system functions and functional exchanges
- ARCH.052 Create initial system exchange scenarios
- ARCH.053 Create initial system functional chains
- ARCH.054 Model data flowing between system functions
- ARCH.055 Model system and actor states
- ARCH.158 Model external interface layers
- ARCH.057 Model non-payload data on external interfaces
- ARCH.058 Define measures of performance

Figure 3 The ARCH process based on the ARCADIA method. For each box in the coloured columns above, there are multiple process steps defined in confluence.

The figure 4 and 5 shows an example of a detailed process step description and a viewpoint description in confluence:

- None -

Goal	Identify and capture functions and exchanges that are needed for a particular capability
Requirements met by this process step	ISO 15288 6.4.2.3 b 1), c 2) EN 50126-1 7.2.2 a), 7.3.2.1 a)
Inputs	AMOD-138 Single system capability context ARCH.M.010 Method for definition of functionality
Outputs	AMOD-056 System functions and exchanges (single system capability)
Methodology	• This step should run in parallel with ARCH.052 Create initial system exchange scenarios and ARCH.053 Create initial system functional chains • Define the functions that are needed for the capability of interest. • The method and the guidelines for the definition of system functions should be followed defined within ARCH.M.010 Method for definition of functionality. • If functions and exchanges already created previously, they can be reused **if it is appropriate**, but if no existing function and exchange meets the need at that point, then new function needs to be created. • Document the justification for a particular design decision with a rationale.
Tools and non-human resources	Team for Capella
Cardinality	Once per system capability.
Completion criteria	The output diagram conforms to its modelling rules. The set of functions and functional exchanges identified is safe enough to try.
Design review	ARCH.R.3 System capability review
Step done by (Responsible)	Lead system architect
Provides input to/assists (Contributes)	• Expert for GoA4 railway operation • System architect • Engineer • Cross-cutting engineer
Uses outputs (Informed)	RAMS manager Verification & validation manager

Figure 4 Example of a process description on the lowest level (ARCH.088 in Figure 3).

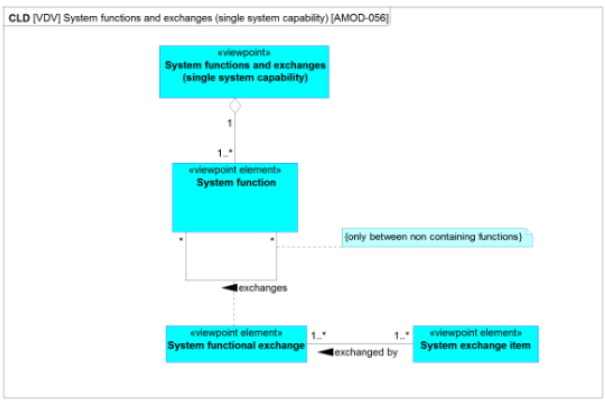
Created by	ARCH.088 Define system functions and functional exchanges
Concerns	Define the functions needed to achieve a particular system capability Define the functional exchanges needed to achieve a particular system capability
Modified by	ARCH.054 Model data flowing between system functions ARCH.069 Add system level risk measures ARCH.084 Execute automatic transition of system elements to logical level ARCH.916 Incorporate risk control measures in the system needs ARCH.152 Consolidate system functionality
Used by	ARCH.066 Identify system level deviations ARCH.069 Add system level risk measures ARCH.066 Identify system level deviations ARCH.055 Model system and actor states
Viewpoint	**In Capella this corresponds to a SDFB diagram.** **Mandatory Viewpoint:** 
Diagram modelling rules	2. Modelling Rules for System Analysis#ModRules_VP_SDFB
Element modelling rules	2. Modelling Rules for System Analysis#ModRules_VE_SysFuncExchange 2. Modelling Rules for System Analysis#ModRules_VS_SysFunction
Viewpoint modelling rules	2. Modelling Rules for System Analysis#ModRules_VS_AMOD-056

Figure 5 Example of a viewpoint definition created for process step ARCH.088 presented in Figure 4. Here also the corresponding modelling rules are linked.

4 Explanation of ARCH layers

The ARCADIA method does not attempt to provide a full blown process for specific domains such as RCA. It provides a method for a functional analysis to derive a system architecture. All domain specific needs have to be defined by the domain experts. Therefore, M&T provides with the ARCH process such a framework.

In the following, an example per layer is presented. Especially in examples including interfaces to the Traffic Management System (TMS) and the train onboard CCS system (e.g. OCORA) the overall complexity becomes visible.

4.1 Operational Needs (OA) layer

Definition of the operational processes that shall be (eventually only partially) supported by the system

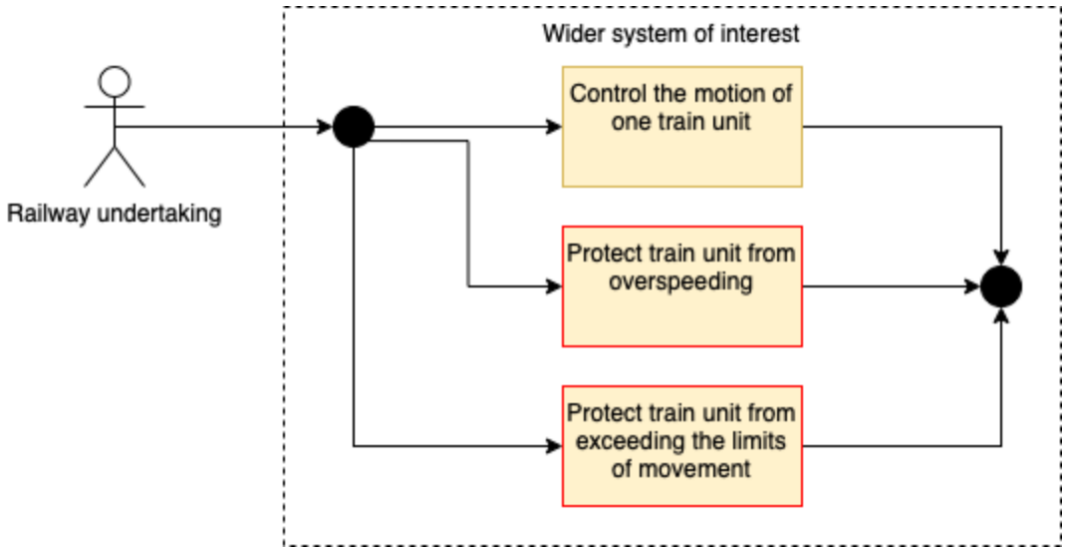


Figure 1: In this layer, we focus purely on the operational processes, In this layer, we focus purely on the operational processes, as agnostic of technology as it can get!

- ⌘ We avoid mentioning specific technologies like ETCS or ATO
- ⌘ Layer must pass the "steam train test"

On this layer, we can start to perform risk analyses. We do this by identifying pure operational hazards and thinking of countermeasures. This is done for 3 categories of risks:

- ⌘ Safety risks ("S", typical railway hazards) - see red boxes above
- ⌘ Security risks (second "S", already possible partially on process level, e.g. authorisation can be found here)
- ⌘ Risks to the business ("RAM", e.g. missing performance or reliability is a business risk)

4.2 System Needs (SA) layer

This layer represents the system needs or "Definition of work statement" - this layer does not show any solutions, just what the task is that is to be done and how that task is restricted.

The area of concern for this layer is to define:

- ⌘ what part of the operational capabilities the system shall do and which parts not
- ⌘ which constraints are applicable to a specific solution

Example:

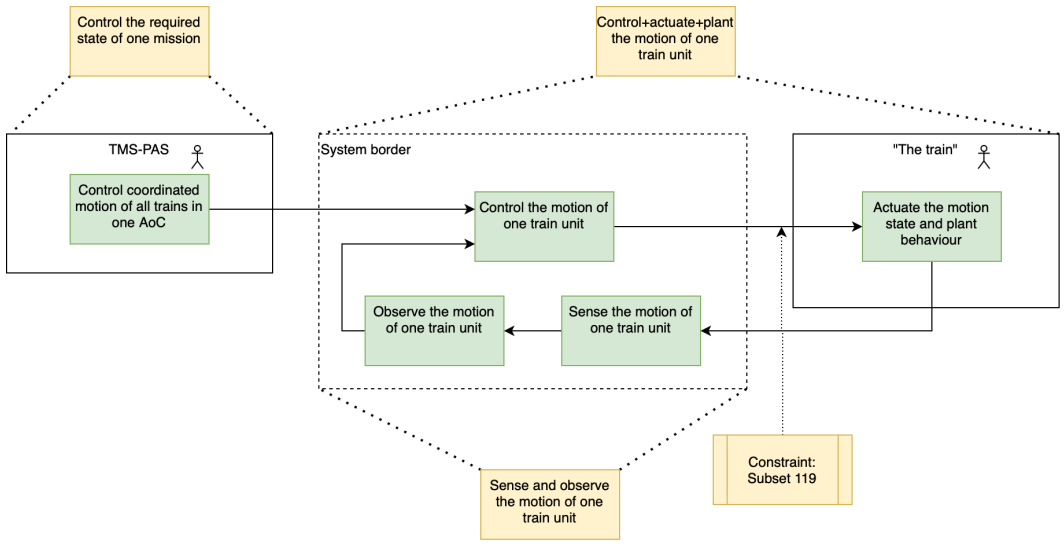


Figure 1: In this example, that roughly translates to:In this example, that roughly translates to:

- We need a system that interacts with TMS and the train
- The system needs to control the motion of that train
- The system must need to be able to sense and observe the motion of that train
- The observation must be fed back to form a control loop
- The interface to the train must be compliant to subset 119.
- The interface to TMS is not constraint (we can design whatever we want: morse code, fax, homing pigeon).
- The SA layer doe not yet mention any solution concepts and should be agnostic as possible. The above example could be implemented by today's electronic interlockings, track-side signals and and train control systems like legacy PZB or ETCS.

System functions are following a certain pattern according to control loop theory:

4.3 Logical Architecture (LA) layer

The area of concern is "Defining sub-problems, introduction of logical solution concepts and logical building blocks" - this layer shows which basic ideas and concepts are used to define small "bricks" of solutions that can be used to build an architecture upon.

LA layer **does not yet define a system architecture, but a library of building blocks to construct an architecture from.** Also LA layer does not define, if a solution is realised e.g. on the track or onboard side.

There are also other splitting criteria to split the system functions, but one is a layer architecture of layers of a specific purpose, e.g. Safety Control is a layer that decides authoritatively about the state changes of an controlled object.

Example:

- None -

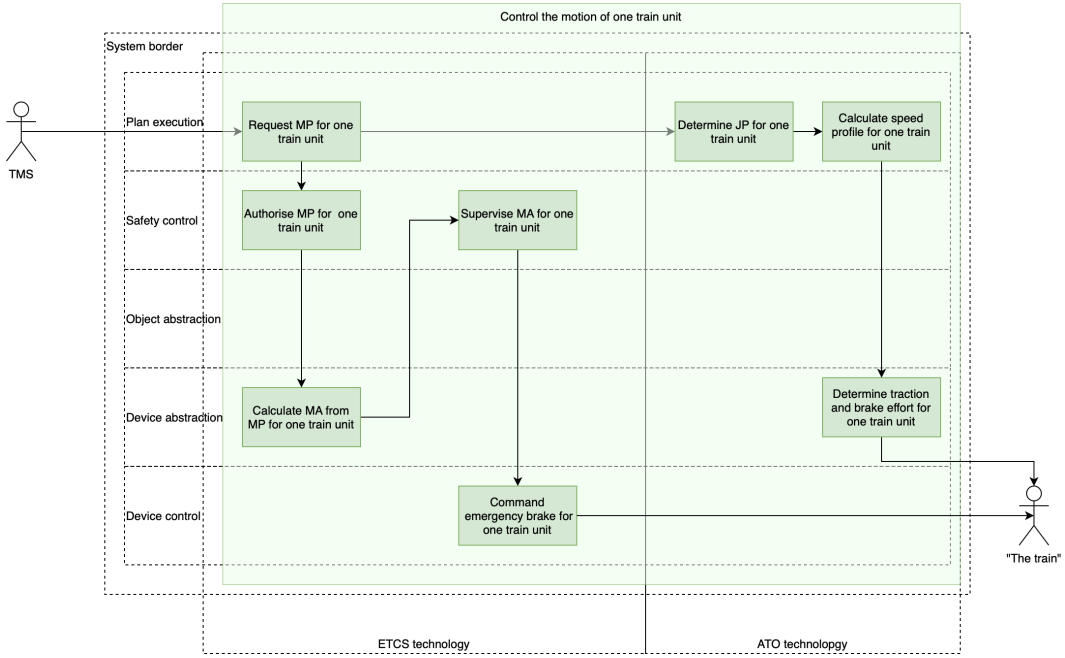


Figure 1: in the represented example case, that roughly translates to (only shown on the control function):in the represented example case, that roughly translates to (only shown on the control function):

- The system function splits up into 5 logical functions according to the purposes of the layers. One layer can be used multiple times!
  - First we introduce the Movement Permission (MP) as concept; the MP is requested by the plan execution function (as it executes the operational plan coming from TMS)
  - Then the MP is authoritatively allowed or denied by a safety control function - meaning: there is nobody else allowing or denying a MP
  - Then the MP is translated to an MA according to SS026 in the abstraction layer
  - Then the driving is supervised against the MA including the Most Restrictive Speed Profile (MRSP) formed based on the MP, this again is a safety layer function
  - In case of a violation of the MA a safety reaction is triggered and commanded to the vehicle by the device control layer
- We can use all the nice concepts we think of: MPs, MAs, etc. all those solution concepts are fully valid here and determine the splitting of a system function

Logical functions are also the elements that will carry the specification of their behaviour. This can be done by natural language (not recommended) or by semi-formal or formal means of specification. Also non-functional requirements are derived for the functions (e.g. accuracy, latency, failure rates...).

Functions are then allocated to logical components. These logical components are:

- *not subsystem* but small providers of services, like in a micro-service architecture
- not instantiated, so there is only 1 per system
- not allocated to the deployment location (track-side, OB or cloud...)

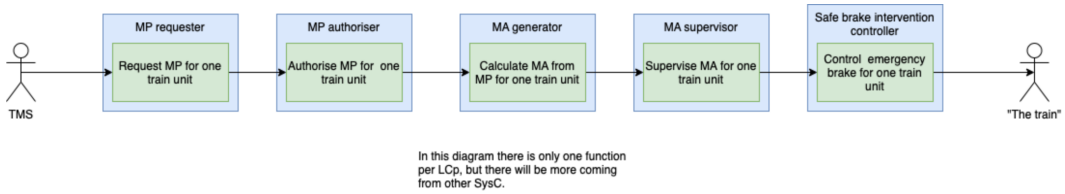


Figure 1:

4.4 Subsystem Architecture (PA) layer

"Do system architecture, define subsystems and interfaces" - in this layer, the "real" subsystem architecture is created. **Only from here** an allocation to track-side/on board is possible, also in terms an allocation of functionality to domains should happen here.

Usually multiple physical architectures are possible based on the **same** logical architecture.

Example 1:

The first example shows, how the logical components are allocated to subsystems in a classical way creating roughly the exiting ETCS SS026.

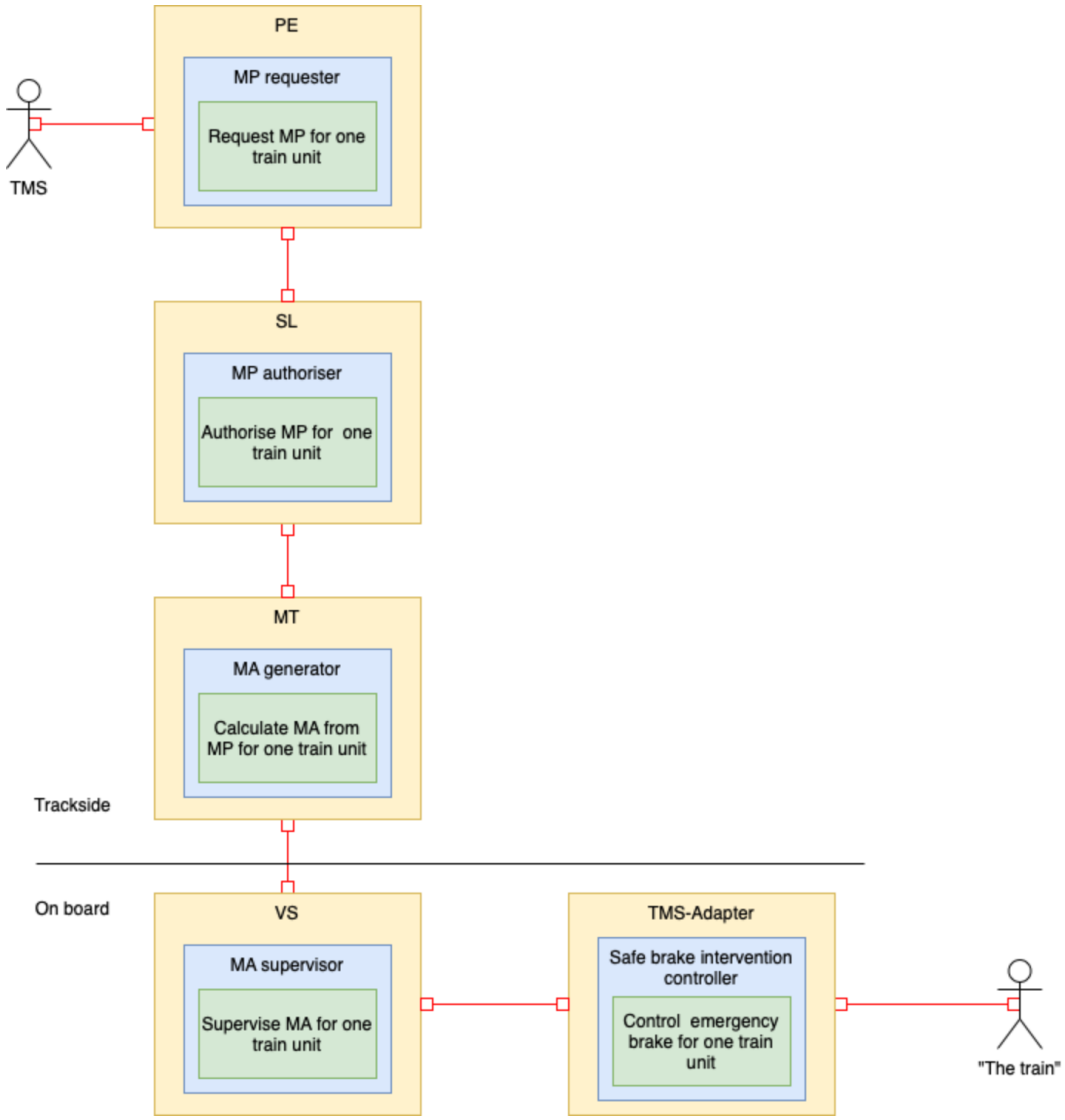


Figure 1: Example 2:

The second example shows an architecture that was (if I remember correctly) at one point in time proposed by some IM for a virtual European Vital Computer (EVC) implementation on the track-side (!). In this architecture, only brake commands are transmitted to the train (I know that is not what we want). But the interesting fact is that both architectures are based on the same logical components.

- None -

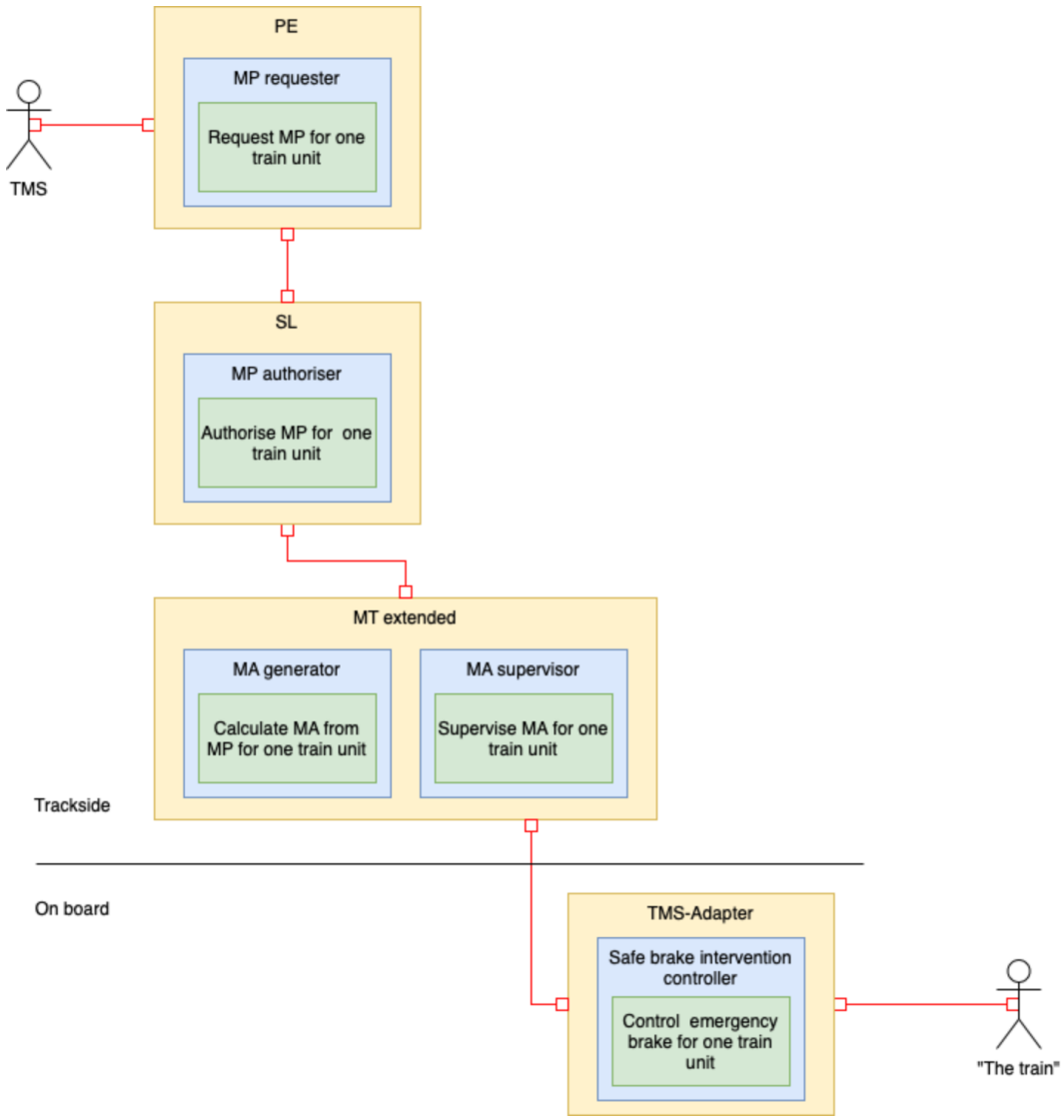


Figure 1:

After the logical components have been allocated to the subsystems, all interfaces and their extended behaviour have to be specified. This means that also behaviour has to be added that takes the existence of interfaces into account, e.g. the fact that they can fail or need to be initialised.

This will create a lot of work and requires deep knowledge of the subsystems and in any case needs to be done in the domains. In the end, on this layer all requirements will come together and are the basis for generating specification documents.

- None -

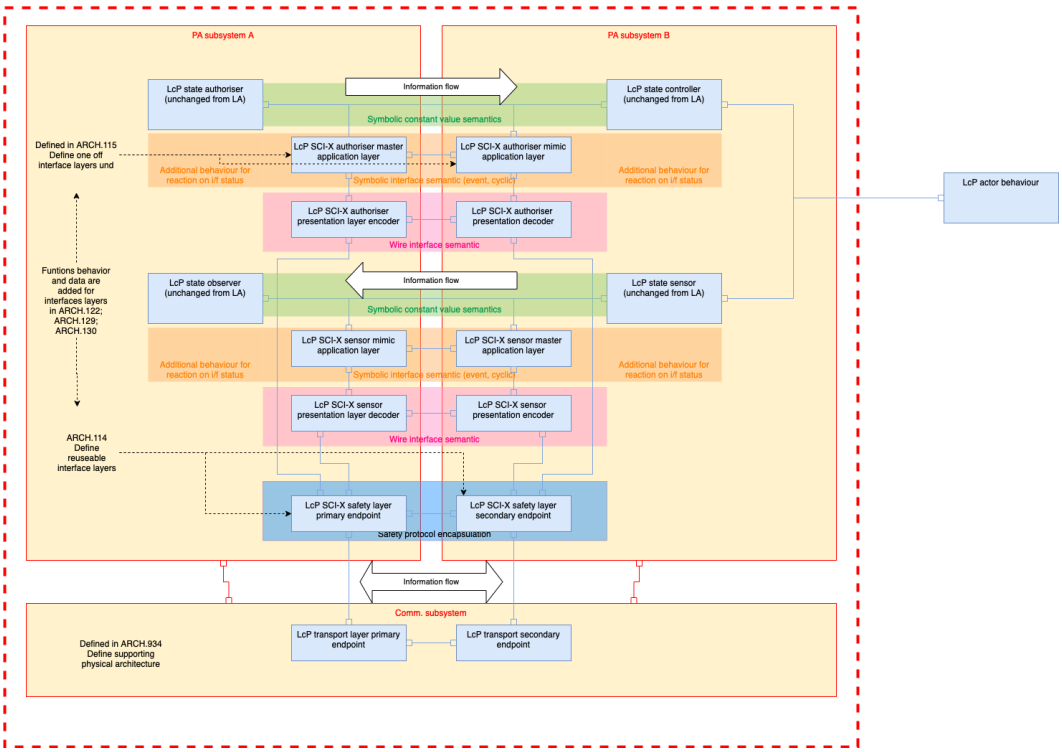


Figure 1: This layer defines:This layer defines:

- which individual subsystems are in the architecture
- by which standardised interfaces are these subsystems connected (full stack, at least so many layer as are needed to define the interface on FFFIS level)