Deliverable D 18.1
Common Framework Orientations

Project acronym:	FP3 - IAM4RAIL
Starting date:	01/12/2022
Duration (in months):	48
Call (part) identifier:	HORIZON-ER-JU-2022-01
Grant agreement no:	101101966
Due date of deliverable:	Month 12
Actual submission date:	08-02-2024
Responsible/Author:	JOLY / SNCF
Dissemination level:	PU
Status:	Issued

Reviewed: (yes)

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101101966.

Document history		
Revision	*Date*	*Description*
0.1	9/12/2023	First issue
0.3	15/01/2024	Final version after TMT and SC approval
0.4	07/02/2024	Version after quality check
1.0	08/02/2024	Final version submitted to ERJU

COVERING DOCUMENT

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Louis-Romain JOLY	SNCF	Initial Draft and refinement
Clara CUSSAGUET	SNCF	Review
Vincenzo MANNO	FSI	Review

PART A – MIDLLEWARE SELECTION

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Björn Kahl	SNCF	Initial Draft
Louis-Romain JOLY	SNCF	Workpackage internal review, answers to reviewer comments
Clara CUSSAGUET	SNCF	Workpackage internal review
Luca TISENI	FSI	Workpackage internal review
Vincenzo MANNO	FSI	Workpackage internal review

PART B – ADVANCED MODULARITY

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Louis-Romain JOLY	SNCF	Initial Draft
Clara CUSSAGUET	SNCF	Review
Vincenzo MANNO	FSI	Review
Luca TISENI	FSI	Section editing and review
Emilio SANCHEZ	CEIT	Section editing and review
Björn KAHL	SNCF	Section editing and review

PART C – SAFETY ASSESSMENT

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Louis-Romain JOLY	SNCF	Initial Draft
Rachel HEGEMANN	DB	Updated Sections 6 and Executive Summary
Maria VRSALOVIC	DB	Supply of the original safety plan template and review
Holger SCHLINGLOFF	SNCF	Formulations in 6.3
Clara CUSSAGUET	SNCF	Workpackage internal review
Vincenzo MANNO	FSI	Workpackage internal review
Luca TISENI	FSI	Workpackage internal review
Marcos CONCEICAO	UIC	Review

Juan BENAYAS ARROYO	THALES	Internal review

PART D – VISION OF ROBOTICS" IMPACT

Report contributors		
Name	Beneficiary Short Name	Details of contribution
Louis-Romain JOLY	SNCF	Initial Draft
Vincenzo Manno	FSI	Workpackage internal review
Luca TISENI	FSI	Workpackage internal review
Michele Pacini	FSI	Workpackage internal review

Table of Contents

List of Figures

List of Tables

1 Executive Summary

Objective:

FP3 - IAM4RAIL WP18 was structured around 2 timeframes. In the short-term view, maintenance robots are being developed. Those robots meet business needs expressed by partners. The aims are different: some want to accompany the growth of their transport offer without having to increase the surface area of their maintenance facilities, others want to acquire new resources to accelerate an equipment deployment program, and still others want to reduce their costs or improve the quality of the information they collect.
To ensure that developments continue beyond these 4 robots, the work package aims to structure a railway robotics ecosystem in the medium-term view. This ecosystem supports the technical policy of robot modularity, which is essential for the expansion of robotics in railway maintenance. In this first year of work, we set out to determine the main guidelines that will govern the development of common tools and methodologies for our ecosystem. The purpose of this document is to set out these orientations.

Rather than presenting all our orientations in a single document, we have chosen to draw up 4 documents that can be read independently:

- Part A : justification of the choice of a common middleware;
- Part B : clarification of the concept of advanced modularity and proposals for the tools needed to implement it;
- Part C : principles adopted for a safety demonstration methodology when robots are used in a railway maintenance operation;
- Part D : a vision of the benefits of robotics for railway asset management.

Conclusions:
Concerning the common middleware, the project selects the ROS2 software development. Although technologically more advanced options exist, ROS2 provides the best balance between technological capabilities, wide-spread support, especially with hardware vendors, and a fast, vibrant global developer community, which makes its long-term survival highly likely. Besides, ROS2 or its predecessor ROS is already in use at multiple project partners for prototyping and development, thus minimizing on boarding efforts.
ROS2's major drawback of lack of structure development tools is mitigated by longstanding and ongoing work from project partners to bring modern, model-driven development to the ROS world.

To guarantee an advanced modularity the project propose to develop a set of tools which will benefit from work previously carried out by Fraunhofer IPA. This minimizes our development effort while maximizing our chances of success. Our development efforts will focus on 7 fundamental tools (eg. data structures for component and system description, related database, plugin for properties visualization, properties extractor from C++ or Python code…). For 2 tools

(automatic documentation generation and code quality check), we offer to test existing solutions and make recommendations for use and/or further development. The market offers solutions, and we don't know enough about their limits to justify further development. For one last tool (robot's software distribution from its model), we will simply draw up recommendations for future developments. Our resources do not allow us to address this theme in parallel with the others, and it does not appear to be a priority.
The corpus of tools that will be available at the end of the project will provide a solid, concentrated core of Model Design Engineering tools with a strong impact on product quality and development productivity. Beyond the scope of the project, efforts will have to be continued to increase the perimeter of certain tools (moving the system's data structure from the "logical" scale to the "physical" scale, then to the mechanical scale) or by undertaking work on tools that are not yet covered (software distribution tool).
4 tools will export constraints to developers. We have endeavoured to limit this number. The precise list of constraints will be known in January 2024, following the completion of a study conducted at the University of Stuttgart with the help of students.

The Safety Plan is the document that summarizes all the elements to be produced for the safety demonstration. It is divided into 4 chapters (Purpose of the Safety Plan, System Definition, Safety Proof Concept, Safety Assessment Report), which represent the basic pillars of the safety case for the process change of maintenance measures in the rail system.
The first section "Purpose of the Safey Plan" indicates that the safety plan is a covering document which organizes and references the most important documents in the safety demonstration. The second section "System Definition" allows the specific project to be broken down into its most important components and ensures that all aspects of the process change have been considered for a proper safety verification. The third section "Safety Proof Concept" structures the path that is to be followed to provide proof of safe operation. The path has been organized around 5 categories each one integrating the level of autonomy. All the elements mentioned above lead to the forth and final section, a central safety assessment report.
This deliverable is the fruit of initial work that needs to be enriched. We will be working on two types of improvements over the coming months. We will be developing or continuing to develop templates and guidelines to help write the safety plan sections themselves. The second axis will be the development of a Unified Safety Process for Railway Maintenance Robots. The work done so far describes what needs to be delivered, but not how best to organize the work to deliver it.

To build a vision of the robotics impact on railway maintenance, the adopted approach is a mix between a high perspective analytical approach and a high perspective fictional approach. The analytical approach is based on a breakdown of maintenance into more basic processes. For each of the elementary processes, a short list of relevant indicators (in the context of the introduction of robotics) is proposed. Reference levels are determined. The last step consists of evaluating the evolution of these indicators on a scale of approximately 5 years.
The fictional approach is inspired by Red Team Defense offered by Paris Sciences & Lettres to the French armies. They propose, over a longer time horizon, futures for which the probability of occurrence is not the key point. It is the reactions to be implemented in the face of these new situations that have important value. Creating a collective imagination in addition to more

traditional commercial relationships can also be a strong glue in a new-born ecosystem.
The total duration of the selected approach is 18 months, based on 4 stages for the analytical approach and on an iterative work of 6 to 9 months for the fictional part.

2 Abbreviations and acronyms

Abbreviation	Definition
EU MAWP	**E**urope's Rail Joint **U**ndertaking **M**ulti-**A**nnual **W**ork **P**rogramme
ROS	**R**obot **O**perating **S**ystem
WP	**W**ork **P**ackage

3 Background

The present document constitutes part of the Deliverable D18.1 "Common Framework Orientations" in the framework of the Flagship Project 3 – IAM4RAIL as described in the EU-RAIL MAWP.

4 Objective/Aim

FP3 - IAM4RAIL WP18 was structured around 2 timeframes. In the short-term view, maintenance robots are being developed. These robots cover in a matrix fashion the 2 fields of infrastructure and rolling stock, and the 2 maintenance levels of inspection and intervention. Those robots meet business needs expressed by partners. The aims are different: some want to accompany the growth of their transport offer without having to increase the surface area of their maintenance facilities, others want to acquire new resources to accelerate an equipment deployment program, and still, others want to reduce their costs or improve the quality of the information they collect.
To ensure that developments continue beyond these 4 robots, which are far from covering all relevant uses, the work package aims to structure a railway robotics ecosystem in the medium-term view. This ecosystem supports the technical policy of robot modularity, which is essential for the expansion of robotics in railway maintenance.
These 2 timeframes of work feed off each other. The components developed on the 4 robots will be the first building blocks of our ecosystem. Similarly, the first ecosystem tools will facilitate the development of the 4 robots.
In this first year of work, we set out to determine the main guidelines that will govern the development of common tools and methodologies for our ecosystem. The purpose of this document is to set out these orientations. As a preamble, we will come back to the general philosophy that guided the structuring of this work package.

Rather than presenting all our orientations in a single document, we have chosen to draw up 4 documents that can be read independently.

The first document concerns the choice of a **common middleware**. It is an essential tool in robotics. Creating a data bus that allows a very fluid flow of information, makes it possible to have a set of unitary programs carry out complex tasks rather than a single large program. It is much easier to make a unitary software context-independent. This independence of context means that the code can be reused for a wide range of use cases. This document justifies the choice of middleware made by the partners.

The second document concerns what we have called the **overlay for advanced modularity**. As mentioned above, middleware is the cornerstone of modularity. They are so flexible that, in an industrial context, this can run counter to the ratio of product quality to development time. We therefore feel it is necessary to provide tools that place modularity at the expected level and speed up the design phase. This document describes the sub-concepts of advanced modularity that we wish to promote and proposes a list of tools capable of supporting them. This document also indicates the constraints that will result from the use of these tools for developers.

The third document concerns the **safety assessment**. Here, too, the aim is to develop the principles that have been adopted and to propose a suitable methodology at the end of the project, even if some of the tools needed to implement it have not been developed yet. Once again, the impact on development will be explained. Methodological developments will be carried out in parallel with product developments. The aim is therefore to synchronize the 2 approaches

as closely as possible, to minimize the need to adapt products that may not conform to the finalized methodology.

The fourth document looks at the contribution **(vision of the impact) of robotics to railway asset management**. Rather than drawing up a detailed list of railway maintenance applications where the use of robotics would make sense, we provide a macro-perspective of the values generated in various scenarios.

5 Work-package roadmap

Our society is faced with considerable environmental challenges: climate change, increasing rarefaction of raw materials, etc. Rail transport has several strengths to respond to this unprecedented situation.
Maintenance is an important element of the rail system, and one on which safety is built. But maintenance is a cost and unavailability factor. It must evolve to better support the railways' values proposition.
Maintenance can be seen as a continuous cycle based on 3 pillars: monitoring, decision-making and action. Monitoring consists of enquiring about the state of the system. Once this information has been acquired, the next step is to decide what needs to be done to ensure that the equipment being maintained meets the desired objectives. A part needs to be replaced, another needs to be adjusted, and monitoring needs to be reinforced... Once the decision has been taken, it must be implemented: this is the action.

Three of today's most popular technologies cover the first 2 pillars. IOT enables better (more often, with more detailed information…) monitoring. AI and massive data processing enable us to make better decisions or better guide decision-making. Better monitoring and decision making is good. It can help keep components as close as possible to the limits, for example by applying a predictive maintenance strategy. However, the weight of operations upstream and downstream of the maintenance act, and the availability of installations, can be real obstacles to translate these decisions into action.
Few technologies can cover the third pillar, that of action. Robotics is one of them. Although it can cover the other two pillars (monitoring and decision-making), it is really on the action side that it will have the greatest added value. For example, by enabling maintenance to be carried out without the need for installation (maintenance pit or walkway), or by freeing up the workforces of operators with key skills, robotics can help maintenance progress. At the same time, robotics can improve working conditions for maintenance operators, making the jobs more attractive.

This is the macroscopic challenge of our work. But what kind of robotics do railways need?
Use cases are numerous and various: visual inspections, repair by metal cladding, cleaning, stripping and anti-corrosion protection, parts installation or replacement...
The robots that can respond are different. Nevertheless, there may be a large commonality in the components that make them up. If they are developed in silos: a robot, a development from a blank page by a different player, a significant part of the developments will be devoted to subjects already covered (in the previous development by other entities). A single player developing all uses (an extreme example designed to highlight the consequences) will be able to take advantage of the massification of components. But it will find itself in a de facto position of strength, compared to other technology providers and end-users.
This massification of railway robot components therefore needs to be organized by the sector, ideally on a European scale to exceed a critical size. This massification of rail robot components must be carried out across rolling stock and infrastructure. It concerns software, but also hardware (sensors, mobile bases, computation units, effectors, etc.).
Modular robots are what the rail sector needs.

Modular railway robots will only come into being through a platform policy. IAM4RAIL's WP18 aims to launch this policy. Two time horizons are being worked on: the short term and the medium term.
In the short term, we need to develop and test the first modular robots that respond to business needs. Parallel to the Technology Readiness Level, the Demand Readiness Level needs to be advanced, to show what can be done, and to inspire future users.
A final component is essential. This is the Manufacturing Readiness Level. Tomorrow, we need to have an operational technical and economic network ready to support the ramp-up of rail maintenance robots. That's what we want to do in the medium term, to structure a European railway robotics ecosystem.

Four modular robots and a laboratory prototype will be developed. Two robots will be used for train maintenance, and two for infrastructure maintenance. Moreover, two robots are inspection robots and two robots are intervention robots. The laboratory prototype will be used to develop a fifth robot, an infrastructure repair robot, at a later stage.
The robots will be used for various use cases: to disinfect trains and small stations, to measure track gauge, to inspect catenaries and tunnels, to inspect trains underbody and install objects (ERTMS balise, axle counter…) on the track.

In the medium term, WP18 will build a first set of common tools to support the ecosystem and its technical policy (platform of modules). The first of these tools will be a common middleware. Its selection is the subject of a document within this deliverable. This middleware is a key element, as it enables software modularity. It should be noted that we use the term middleware rather loosely. The core functionality of middleware is to orchestrate data exchanges between other programs. In robotics, other functionalities are systematically combined with basic software modules. We could therefore speak of a development framework. As the term "framework" is not very precise, as it is used in very different contexts, we have chosen to use the term "middleware", even if it does not cover the entire technical scope underlying the choice.
We wrote that middleware was the key to modularity. It enables software to be interconnected by organizing the data exchange. For use in industrial environments, today's middlewares are even too permissive. In a caricature, we could say that they connect any program to any other. If basic checks are carried out, for example on the type of data being exchanged (image, integer number, float number, string, point cloud, GNSS measurement data, etc.), more detailed checks are required to guarantee the quality of the programs chain. It is therefore necessary to guarantee the correct interconnections between software modules in order to reduce development times and increase assembly quality. Our aim is to offer tools based on model design engineering, but without imposing the entire approach, so as not to impose too many constraints on developers.
All these components we're talking about must also be distributed. This is in the interests of those who develop and offer these components, and those who wish to use them. A marketplace must therefore be the right tool. It's this marketplace that will form the truly visible part of the ecosystem we want to build. By building a marketplace prototype as part of this project, we will be able to start discussing the business model that will make exchanges viable in the long term.
Finally, a very important element for our ecosystem is a common safety demonstration method.

Safety is one of the pillars of rail transport. In an industrialization project, a significant proportion of resources can be devoted to safety demonstrations. The modularity we are promoting can be used to shorten this phase and reduce its cost. This must be organized rigorously and transparently, to build trust between all stakeholders.

Beyond ERJU's first call, other actions will be necessary to launch and then sustain the European railway robotics ecosystem (integration of a more significant number of technology providers, and setting up a governance structure...). Nevertheless, with IAM4RAIL WP18 we are helping to lay solid foundations for the rest of the process.

The roadmap is presented graphically in Appendix A.

6 Conclusion

6.1 Middleware selection

The project selects the ROS2 software development framework as its common middleware. Although technologically more advanced options exist, like the Japanese ORiON framework, the European SmartSoft framework, or the FiWare framework, ROS2 provides the best balance between technological capabilities, wide-spread support, especially with hardware vendors, and a fast, vibrant global developer community, which makes its long-term survival highly likely. Besides, ROS2 or its predecessor ROS is already in use at multiple project partners for prototyping and development, thus minimizing onboarding efforts.
ROS2's major drawback of lack of structure development tools is mitigated by longstanding and ongoing work from project partners to bring modern, model-driven development to the ROS world.

6.2 Advanced modularity

From the general "advanced modularity" concept, we have established a list of sub-concepts detailing our expectations (chapter 6). Then, those sub-concepts have been transformed into a first set of 11 software tools (chapter 7). Most of the tools we propose to develop will benefit from work previously carried out by Fraunhofer IPA. This minimizes our development effort while maximizing our chances of success.
Our development efforts will focus on eight tools:

- a data structure for component description;
- a database of components properties (using the above-mentioned structure);
- an automatic component properties extractor (from code);
- a component properties visualizer;
- a data structure for the system (robot);
- a tool to model the system as an assembly of components (robot);
- a tool that automatically generates code (except for the documentation);
- a tool for checking incompatibilities and missing information in a system model.

For two tools, the project will test existing solutions and make recommendations for use and/or further development:

- a visual tool to launch or stop robot software components;
- a tool that checks code quality.

For one tool, we will draw up recommendations for future developments:

- a tool that creates the robot's software distribution from its model.

The corpus of tools that will be available at the end of the project will provide a solid, concentrated core of Model Design Engineering tools with a strong impact on product quality and development productivity. Beyond the scope of the project, efforts will have to be continued to increase the

perimeter of certain tools (moving the system's data structure from the "logical" scale to the "physical" scale, then to the mechanical scale) or by undertaking work on tools that are not yet covered (software distribution tool).

The four tools that export constraints to developers have been identified. We have endeavoured to limit this number. The precise list of constraints will be known in January 2024, following the completion of a study conducted at the University of Stuttgart with the help of students.

6.3 Safety assessment

The Safety Plan is the document that summarizes all the elements to be produced for the safety demonstration. It is divided into 4 chapters (Purpose of the Safety Plan, System Definition, Safety Proof Concept, Safety Assessment Report), which represent the basic pillars of the safety case for the process change of maintenance measures in the rail system.

The first section “Purpose of the Safety Plan” indicates that the safety plan is a covering document which organizes and references the most important documents in the safety demonstration. For this reason, this document does not itself contain the concrete performance of the system components or the scope of the automation. This concretization takes place in the elements of the safety-proof concept. It also presents the two cases for which the document is designed:

- initial verification of the maintenance procedure change regarding the replacement of manual maintenance activities by a defined automated solution;
- verification of the maintenance procedure change involving the use of an evolved automated solution (the change may come from the procedure, the automated system, or both).

The second section “System Definition” allows the specific project to be broken down into its most important components and ensures that all aspects of the process change have been considered for proper safety verification. The overarching goal of the system definition is to create transparency on the purpose, intended environmental context, boundaries, and functions of the system.

The third section “Safety Proof Concept” structures the path that has to be followed to provide proof of safe operation. The path has been organized around five categories. Four of those five categories are based on the macroscopic machine functions (basic machine safety, movement safety, inspection safety and intervention safety). A unified categorization would have been more difficult to be achieved by working on technologies or components. These can be very varied in robotics. The last category (Information safety) is an exception. It concerns the communication of information between the robot and its environment. This enables us to emphasis cybersecurity issues, which are becoming increasingly important in our society.
For each category, we established what has to be demonstrated. Our original intention was also to suggest ways of establishing the "how" for each category. Unfortunately, it became clear to us that, here too, technological diversity makes it impossible to unify methods for measuring the

effectiveness of devices in meeting safety requirements.

All the elements mentioned above lead to the fourth and final section, a central safety assessment report.

This deliverable is the fruit of initial work that needs to be enriched. We will be working on two types of improvements over the coming months:

- we will be developing or continuing to develop templates and guidelines to help write the safety plan sections themselves.
    - this will be the case for "basic machine safety", where the relevance and contribution of the numerous standards to risk management in the railway context will be highlighted;
    - for the "system definition" a first template has been established. Its application to several of the project's demonstrators should enable it to be enriched;
    - a guide to the correct classification of a system in the categories useful for the "safety proof concept" will probably be necessary;
    - a template for the safety report will be developed alongside corresponding guidelines.
- while we have specified here the elements to be supplied for the safety assessment, we have not detailed how the various necessary activities are to be carried out. The second axis will be the development of a Unified Safety Process for Railway Maintenance Robots.

6.4 Vision of robotics impacts on railway maintenance

Possible approaches fall into 2 families: “bottom-up” and “top-down” methods. Due to the difficulties associated with the generalization stage, bottom-up approaches were quickly discarded. Therefore, various alternative top-down methods were examined. The approach adopted is a mix between a top-down “analytical approach” and a top-down “fictional approach”.

The “analytical approach” is based on a breakdown of maintenance into more basic processes. For each of the elementary processes, a short list of relevant indicators (in the context of the introduction of robotics) is proposed. Reference levels are determined. The last step consists of evaluating the evolution of these indicators on a scale of approximately 5 years.

The “fictional approach” is inspired by Red Team Defense offered by Paris Sciences & Lettres to the French armies. They propose, over a longer time horizon, futures for which the probability of occurrence is not the key point. It is the reactions to be implemented in the face of these new situations that have important value. Creating a collective imagination in addition to more traditional commercial relationships can also be a strong glue in a new-born ecosystem.

The total duration of the selected approach is 18 months, based on 4 stages for the analytical approach and on an iterative work of 6 to 9 months for the fictional part.

PART A – MIDDLEWARE SELECTION

1 Executive Summary

Objective:
The primary objective of the work within task 18.1.1 "*Selection of a common middleware*" is to evaluate options for a common software development middleware for robot control software for use in the project and beyond. The need for a common software and hardware development framework - or middleware - stems from the overall objective of introducing robotic automation solutions for maintenance tasks, ranging from inspection, to repair to construction, and operation tasks like regular cleaning into the railway sector. Developing a fit-for-purpose robotic system is a technically challenging and economically expensive undertaking. Given the structure of the railway sector and the size of individual operators and service providers specialized, individual solutions are economically unfeasible. Individual sector markets, like train inspection, simply do not offer sufficient market capacity for any robotic solution provider to invest in a "one-of" specialised robotic system.
Aiming at an open robotic hardware and software ecosystem, where components can be developed once and reused across multiple use cases, combined with development tools designed to minimize system integration efforts (traditionally 40% or more of development costs for robotics systems) creates the necessary market size for robotic component and solution providers to enter the railway sector. The common software development framework discussed here is a cornerstone of such an open and economically viable market.

Methodology:
We first present a survey of commonly used past and present robotic software frameworks. The candidates have been selected through a literature review of peer-reviewed papers dealing with software development in robotics. Common evaluation criteria have been extracted from the surveyed frameworks and adjusted to the requirements for a common software development framework. The list of candidate frameworks and the list of selection criteria have been discussed and refined in multiple online meetings with all relevant stakeholders in the project. Based on the agreed criteria, a decision for one candidate framework has been taken.

Conclusion:
The project selects the ROS2 software development framework as its common middleware. Although technologically more advanced options exist, like the Japanese ORiON framework, the European SmartSoft framework, or the FiWare framework, ROS2 provides the best balance between technological capabilities, wide-spread support, especially with hardware vendors, and a fast, vibrant global developer community, which makes its long-term survival highly likely. Besides, ROS2 or its predecessor ROS is already in use at multiple project partners for prototyping and development, thus minimizing onboarding efforts.
ROS2's major drawback of lack of structure development tools is mitigated by longstanding and ongoing work from project partners to bring modern, model-driven development to the ROS world.

2 Abbreviations and acronyms

Abbreviation / Acronym	Description
ACRoSeT	Arquitectura de Control para Robots de Servicio Teleoperados - framework for developing robotized systems
AERO-STACK	Software framework for aerial robotic systems
ARM 64	Advanced RISC Machines (and originally Acorn RISC Machine) is a family of RISC instruction set architectures for computer processors with 64-bit internal structure. A reduced instruction set computer (RISC) is a computer architecture designed to simplify the individual instructions given to the computer to accomplish tasks.
ArmarX	Event-driven component-based Robot Development Environment
ASEBA	Robot Development Environment (link to the Thymio robot)
BIP	Robot Development Framework
BSD license	BSD (Berkeley Software Distribution) licenses are a family of permissive free software licenses, imposing minimal restrictions on the use and distribution of covered software.
C/C++	C and C++ are high-level, general-purpose programming languages
CAMUS	Robot Development Framework
CANOpen	CANopen is a communication protocol and device profile specification for embedded systems used in automation.
CLARAty	Coupled Layer Architecture for Robotic Autonomy (CLARAty), which is designed to improve the modularity of system software
CMake	CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method.
CoolBot	Open Source Distributed Component Based Programming Framework for Robotics
CPU	A central processing unit (CPU) is the most important processor in a given computer. Its electronic circuitry executes instructions of a computer program, such as arithmetic, logic, controlling, and input/output (I/O) operations.
DDS	The Data Distribution Service (DDS) is a standard that aims to enable dependable, high-performance, interoperable, real-time, scalable data exchanges using a publish–subscribe pattern.
DSL	A domain-specific language (DSL) is a computer

Abbreviation / Acronym	Description
	language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains.
ehterCAT	Ethernet-based fieldbus system standardized in IEC 61158. It is suitable for both hard and soft real-time computing requirements in automation technology.
ESROCOS	Open source framework which can assist in the generation of flight software for space robots
EU-RAIL MAWP	Europe's Rail Joint Undertaking Multi-Annual Work Programme
Fawkes	A component-based Software Framework for Robotic Real-Time Applications for various Platforms and Domains
GeNOM	Generator of Modules is a tool to design real-time software architectures
Github	GitHub, Inc. is a platform and cloud-based service for software development and version control using Git, allowing developers to store and manage their code.
GUI	A graphical user interface, or GUI, is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicators instead of text-based UIs, typed command labels or text navigation.
ICE	The Internet Communications Engine (Ice) is an object-oriented RPC framework that helps you build distributed applications with minimal effort.
IDE	An integrated development environment (IDE) is a software application that provides comprehensive facilities for software development.
IEEExplore	Digital Library
Linux	Linux is a family of open-source Unix-like operating systems based on the Linux kernel.
LTS	Long-term support (LTS) is a product lifecycle management policy in which a stable release of computer software is maintained for a longer period of time than the standard edition.
MacOS	MacOS is an operating system developed and marketed by Apple Inc. since 2001.
MARIE	Middleware framework oriented towards developing and integrating new and existing software for robotic systems
MCA	Software framework with real-time capabilities
MiRo	Distributed object-oriented framework for mobile robot control
MIT license	The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology

Abbreviation / Acronym	Description
	(MIT) in the late 1980s;
MOOS	A C++ cross platform middle ware for robotics research
OCP	Robot Development Framework
OpenRDK	modular framework focused on rapid development of distributed robotic systems
OPRos	Open Platform for Robotic Services (OPRoS) is a component-based open-source platform, and it has Integration Development Environment (IDE) tools, a robot framework for robot operation, a server, and a test and verification tool.
Orca	Open-source framework for developing component-based robotic systems
ORiN	ORiN (Open Resource Interface for the Network) is a framework for applications that can handle a variety of resources, ranging from robots to databases and local files, in an integrated manner.
Orocos	Open Robot Control Software is a portable C++ library for advanced machine and robot control.
OS	An operating system (OS) is system software that manages computer hardware and software resources, and provides common services for computer programs.
PEIS-Ecology	Ecology of Physically Embedded Intelligent Systems, Framework for Robotics
Player/Stage	The Player Project (formerly Player/Stage Project) creates free and open-source software for research into robotics and sensor systems.
pocolibs	System communication and real-time primitive layers used by GenoM modules.
Python	Python is a high-level, general-purpose programming language.
QA	Quality assurance (QA) is the term used to describe the systematic efforts taken to assure that the product meets the expectations.
Robocup	RoboCup is an annual international robotics competition.
RoboFrame	Software framework tailored for heterogeneous teams of autonomous mobile robots
ROCK	Robot Construction Kit - a robot software development environment
ROS	Robot Operating System A collection of tools, software libraries and common practices to build sophisticated robot control software from reusable building blocks and a powerful communication layer. The de-facto standard in academia for robot software development.
ROS2	Version 2 of ROS, addressing most of ROS' architectural

Abbreviation / Acronym	Description
	and cyber-security shortcomings.
RPC	A remote procedure call (RPC) is when a computer program causes a subroutine to execute another computer on a shared network without the programmer explicitly writing the details for the remote interaction.
RSB	Robotics Service Bus (RSB) is a message-oriented, event-driven middleware aiming at scalable integration of robotics systems in diverse environments.
RSCA	Robot Software Communication Architecture provides a standard operating environment for robot applications together with a framework that expedites the development of such applications.
RT-Middleware	RT-middleware (Robotics Technology Middleware) is a common computing platform technical standard for robots based on distributed object technology
Ruby	Ruby is an interpreted, high-level, general-purpose programming language.
SBC	A single-board computer (SBC) is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output and other features required of a functional computer.
SCOPUS	arge, multidisciplinary database of peer-reviewed literature: scientific journals, books, and conference proceedings
SLICE	Slice is the interface definition language used by Ice. With it, you can define the client–server contract for your application, including all of the interfaces, operations, parameters, data types, and exceptions.
SmartSoft	Framework for developing component-based robotics systems
TCP/IP	The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP).
UDP	The User Datagram Protocol (UDP) is one of the core communication protocols of the Internet protocol suite used to send messages (transported as datagrams in packets) to other hosts on an Internet Protocol (IP) network.
Unix	Unix is a family of multitasking, multi-user computer operating systems that derive from the original AT&T Unix.
Windows	Microsoft Windows is a group of several proprietary graphical operating system families developed and marketed by Microsoft.

Abbreviation / Acronym	Description
WP	Work Package
X86_64	x86_64 is a family of complex instruction set computer (CISC) instruction set architectures with 64 bits internal structure.
XBotCore	Light-weight, Real-Time software platform for robotics
XML	Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data.
YARP	Yet Another Robot Platform (YARP) is a software framework designed to allow different components of a robot system to communicate with each other and to interact with the outside world.
YCM	YCM contains a set of CMake files that support the creation and maintenance of repositories and software packages. YCM has been written to solve some of the problems encountered while managing large research projects but it can be used outside its initial context.

3 Objective/Aim

Within the project we aim to establish a modular, versatile robotics platform for performing various maintenance tasks in the railway sector. Such maintenance tasks can range from the inspection of trains, infrastructure (tracks, signalling…), and stations, to the cleaning, actual repair, or construction works. Given the shortage of skilled workers across Europe, which is only aggravated by our ageing societies, it seems natural to automate processes and use robotic solutions to relieve human workers from tedious, physically hard, or dangerous work.
Today's robotic solutions are predominantly designed for indoor use in highly structured environments that are designed with robot automation in mind, and are only economically feasible when run at capacity for long periods. Today's robotic solutions are tailored for purpose, with the prime example being the highly optimised automotive production lines. Such type of robotic automation is not economically feasible in the railway sector, particularly in maintenance, due to the disproportional high variance in tasks at rather low volume per task.
Instead of designing one individual robot system "from scratch" for each task (and specifically tailored to that task), we need to aim for a **robotic ecosystem** that maximizes the reuse of components on the hardware and software side. This allows us to build task-matching robotic systems with significantly reduced construction efforts from pre-existing building blocks. Only by maximising the reuse (and keeping the number of specialized components low) we can reach an "economy of scale" where it becomes economically viable for robot technology providers to enter the railway sector.

This document looks at the software side of building robotic systems. Although a robot is a physical entity, its versatility and performance are primarily software-defined. This is even truer for robots working in lesser structured environments or facing a significant variance in their tasks.
Specifically, this document provides an overview of available frameworks for modular robot software design and software component reuse. The document lists possible selection criteria for the choice of the main framework used throughout the whole project, names the selected framework, and provides the reasoning for its selection.

4 Methodology

To select a common middleware framework for the project, a structured approach focused on multiple online workshops has been chosen. Deploying an open, interactive approach ensures early and transparent involvement of all relevant stakeholders. The workshops have been prepared by Fraunhofer IPA, working as an associate entity to SNCF in the project. Fraunhofer IPA has a track record in robot software development and is one of the leading research institutions in the world in the field of open-source robotics, and robot system design. The process of selecting a common middleware or software development framework had four stages:

- surveying the available options;
- presenting and discussing available frameworks with the consortium;
- deriving common selection criteria;
- selecting a candidate framework based on available candidates and common selection criteria.

The unusual approach of first surveying without previously defining a set of requirements for the wanted framework has a two-fold reason: First, taking advantage of parallel work in a PhD thesis project recently started at Fraunhofer IPA, which was looking at robotic software frameworks at a larger scale. Second, ensured realistic expectations about the state of the art in software frameworks, which enabled us to define clearer and measurable requirements.

4.1 Survey of robotic software frameworks

4.1.1 Common Middleware vs. Software Development Framework

Although the deliverable scope is task 18.1.1 "*Selection of a common middleware*", it would not be sufficient to use only a middleware. A mere middleware in the narrow technical sense would not be sufficient. To enable a robotic (software) ecosystem we need more than the ability to exchange messages between parts of a system, which is the typical task of a middleware. We need a full-fledged component framework that promotes the development and use of interchangeable hard- and software components, that can be composed into a larger system with the least possible effort. While a common middleware (in the narrow sense of the word) is a precondition for any such component framework, any useful framework adds a set of tools and best practices on how to implement components and make use of the middleware. Specifically, it adds in common, built-in infrastructure components for managing the middleware and the system lifecycle. As such, we only looked at full software development frameworks in use in robotics over the last decade, and not just at the underlying types of communication middleware. Some of the surveyed frameworks support multiple types of communication middleware.

4.1.2 Initial data selection

The main methodology is depicted in Figure 1. The study had been performed by our colleague Christoph Hellmann-Santos as part of his PhD work at Fraunhofer IPA and is as of yet (December 2023) unpublished. Data and presentation here are used with permission.

The selection process was initially conducted in two parallel search phases, one based on keywords, the other on citations. The results of the 2 search approaches were then combined.

The study started with a keyword search across relevant online publication databases, in particular IEEEXplore and Scopus, yielding 208 candidate papers. Quality and relevance thresholds (131 papers out of 208) and duplicate removal (20 papers out of 208) yielded 57 candidate papers, each expected to describe one framework in depth or comparing multiple frameworks. Pre-screening excluded 23 papers (out of 57) for not falling in the scope of the survey based on abstract and other publicly available information. For the remaining 34 papers full text was retrieved. Of those two were excluded for discussing the same framework discussed in other candidate papers, 11 were excluded for not providing substantial information, e.g. only providing citations, and 11 more were excluded for not proposing a real framework.

24 additional papers were identified by citation search, which were not covered by the previous keyword search. Pre-screening and screening have also been applied to the 24 additional papers identified by citation search. None were excluded.

This yielded 34 frameworks, each covered by at least one comprehensive paper, for inclusion in the study.

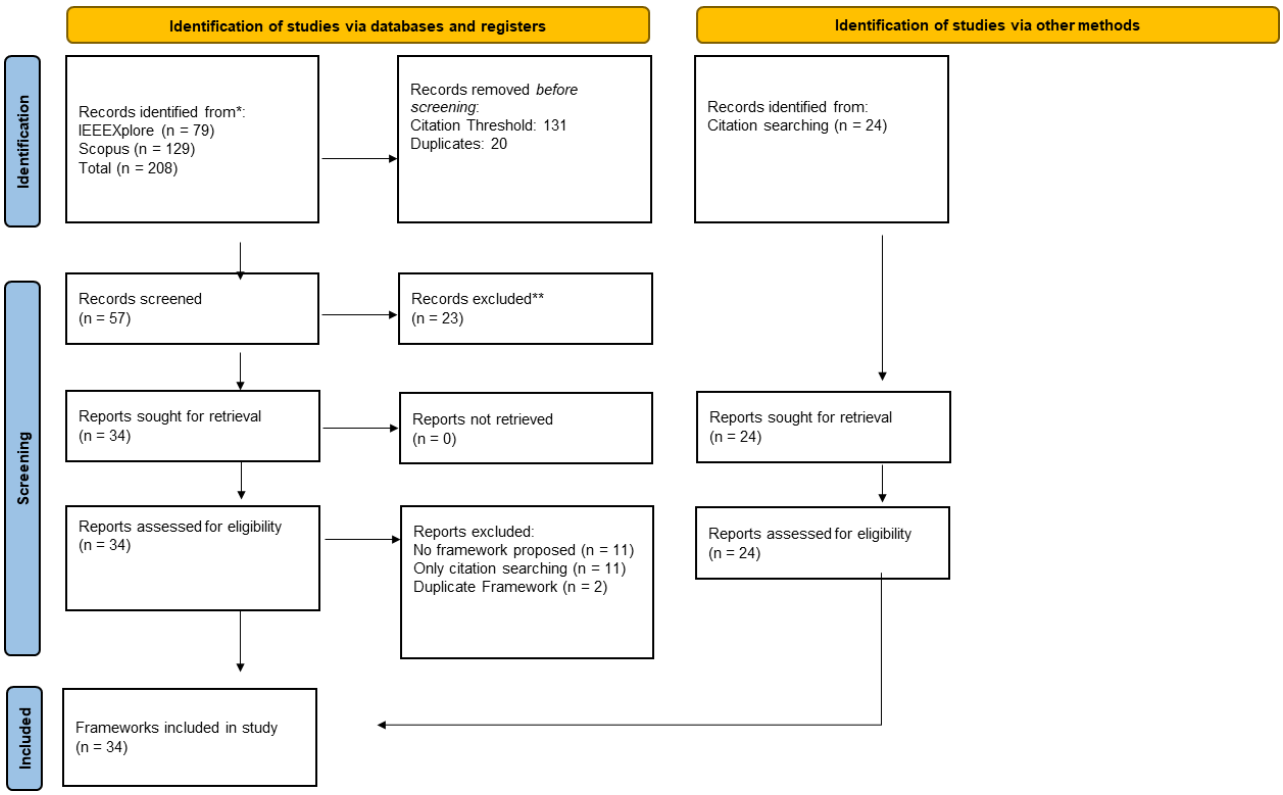

Figure 1 Flow chart for identification of candidate frameworks for comparative study.

4.1.3 Preliminary analysis and filtering

The survey identified 34 software frameworks for robotics. These can be in part grouped into framework families, as shown in Figure 2. The surveyed frameworks and their ecosystems vary significantly in terms of size, approach, community and used terminology.

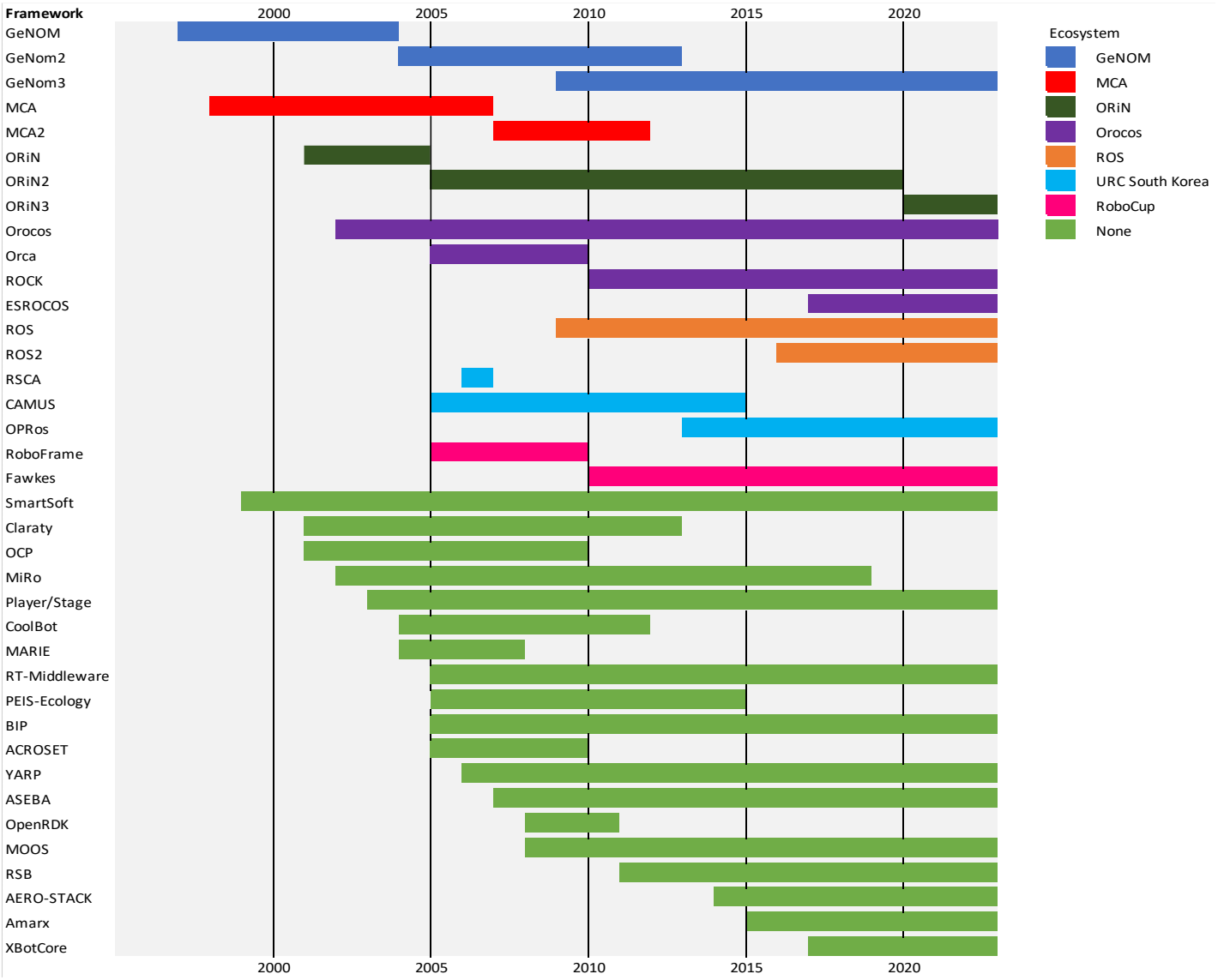

Figure 2 Family inheritance chart for surveyed software frameworks.

For this study, comparison criteria were split into two broad categories:

1. Technical criteria:
    a. type of software artefacts present in a framework or ecosystem;
    b. architectural design of the framework.
2. Social criteria:
    a. type of (defined) roles in the community supporting the framework or ecosystem;
    b. rules and practices to ensure quality.

For some criteria, further sub-criteria or indicators were defined. In particular, the varying terminology used in different surveyed frameworks was unified for proper comparison. The following unified terms were used throughout the survey:

For Artefacts (criterion 1.1):

1. *Library:* Source/Binary piece of software providing (elementary) routines within a specific task domain.
2. *Component:* An executable piece of software, usually based on multiple libraries, used to achieve an objective.
3. *Package:* Deployment artefact, a combination of libraries and components, targeted at a specific objective.
4. Bundle: Sort of application, combining one or more packages into a ready-to-use entity to achieve an objective.
5. *Distribution:* Combination of bundles and other packages, all compatible with each other, addressing multiple objectives.

For Roles (criterion 2.1):

6. *Library developer:* People maintaining the source code of a library.
7. *Component developer:* People building an executable, often reusable, entity solving a specific task.
8. *Packager:* People combining components and libraries with development, deployment and dependency information.
9. *Bundle developer:* People combining packages into something solve an application-level task.
10. *Distributor:* People combining compatible packages, components and low-level libraries.

Eligibility elimination

Two initial requirements that must be met by any framework to qualify for further analysis were defined: availability of a package management system (needed to support distributed development and separation of component developers from users) and active maintenance. Obviously, we can only build a future ecosystem of robotics for railways on top of an actively maintained framework. The following Table 1 lists all 34 frameworks and their maintenance and package management status.

Table 1 All 34 surveyed robotic software frameworks and their maintenance status

An empty cell means no data is available. A ✓ means the feature is present. A ✳ means the feature is partially present (but e.g. not well supported or maintained). A ✗ means the feature is not present.

Framework	Source	Active	Pkg. mngnt
ACRoSeT	Iborra et al.(2009)	✗	✗
AERO-STACK	Sanchez-Lopez et al.(2016)		
ArmarX	Vahrenkamp et al.(2015)	✓	
ASEBA	Magnenat et al.(2007)	✗	✗
BIP	Basu et al.(2011)	✗	
CAMUS	Kim et al.(2005)	✗	✗
CLARAty	Volpe et al.(2001)	✗	✳
CoolBot	Dominguez-Brito et al.(2004)	✗	✳
ESROCOS	Arancon et al.(2017)	✓	✓
Fawkes	Niemueller et al.(2010)	✓	✳
GeNOM	Fleury et al.(1997)	✓	✓
MARIE	Cote et al.(2004)	✗	✗
MCA	Scholl et al.(2001)	✗	✳
MiRo	Utz et al.(2002)	✗	✗
MOOS	Benjamin et al.(2010)	✓	
OCP	Paunicka et al.(2001)	✓	✗
OpenRDK	Calisi et al.(2008)	✗	✗
OPRos	Han et al.(2012)	✗	
Orca	Brooks et al.(2005, 2007)	✗	✳
ORiN	Mizukawa et al.(2002)	✓	✳
Orocos	Bruyninckx et al.(2003)	✓	✓
PEIS-Ecology	Saffiotti and Broxvall(2005)	✗	✗
Player/Stage	Player/Stage Gerkey et al.(2003)	✗	✳
RoboFrame	Petters and Thomas(2005)	✗	✗
ROCK	Joyeux and Albiez(2011)	✓	✓

Framework	Source	Active	Pkg. mngnt
ROS	Quigley et al.(2009)	✓	✓
ROS2	Thomas et al.(2014)	✓	✓
RSB	Wienke and Wrede(2011)	✗	
RSCA	Yoo et al.(2006)	✗	✗
RT-Middleware	Ando et al.(2005)	✓	✳
Smartsoft	Schlegel and Worz(1999)	✓	✳
XBotCore	Muratore et al.(2017)	✓	
YARP	Metta et al.(2006)	✓	✓

4.1.4 Candidate frameworks detailed analysis

After eligibility filtering, keeping only the frameworks that are active and have (or potentially have) package management, only 11 frameworks remained. The following section briefly introduces each remaining candidate:

ArmarX

ArmarX is a robot programming environment developed by the Karlsruhe Institute of Technology for humanoid robots with a strong focus on the ARMAR robot series. ArmarX is organised in three layers, the middleware layer, framework layer and application layer. ArmarX's middleware is built upon ICE which uses the SLICE interface definition language. The middleware provides basic building blocks called ArmarX-Objects and an ArmarX-Manager that makes sure all necessary ArmarX-Objects necessary for a process are available and loaded. ArmarX applications are composed of two different entity types, state charts and components. While components create an interface to an algorithm and state charts execute a process based on the components. ArmarX provides a developer tool for package management as well as a deployment tool. ArmarX is actively maintained and used on the ARMAR robot series.

Properties

- Libraries: In ArmarX libraries are handled as system dependencies in CMake files that need to be installed manually or as ArmaX packages.
- Components: ArmarX components do not have a specific interface description, they are defined by their source code.
- Packages: ArmarX packages are defined by their structure and CMake Build Instructions.
- Bundles: ArmarX supports bundles via so called statecharts. These are modelled in a development environment, C/C++ code is generated and finally manually adapted. Bundles are handled in the same way as other packages. It is unclear, if nesting of state charts is possible.

- Distribution: ArmarX provides code components in maintained repositories with coordinated release tags. A package index is not available.

Fawkes

Fawkes is a robot software framework that was designed to meet the requirements of software for controlling robots. Fawkes is mainly used in the area of service robotics and has been thoroughly tested in RoboCup competitions. The core of the framework are a communication mechanism and a component concept. In Fawkes, components are plugins, that can be loaded at runtime and communicate via defined interfaces using a blackboard communication mechanism. The blackboard communication mechanism is realised using shared memory but there is also support for remote access. Fawkes comes with a set of plugins for common robotics problems such as navigation which are stored in the main repository. Fawkes was actively maintained until mid-2022.
Due to the lack of recent activities, the "Properties" section has not been compiled.

GeNOM

GenoM is a robot software framework which uses a specification language to describe and generate robot software components and was introduced by Fleury et al.(1997). It originally used pocolibs as middleware but has since then been adapted to OROCOS and is since version 3 abstracted from middleware. The software is available under open source license (BSD licence). GenoM is part of the open robots architecture. It does provide a formal description for each module and enables integration of the specified modules. GenoM packages are managed by the robotpkg tool.
Properties

- Libraries: Can be packaged.
- Components: Are modelled using a DSL. Components can be generated for different middleware types.
- Packages: Are described using robotpkg approach which will define Build instructions, dependencies, package description, source code archive and other information about the package.
- Bundles: Are not supported.
- Distribution: robotpkg provides a collection of available packages.

MOOS and MOOS-IvP

MOOS is a mission oriented operating suite targeting marine robots. The suite consists of two parts first MOOS which is a robotics middleware and second MOOS-IvP which is a set of open source modules that provide autonomy to for marine robots. The middleware provides a simple publish and subscribe communication mechanism implemented based on TCP/IP that uses a central blackboard called MOOSDB. The autonomy layer provides modules for core autonomy, behaviours, simulation, mission control and more. Moos and Moos-IvP are distributed separately. A distribution contains all available modules in one repository. The framework is actively

maintained.
Properties

- Libraries: In MOOS libraries are handled as dependencies of a module.
- Components: Components are called modules in MOOS-IVP.
- Packages: MOOS does not differentiate between components and packages. Modules are described by a manifest file.
- Bundles: MOOS does have implicit and limited support for bundles. MOOS provides pAntler and a launch description file format (.moos) for launching multiple processes. The launch description files seem to not support import functionality for subsystems specified in other ".moos" files, this limits the capability to specify bundles.
- Distribution: MOOS and MOOS-IVP have a distribution that clusters all modules in a single source code archive. Distributions are described using a manifest file. MOOS does not have automated package management tools.

ORiN

Mizukawa et al.(2002) introduce ORiN. ORiN focusses on connecting applications running on a PC to an industrial PC. The framework uses RRD to describe components, the RAO engine to create a RAO object that represents a robot during run-time and exports robot variables that can be accessed by the robotic application that is running on the PC. The Orin framework is active as the ORiN Consortium and has published specifications. Regarding package management, the framework is shipped as an SDK that includes most available components. This framework is used almost exclusively in Asia, obth in industry and academia. Beyond its possible technical potential, this is a limiting factor for its use in our context.

Properties

- Libraries: Visual Basic, Visual C, C++, Delphi, Labview
- Components: Visual Basic, Visual C, C++, Delphi, Labview
- Packages: no difference between components and packages described as plugins. core elements include in the SDK
- Bundles: uncertain notion in this environment
- Distribution: through the SDK

Orocos

Bruyninckx et al. (2003) introduce the robot software framework Orocos. The framework focusses on providing an open-source real-time framework for simplifying research on servo algorithms, motion interpolators and inter-process communication. The framework consists of a kinematics and dynamics library (KDL), a baseyian filtering library (BFL), real-time finite state machine (rFSM) and the real-time toolkit (RTT). The framework is actively maintained. The Orocos toolchain uses the tool autoproj for package management. The tool uses so called package sets to list and describe packages to build and install from source.
Properties

- Libraries: C/C++, Python.
- Components: C/C++, Python (no real distinction between libraries and components).
- Packages: Implemented as simple Folders and CMake scripts.
- Bundles: Implemented as Orocos script artefacts.
- Distribution: Only core components, no ecosystem distribution.

ROCK

The robot construction kit (ROCK) [33] builds on OROCOS. ROCK extends OROCOS with additional components but the main addition is the orogen toolchain. The orogen toolchain is a model-driven approach for designing components based on OROCOS' real-time tool kit. The model driven approach leverages a ruby based DSL for describing components and composing systems. ROCK is actively maintained and has a large component ecosystem. ROCK uses autoproj as package manager. Package collections are defined in package sets. Package management focusses on fetching a packages source code which is then built locally.

Properties

- Libraries: ROCK handles libraries as separate packages that can be included into ROCK components.
- Components: ROCK components are described by an oroGen specification which is written in Roby a Ruby-based DSL, which indicates communication ports and configuration parameters and other information about the component.
- Packages: ROCK packages are described by a manifest.xml, that follows the package manifest specification. Build instructions for packages are handled by CMake.
- Bundles: ROCK supports bundles. Bundles are described in roby language and can be run with the syskit tool. Bundles itself are handled as packages and contain a manifest.xml as well.
- Distribution: ROCK packages are distributed using package sets. Package sets are stored dispersedly in different GitHub organisations that host rock packages. An incomplete index of packages exists on the rock website.

ROS/ROS2

ROS, the robot operating system was first presented in 2009. ROS provides a programming framework and communication mechanism for creating complex robot software. ROS enables creating a computation graph consisting of nodes (processes) and edges (communication). ROS provides three different communication paradigms: publish and subscribe, service/client and action/client communication. ROS also comes with a rich suite of tools to develop, launch, debug and analyse ROS systems. A large component ecosystem exists for ROS as well as a method for creating ROS distributions of compatible packages. An index of packages that are released into ROS distributions is available.

Properties

- Libraries: In ROS external libraries are handled as system dependencies or if not available a vendored ROS package for the library is created.
- Components: ROS components are not specifically described. Their interface is characterized by their source code.
- Packages: ROS packages are described by a package.xml file that follows the package manifest specification. ROS packages use CMake with catkin extension for build instructions.
- Bundles: ROS supports bundles implicitly using launch files and package descriptions. The launch file indicates which components to launch and how to connect their communications interfaces.
- Distribution: ROS packages are distributed via rosdistro in different distros (versions). ROS package maintainers can release their package into rosdistro via an automated tool called bloom. ROS distros are available as binary packages for certain target operating systems.

RT-Middleware

Ando et al.(2005) introduce RT-Middleware which started in 2002. Its purpose is to establish technologies for integrating robot components into robot systems. RT-Middleware describes a component architecture that each component needs to implement. Components can be assembled using a GUI tool, script language or xml file. An open source implementation of the middleware exists and is actively maintained. An online catalogue of existing components is available. Apart from the catalogue no package management system exists.

Properties

- Libraries: In RT-Middleware external libraries are handled as dependencies on component level.
- Components: RT-Middleware components are described using an xml file.
- Packages: RT-Middleware does not have a package manager. Common practice seems to be a separate source code folder per component that includes build instructions and some information about dependencies.
- Bundles: RT-Middleware supports bundles implicitly using RT-System description files in XML format. RT-System files can be created using the RT Development Environment.
- Distribution: RT-Middleware does not provide a distribution of ecosystem packages. An index of available packages is available online, but packages need to be downloaded manually and many links are dead.

SmartSoft

Schlegel and Worz(1999) introduce SmartSoft as a software framework to implement sensorimotor systems. The framework not only dictates modularized software components but

also contains structural rules and templates for robotics systems. Stampfer et al.(2016) present updates to the SmartSoft ecosystem that focus on methodologies and tools for model driven composition of components. The SmartSoft framework features ready to use packages, however there is no dedicated tool for managing these packages and a commercial component marketplace is available.

Properties

- Libraries: Are not managed by SmartSoft and need to be installed manually by the user.
- Components: Are modelled using a graphical IDE and the code structure is generated in C++.
- Packages: Are defined as Eclipse projects.
- Bundles: SmartSoft supports bundles via so-called systems. These are modelled in a development environment.
- Distribution: SmartSoft does not provide a distribution. There are however GitHub repositories with common components and systems. No package management tool is available, manual download is required as well as manual dependency installation.

YARP

Metta et al.(2006) introduce yet another robot platform (YARP). YARP was developed for meeting the requirements of humanoid robotics development. Humanoid developers are facing fast changing hardware platforms. Therefore, software reuse is very important. YARP supports this by introducing concepts for modularity, abstraction and platform independence. YARP is available as an open-source library and actively maintained. To deal with the bleeding edge nature of the YARP ecosystem, the source management and build tool superbuild was introduced.

Properties

- Libraries: In YARP external libraries are handled via CMake or YCM.
- Components: YARP components are not specifically described. Their interface is characterized by their source code. YARP has standard component interfaces for a number of component classes.
- Packages: YARP packages are defined by their structure and CMake file.
- Bundles: YARP supports bundles implicitly using robot interface xml files that describe components that need to be launched for a specific subsystem.
- Distribution: YARP does not provide a distribution, packages are hosted dispersedly in repositories at different version control hosters. The robotology organization provides a set of packages managed by YCM, which is the closest thing to a distribution in the YARP ecosystem.

4.2 Common requirement criteria

Defining common requirements criteria turned out to be a challenging task, due to limited experience with different robot software development frameworks in the consortium. For this

reason, we first conducted and presented the survey on existing frameworks (section 4.1) and then started to collect requirements for the future IAM4RAIL common framework. Requirement criteria solicitation was carried out in multiple online workshops with interested partners from the consortium from March to May 2023. This resulted in a set of technical and a set of non-technical requirements and a set of overarching or general requirements.

The identified general requirements are:

- support modular hardware and software;
- rich ecosystem of available components;
- long-term support from a strong community;
- robust and mature technology;
- suitable for modern software engineering methods.

These general requirements were in subsequent discussions refined into better technical and non-technical requirements. The non-technical requirements were dubbed “social requirements” for better distinguishability. The following initial requirements served as starting point for the discussions to refine the technical requirements into criteria that could be used to rank the remaining 11 candidate frameworks from the survey:

- decoupling of drivers and algorithms;
- real-time support;
- support of multiple languages wanted (Python, C/C++, …);
- support of multiple OS;
- support for closed-source components as well as open-source components.

Correspondingly, the following initial requirements served as a starting point to refine the “social” requirements:

- rich set of ready-to-use components;
- middleware must be useful past project runtime;
- long-term support from the existing community needed;
- need a defined way to collaborate and influence the future direction of framework/middleware

The discussion of criteria yielded additional questions regarding the relevance of some previously discussed criteria to guide the selection process:

- Is the support of multiple OS really a key point or is Linux support sufficient?
- Do we need the compatibility with multiple CPU architectures: x86_64, ARM 64
- Do we also need compatibility with light-embedded systems like microcontrollers?
- Are there significant differences between the frameworks in terms of performances: min latency (or max frequency), bandwidth… or is the performance so high that it doesn’t really matter? How does the performances depend on other elements of the system?
- Is it possible to measure the robustness of the core components (the ones that provide the “middleware functionalities”)? Are there significant differences?

- Some middleware (at least ROS2) seem to use DDS some others do not. What are the impacts of this choice?
- Which technical properties can we attach to the certifiability of a middleware?
- Which technical properties can we attach to the maintainability of a middleware? Degree of adoption of external standards? Is it feasible to compare the number of proper lines of codes used for the core functions?
- The study "Comparison Study of Robotic Middleware for Robotic Applications" by Gergely Magyar claims for the Orocos middleware "modules have high quality from the technical point of view". Is that really based on a technical concept or is that because of the more rigorous work of the developers? Can we close the gap with our layer for advanced modularity?
- Which tools do we need? Simulation? IDE? Graphical IDE? Data visualization? Data recording and playback?
- A study (https://www.hindawi.com/journals/jr/2012/959013/tab2/) mention different properties: fault tolerance, distributed environment, dynamic wiring and safety. What does it mean?
- The same study mentions a "control model" property. What could be the impact of this property for us?
- Concerning network protocols (RPC services, TCP, UDP…) are there differences that should draw our attention because these choices may significantly restrict certain uses?

While some of these additional guiding questions address measurable properties of the frameworks, other serve more as a means to clarify what we actually want from our common framework.

The following answers have been identified (questions not listed in Table 2 have been dropped for varying reasons):

Table 2 Guiding question when defining selection criteria with answers.

Guiding question	Resolution
Is the support of multiple OS really a key point or is Linux support sufficient?	No. We expect to run Linux on board of the robot. Everything not running Linux will likely be specialized hardware/software combinations existing outside the common framework and interface with it through some sort of software bridge.
Do we need the compatibility with multiple CPU architectures: x86_64, ARM 64	Yes, at least support for ARM based SBCs is needed
Do we also need a compatibility with light embedded systems like microcontrollers?	No
Are there significant differences	Yes, but they are hard to quantify and depend on the

Guiding question	Resolution
between the frameworks in terms of performances: min latency (or max frequency), band width… or is the performance so high that it doesn't really matter? How does the performances dependents on other elements of the system?	task, and system resources. Also, it is unclear if observed differences really are a consequence of architectural properties of the framework, or mere side-effects of the implementation of some algorithms. A proper experimental validation was deemed not worth the effort.
Is it possible to measure the robustness of the core components (the one that provide the "middleware functionalities")? Are there significant differences?	Theoretically possible to measure but challenging in practice and requiring a multitude of the efforts available in the work package. Empirical measuring the ROS robustness in a real-world setting has been done at IPA in a different project and had consumed several person months of work.
Some middleware (at least ROS2) seem to use DDS some others do not. What are the impacts of this choice?	DDS, if implemented and used correctly, offers (hard) real-time guarantees (on hardware that can support this), as well as data integrity and data security (cryptographically strong authentication and authorisation)
Which technical properties can we attach to the certifiability of a middleware?	Unknown, to be addressed in WP18.2. However, and in light of standards like IEC61508-3, we assume that traceability, availability of source code and (for open-source projects) a well-defined governance structure will contribute.
Which technical properties can we attach to the maintainability of a middleware? Degree of adoption of external standards? Is it feasible to compare the number of proper lines of codes used for the core functions?	In principle we can use established code quality metrics to gauge the maintainability of a software framework. However, the question is, which parts to measure: Supporting tools? Core components? The actual underlying middleware? A simple measure like number of proper lines of code can be misleading, especially when the frameworks in question offer significant differences in functionality.
Which tools do we need? Simulation? IDE? Graphical IDE? Data visualization? Data recording and playback?	No clear consensus could be reached on what is needed. For example, tools for data visualization are not framework specific. Surveyed frameworks do not come with own IDEs, although some come with framework-specific plugins for common IDEs or specialised tools to handle certain development steps that are framework specific.

4.2.1 Final set of technical criteria

The following set of final “technical” requirements have been identified:

1. Performance of middleware
   1.1. Data throughput and latency
2. Supported OS types and versions
3. Supported compute platforms (i.e. CPU Architectures, SBC vs. workstation etc.)
4. Average release lifetime
5. Number of supported robots, mobile bases, sensors etc.
6. Bus systems or network protocols supported

4.2.2 Final set of “social” criteria

The following set of final “social” requirements have been identified:

3. Number of active developers
4. Number of scientific publication referencing / using middleware
5. Number of university courses using a middleware for exercises or as main topic
6. Number of commercial entities openly supporting the middleware or offering drivers etc.
7. Does an active governance body exist?
8. Are contribution guidelines / processes well defined?
9. How does quality assurance work?
10. Does a defined QA process exist?
    a. For core parts?
    b. For components?

4.3 Selection of the common middleware framework

For the final selection of a common middleware, we developed a scorecard based on the previously identified criteria. The criteria were further grouped in categories and their importance was defined on the scale F0 to F4 and “informative” for criteria that either can’t be quantified or where the quantified value does not directly bear meaning or comparability towards the goal of ranking the frameworks. Some criteria have been reworded or merged together for practical reasons in the process. The importance scale is defined in Table 3 and **Error! Reference source not found.** shows the score card.

Table 3 importance scale for selection criteria.

F0	Excluding criteria, if not given no candidate
F1	Strongly needed, if more than 5 not given, no candidate
F2	Needed
F3	Wanted
F4	Nice to have
(Informative)	Data that informs other criteria or is not directly

	usable to compute an overall score.

The final selection of the common middleware framework took place in a technical online meeting of all relevant partners on May 30th, 2023.

Table 4 - Selection criteria organized into a scorecard to rank different frameworks.

criterion class	criterion	Importance
Performance of middleware		(not applicable)
	Data throughput and latency	(Informative)
OS support		(not applicable)
	Linux	F0
	--Linux Distributions	(Informative)
	Windos	F3
	MacOS	F3
Architecture support		(not applicable)
	X86_64	F0
	Arm	F0
	GPU accelleration	F1
	mCU	F4
Relase criteria		(not applicable)
	Apllications run unchanged for 5+ years	F0
	--Middleware Release lifetime	(Informative)
	--Middleware release mode	(Informative)
Hardware support		
	Supports multiple robor arms	F2
	Supports multiple sensor classes	F2
	Supports multiple mobile bases	F4
	Number of maintained driver components	(Informative)
	Number of vendor-maintained driver components	(Informative)
Communication support		(not applicable)
	Canbus	F2
	Ethercat	F3
	TCP/IP	F2
	Reliable communication	F1
	Lattency / Throughput controll	F3
	Tamper resistence (IT-Secutity)	F2
	Peer-to-Peer component communication	F1
	Dynamic rewiring (fault tolerancy)	F2
Software quality & Governance		(not applicable)
	Defined project governance rules	F3
	Defined contribution guidelines	F4
	Defined Quality assurance rules	F4
	Quality of core parts	F1
	--Public CI pipeline for core parts?	(Informative)
	Quality of ecosystem parts	F3
	--Public CI pipeline for ecosystem parts?	(Informative)
	Core actively maintained	F1
	--mumber of open core issues	(Informative)
	--average time until core issues are closed	(Informative)
	--active committer in last 12 months	(Informative)
	Number of vendors supporting components	(Informative)

Based on a review of the defined criteria and the corresponding match of candidate frameworks,

existing experiences with frameworks (ROS and FiWare), and long-term maintainability considerations as well as short term considerations regarding timely implementation of the project's demonstrators, the participating parties unanimously opted for ROS2 as the common framework.

A partially completed version of the table 4 is available in appendix B. We were able to collect information on all middleware for just 3 criteria. Other information is not directly accessible (for example in the online available documentation). Searching those elements require a time-consuming work. To optimize our limited resource, we have chosen the middleware whose characteristics were best known and which, as we've already said, is the most used by the partners.

4.3.1 Justification for selecting ROS2

Primary selection reason is the high score in the "social" criteria. ROS2 has by far the largest commercial support from all surveyed frameworks. It is the de-facto standard framework in academia and used in the majority of all practical robotics university courses. It is the framework of choice in the international RoboCup competitions.
Commercial use is backed by the ROS-Industrial consortium, an association of robot system provider, sensor providers and automation equipment providers. The European branch of the ROS-Industrial consortium has 32 members (as of 2022) from hard- and software providers to end-users applying ROS in their daily business operation. The European, American and Asia/ Pacific consortia combined have 85 member organisations[1]. The statistics page of ROS lists almost 2,500 packages[2] as of June 2023 and over 70,000 visits to the packages index in that month.
The core components are all Open-Source Software under a permissive licence, usually BSD or MIT and available on GitHub[3]. It has a vibrant international developer community with regular developer events like the regional ROSCon conferences, online meetups and a defined Governance structure[4]. The overall ROS project steering rest with the Technical Steering Committee (TSC), while specific technical topics are discussed in Working Groups. The process to join the TSC as well as the process to initiate new Working Groups is publicly described. Contributing guidelines support and welcome new developers and new ideas for enhancing ROS for everyone[5].
On the technical side, its main target platform is Ubuntu Linux with its five years stable LTS (Long Term Support) releases. ROS itself is release more frequently in so-called ROSDistros, collections of package versions that match each other. The current ROS2 latest release is called "Iron Irwini" and has been released May 23rd, 2023. The current stable LTS release is "Humble Hawksbill" and will be supported until May 2027. The next LTS release would be expected in early 2026, providing for one year overlap, although no release date has been fixed so far. As ROS runs on top of Linux, it runs on the same wide set of hardware architectures as Linux, although some packages with

[1] https://rosindustrial.org/ric/current-members
[2] https://metrics.ros.org/repos_table.html
[3] ROS-Industrial repositories: https://github.com/ros-industrial
General ROS repository overview: https://github.com/ros/
[4] https://docs.ros.org/en/iron/The-ROS2-Project/Governance.html
[5] https://docs.ros.org/en/iron/The-ROS2-Project/Contributing.html

special hardware requirements may not be available on all platforms. For example, in June 2023 about 10% of package downloads were for the ARM architecture. Ports for MacOS and Microsoft Windows exist. The main target platform of (Ubuntu) Linux and the long-term support matches our requirements in IAM4RAIL. ROS2 supports all major communication protocols needed, including plain TCP/IPA, EtherCAT, DDS and CANOpen.
All partners in the consortium involved in software development for the demonstrators have prior knowledge in ROS (although not have experience with the specific differences of ROS2).

5 Conclusions

Within the wider IAM4RAIL project, WP18 aims at introducing robotic solutions to maintenance and service tasks within the railway ecosystem. Introducing specialized robotic solutions is only economically viable, if there is a sufficiently large market for such robot systems to justify the development costs for robot system providers, and if the productivity gain in the railway system justifies the total cost of ownership for such robotic systems. It is well known that standardisation and a component-based approach to robot system design can significantly reduce development time and costs, while simultaneously enhancing robustness, maintainability and potentially ease the certification process. Therefore, and to support interoperability across the railway sector, WP18.1 aims at enabling development of robotics systems in the railway sector around a common development and component framework.

This work, Task 18.1.1, was aimed at identifying an existing robotic software framework, which can be used as such a common development and component framework. A consensus had to be reached on which framework to use throughout the project's demonstrators. A key feature needed for any such framework is significant support from relevant hardware vendors (robot arms, mobile bases and all sorts of sensor and actuators) and a very high probability to be still around and well maintained in 10+ years.

We employed a three-step process to select a common framework: First we conducted a survey about existing development frameworks in robotics. Second, we derived metrics to describe and compare these frameworks and defined our own project's requirements for the common framework. Third, throughout multiple online sessions with all relevant stakeholders in the project, we developed a common understanding of our requirements and the matching frameworks.

The survey covered 34 frameworks, of which 11 were shortlisted for further discussion, after filtering for must-have requirements like package management or active development. Discussions with relevant stakeholders in the project showed the high priority of "social" criteria like widespread vendor support, large developer/supported base and clearly defined project governance. These three criteria left only ROS 2 as a viable candidate.

We verified that ROS 2 will meet our defined technical requirements, too. The only technical shortcoming of ROS2, not having built-in support for modern model-driven development methods or "advance modularity" concepts, can be mitigate using third party add-ons, like the RosTooling developed at project associate partner Fraunhofer IPA.

With the unanimous decision for ROS 2, this task has achieved its planned objective.

6 References

- Iborra, A., Caceres, D. A., Ortiz, F. J., Franco, J. P., Palma, P. S., & Alvarez, B. (2009). Design of service robots. *IEEE Robotics & Automation Magazine*, *16*(1), 24-33.
- Sanchez-Lopez, J. L., Pestana, J., De La Puente, P., & Campoy, P. (2016). A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. *Journal of Intelligent & Robotic Systems*, *84*, 779-797.
- Vahrenkamp, N., Wächter, M., Kröhnert, M., Welke, K., & Asfour, T. (2015). The robot software framework armarx. *it-Information Technology*, *57*(2), 99-111.
- Magnenat, S., Longchamp, V., & Mondada, F. (2007). ASEBA, an event-based middleware for distributed robot control. In *Workshops and tutorials CD IEEE/RSJ 2007 international conference on intelligent robots and systems* (No. CONF). IEEE Press.
- Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T. H., & Sifakis, J. (2011). Rigorous component-based system design using the BIP framework. *IEEE software*, *28*(3), 41-48.
- Kim, H., Cho, Y. J., & Oh, S. R. (2005, June). CAMUS: A middleware supporting context-aware services for network-based robots. In *IEEE Workshop on Advanced Robotics and its Social Impacts, 2005.* (pp. 237-242). IEEE.
- Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., & Das, H. (2001, March). The CLARAty architecture for robotic autonomy. In *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)* (Vol. 1, pp. 1-121). IEEE.
- Domínguez-Brito, A. C., Hernández-Sosa, D., Isern-Gonzalez, J., & Cabrera-Gámez, J. (2007). Coolbot: A component model and software infrastructure for robotics. *Software Engineering for Experimental Robotics*, 143-168.
- Arancón, M. M., Montano, G., Wirkus, M., Hoeflinger, K., Silveira, D., Tsiogkas, N., ... & Muhammad, A. (2017, June). ESROCOS: a robotic operating system for space and terrestrial applications. In *14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2017)* (pp. pp-1).
- Niemueller, T., Ferrein, A., Beck, D., & Lakemeyer, G. (2010). Design principles of the component-based robot software framework fawkes. In *Simulation, Modeling, and Programming for Autonomous Robots: Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings 2* (pp. 300-311). Springer Berlin Heidelberg.
- Fleury, S., Herrb, M., & Chatila, R. (1997, September). G/sup en/oM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS'97* (Vol. 2, pp. 842-849). IEEE.
- Côté, C., Létourneau, D., Michaud, F., Valin, J. M., Brosseau, Y., Raievsky, C., ... & Tran, V. (2004, September). Code reusability tools for programming mobile robots. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)* (Vol. 2, pp. 1820-1825). IEEE.
- SCHOLL, K., & DILLMANN, R. (2001). M SUZUKI. *Climbing and Walking Robots: From Biology to Industrial Applications (CLAWAR 2001)*, 443.
- Utz, H., Sablatnog, S., Enderle, S., & Kraetzschmar, G. (2002). Miro-middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, *18*(4), 493-497.
- Benjamin, M. R., Schmidt, H., Newman, P. M., & Leonard, J. J. (2010). Nested autonomy for unmanned marine vehicles with MOOS-IvP. *Journal of Field Robotics*, *27*(6), 834-875.

- Paunicka, J. L., Mendel, B. R., & Corman, D. E. (2001, June). The OCP-an open middleware solution for embedded systems. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)* (Vol. 5, pp. 3445-3450). IEEE.
- Calisi, D., Censi, A., Iocchi, L., & Nardi, D. (2008, September). OpenRDK: a modular framework for robotic software development. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1872-1877). IEEE.
- Han, S., Kim, M. S., & Park, H. S. (2012). Open software platform for robotic services. *IEEE Transactions on Automation Science and Engineering*, *9*(3), 467-481.
- Brooks, A., Kaupp, T., Makarenko, A., Williams, S., & Oreback, A. (2005, August). Towards component-based robotics. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 163-168). IEEE.
- Brooks, A., Kaupp, T., Makarenko, A., Williams, S., & Orebäck, A. (2007). Orca: A component model and repository. *Software engineering for experimental robotics*, 231-251.
- Mizukawa, M., Matsuka, H., Koyama, T., Inukai, T., Noda, A., Tezuka, H., ... & Otera, N. (2002, August). ORiN: open robot interface for the network-the standard and unified network interface for industrial robot applications. In *Proceedings of the 41st SICE Annual Conference. SICE 2002.* (Vol. 2, pp. 925-928). IEEE.
- Bruyninckx, H. (2001, May). Open robot control software: the OROCOS project. In *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)* (Vol. 3, pp. 2523-2528). IEEE.
- Bruyninckx, H., Soetens, P., & Koninckx, B. (2003, September). The real-time motion control core of the Orocos project. In *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)* (Vol. 2, pp. 2766-2771). IEEE.
- Saffiotti, A., & Broxvall, M. (2005, October). PEIS ecologies: Ambient intelligence meets autonomous robotics. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies* (pp. 277-281).
- Gerkey, B., Vaughan, R. T., & Howard, A. (2003, June). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics* (Vol. 1, pp. 317-323).
- Petters, S., Thomas, D., & Kiener, D. M. J. (2005). RoboFrame-Softwareframework für mobile autonome Robotersysteme. *TU Darmstadt Diplomarbeit*.
- Joyeux, S., & Albiez, J. (2011, May). Robot development: from components to systems. In *6th National Conference on Control Architectures of Robots* (pp. 15-p).
- Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).
- Thomas, D. (2014). ROS Documentation. *Open Source Robotics Foundation*.
- Wienke, J., & Wrede, S. (2011, December). A middleware for collaborative research in experimental robotics. In *2011 IEEE/SICE International Symposium on System Integration (SII)* (pp. 1183-1190). IEEE.
- Yoo, J., Kim, S., & Hong, S. (2006, October). The robot software communications architecture (RSCA): QoS-aware middleware for networked service robots. In *2006 SICE-ICASE International Joint Conference* (pp. 330-335). IEEE.
- Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., & Yoon, W. K. (2005, August). RT-middleware: distributed component middleware for RT (robot technology). In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3933-3938). IEEE.

- Schlegel, C., & Worz, R. (1999, October). The software framework SMARTSOFT for implementing sensorimotor systems. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)* (Vol. 3, pp. 1610-1616). IEEE.
- Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., & Tsagarakis, N. G. (2017, April). Xbotcore: A real-time cross-robot software platform. In *2017 First IEEE International Conference on Robotic Computing (IRC)* (pp. 77-80). IEEE.
- Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: yet another robot platform. *International Journal of Advanced Robotic Systems*, *3*(1), 8.

PART B – ADAVANCED MODULARITY

1 Executive Summary

Objective:
The need to develop modular robots for railway maintenance was outlined in the middleware selection document. This selection corresponds to subtask 18.1.1 of FP3 - IAM4RAIL WP18.
We remind you that the choice was made for ROS2.
The primary function of middleware is to orchestrate data exchanges between softwares operating within a system. This makes possible the breakdown of a complex algorithmic task into a set of elementary tasks. And the more elementary a task, the greater its potential for reuse in a variety of contexts. Middleware is therefore the essential component that guarantees software modularity. But the freedom offered by middleware in data exchange can be so extensive that it can run counter to industrial interests, for whom constraints on costs, development time and product quality are strong.
Objective of the work in task 18.1.2 “Overlay for Advanced Modularity” is to choose or to develop a set of tools on top of the common selected middleware to make the reuse (from robot to robot) of software components easy, fast, and reliable. The set of tools for robot’s design or support (maintenance) we’re looking for must therefore restrict the possibilities offered by middleware, without restricting the capacity of elementary software components or imposing an over-rigid set of rules on developers.

The purpose of this document is to define the main functionalities expected to guarantee simple, fast, and reliable reuse of software components. The technical means to be used to create these functions will be proposed. The impact of their use on developers needs to be assessed.
The aim is twofold: to show that the projected situation will remain acceptable to developers, and to predispose the developments undertaken as part of this workpackage to the advanced modularity tools that will be developed in parallel.

Methodology:
This document presents the results of cooperative work carried out either physically or online between the participants in this task within WP18.
The starting point was a more precise, but still high-level, formulation of the concepts we wanted to see developed under the term advanced modularity (chapter 6 of this document).

Subsequently, a corpus of basic software tools was elaborated. It is through the application of a tool or combination of tools that concepts can be materialized. Relationships "Concepts/Tools” are shown in Appendix C.
We also set out to describe:

1. the nature of the activities we intend to carry out for each tool in the project;
2. the benefits of these tools for the various stakeholders involved: manufacturers, integrators, end users.

Conclusion:
From the general “advanced modularity” concept, we have established a list of sub-concepts

detailing our expectation (chapter 6). Then those sub-concepts have been transformed into a first set of 11 software tools (chapter 7). Most of the tools we propose to develop will benefit from work previously carried out by Fraunhofer IPA. This minimizes our development effort while maximizing our chances of success.

Our development efforts will focus on 8 tools (data structure for both component and system, related data base, component properties extractor and visualizer, a tool to model an assembly of components (robot), a tool that automatically generates code and finally a tool for checking incompatibilities and missing information in a system model).
For 2 tools (automatic documentation generation, code quality check), we will test existing solutions and make recommendations for use and/or further development.
For one last tool (tool that creates the robot's software distribution from its model), we will draw up recommendations for future developments.

The corpus will be a first set of Model Design Engineering tools with a strong impact on product quality and development productivity. Beyond the scope of the project, efforts will have to be continued to expand this set (e.g. moving the system's data structure from the "logical" scale to the "physical" scale, then to the mechanical scale) or by undertaking work on tools that are not yet covered (software distribution tool).

The tools that generate constraints to developers have been identified. We have endeavoured to limit this number. The precise list of constraints will be known in January 2024, following the completion of a study conducted at the University of Stuttgart with the help of students.

2 Abbreviations and acronyms

Abbreviation / Acronym	Description
API	An Application Programming Interface is a software interface allowing software to communicate with each other.
ARM	Advanced RISC Machines (and originally Acorn RISC Machine) is a family of RISC instruction set architectures for computer processors. A reduced instruction set computer (RISC) is a computer architecture designed to simplify the individual instructions given to the computer to accomplish tasks
CPU	A central processing unit (CPU) is the most important processor in a given computer. Its electronic circuitry executes instructions of a computer program, such as arithmetic, logic, controlling, and input/output (I/O) operations.
DSL	A domain-specific language (DSL) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains.
Fiware	Middleware - similar to ROS and ROS2
GPU	A graphics processing unit (GPU) is a specialized electornic circuit deisgned to accelerate image processing and computer graphics. It's more and more used for non-graphical parallel calculations.
Ignition	Ignition is a simulator for robotics.
IOT	Internet of Things
LTS	Long-term support (LTS) is a product lifecycle management policy in which a stable release of computer software is maintained for a longer period of time than the standard edition.
MBSE	MBSE is a technical approach to systems engineering that focuses on creating and exploiting models.
MDE	Model Driven Engineering
Node	A *node* is a process of a package that performs computation.
OS	An operating system (OS) is system software that manages computer hardware and software resources, and provides common services for computer programs.
OSRF	OSRF is a Non-profit Public Benefit Corporation. Its mission is to support the development, distribution, and adoption of open source software for use in robotics research, education, and product development.
Package	The ROS packages are the most basic unit of the ROS software. It contains the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit. Packages are the atomic build item and release item in the ROS software.
RGB	The RGB model is a very usual model for colours decomposition. It assumes that each colour is a combination of 3 colour channels

	(Red, Green, Blue).
ROS	Robot Operating System A collection of tools, software libraries and common practices to build sophisticated robot control software from reusable building blocks and a powerful communication layer. The de-facto standard in academia for robot software development.
ROS2	Version 2 of ROS, addressing most of ROS' architectural and cyber-security shortcomings.
RPA	The Robot Process Automation is a branch of Artificial Intelligence dedicated to data manipulation (automatic filling of forms, database…)
RQT	RQT is a software framework of ROS that implements the various GUI tools in the form of plugins.
RVIZ2	3D visualization tool for ROS2
Service	Services are named buses over which nodes exchange synchronised messages (one to one communication)
Topic	Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption (one to many communication). In general, nodes are not aware of who they are communicating with.
URDF	The Universal Robot Data Format is an XML type language used in ROS/ROS2 to describe the robot mostly from the kinematical point of view.
V-cycle	The V-cycle or V-model is a graphical representation of a systems development lifecycle. The left side of the "V" represents the decomposition of requirements, and the creation of system specifications. The right side of the "V" represents an integration of parts and their validation.
VS-CODE	Microsoft's cross-platform development tool. This is probably the main text editor used by developers, at least in the ROS community.
X86	x86 is a family of complex instruction set computer (CISC) instruction set architectures.
XACRO	XACRO is an XML type language used in ROS/ROS2 to introduce flexibility (through parameters, conditional instructions…) in the URDF models.
XML	Extensible Markup Language is a markup language and data format for storing arbitrary data. It defines a set of rules for encoding information in a format that is both readable by human and machine.
Xtext	Open-source software framework for developing programming-languages and domain-specific-languages.
YAML	YAML is a human-readable data serialization language, commonly used for configuration data. It targets many of the same applications as XML but has a minimal syntax.

3 Objective/Aim

The need to develop modular robots for railway maintenance was outlined in the middleware selection document. This selection corresponds to subtask 18.1.1 of FP3 - IAM4RAIL WP18.
We remind you that the choice was made for ROS2.
The primary function of middleware is to orchestrate data exchanges between software operating within a system. This makes possible the breakdown of a complex algorithmic task into a set of elementary tasks. And the more elementary a task, the greater its potential for reuse in a variety of contexts. Middleware is therefore the essential component that guarantees software modularity. But the freedom offered by middleware in data exchange can be so extensive that it can run counter to industrial interests, for whom constraints on costs, development time and product quality are strong.
Objective of the work in task 18.1.2 “Overlay for Advanced Modularity” is to choose or to develop a set of tools on top of the common selected middleware to make the reuse (from robot to robot) of software components easy, fast, and reliable. The set of tools we’re looking for must therefore restrict the possibilities offered by middleware, without restricting the capacity of elementary software components or imposing an over-rigid set of rules on developers.
The use of the term middleware overlay is somewhat abusive. In robotics, the data communication layer (pure middleware) is not provided on its own. A set of elements ranging from development tools to a corpus of first elementary components accompanies this core function. The advanced modularity tools we’re talking about are not directly linked to core functionality. They complete the range of development tools. As a result, few of those elements will be active directly on the robots. They will be used in the design or support (maintenance) phases.
We’ve used the term advanced modularity to highlight the benefits of these tools. In the scientific and technical community, they are grouped under the term “Model Driven Engineering” (MDE). This term highlights the means rather than the end. Please note, however, that while we'd like to adapt some of the tools in the MDE approach, we don't want to impose the entire approach on developers.
The purpose of this document is to define the main functionalities expected to guarantee simple, fast, and reliable reuse of software components. The technical means to be used to create these functions will be proposed, in order to assess the impact, they will have on developers.
The aim is twofold: to show that the projected situation will remain acceptable to developers, and to predispose the developments undertaken as part of this workpackage to the advanced modularity tools that will be developed in parallel.

4 Methodology

While the adjective "advanced" that we have attached to the noun "modularity" does indeed express an intention, it is, like other terms in widespread use today (smart, green, etc.) very imprecise. Through online group sessions in the spring of 2023, we clarified the concept we attached to the term advanced modularity. The overall concept has been broken down into sub-concepts.

This was the starting point for a second phase in a core group, during which the technical elements to support the concepts were established. We relied as much as possible on elements already developed by Fraunhofer IPA for its own needs or as part of previous European or national projects.

The residual effort required to develop the elements has been evaluated. It has been compared with a level of desirability in order to retain what it will be concretely possible to develop within the framework of FP3 - IAM4RAIL WP18.

Finally, the constraints on developers were explained. On the one hand, it was a question of validating that our choices did not result in a corpus of rules that was too rigid or too voluminous. On the other hand, it was a question of informing WP18 developers so that they could predispose their software.

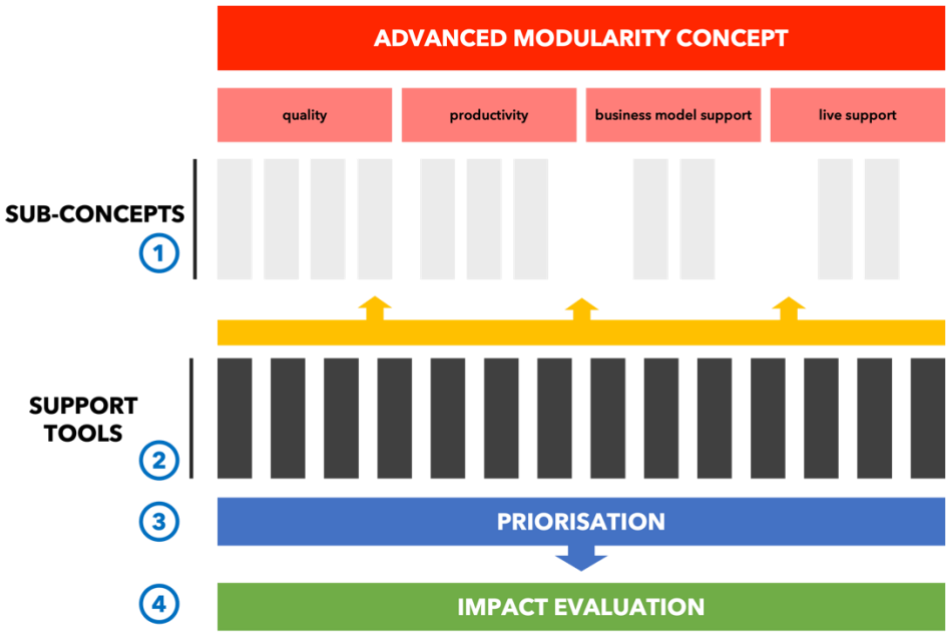

Figure 3 - Principal working steps

The remainder of this document will describe these 4 stages.

5 Clarification of the concept of advanced modularity

Before going into detail on the various sub-concepts or characteristics that define the notion of advanced modularity, we need to specify the level at which we intend to manage modularity. Modularity can be managed at several levels: that of the system (a robot) or that of a system of systems (a fleet of robots, a robot in an environment of connected objects...). We choose to deal with the problem at the system level. The aim is to reduce the level of complexity and the resources required to achieve tangible results.
All the ideas we are going to develop will both increase the quality of the resulting product and reduce development time. But all have a particular tropism for the first or the second criterion.
The partners in this workpackage are mainly end-users. In addition, given the high safety requirements of the rail sector, particular attention will be paid to what can be done to raise quality.
To these 2 types (quality and pure productivity) we need to add 2 other types, which is support for the business model and live support.
In the following we list several aspects of this Advanced Modularity and the tool support we envision for it. It must be noted, that when we talk about “the advanced modularity tool” or similar in the following, we do not mean one single tool delivering all of the aspect listed in 6.1 to 6.4. One highly integrated tool would be too complex to manage. Instead, we envision having multiple, loosely coupled tools that contribute to one or two of the requirements each and that can be combined as needed.

5.1 Quality

5.1.1 from "we can talk" to "we understand each other"

In order to describe our vision of this subconcept, it seems important to us to evoke our experience of the use of robots' components, and in particular of software components. First of all, we have to say that our practice is based on the ROS or ROS2 middleware. As already mentioned, the main reason for the existence of those middleware is precisely modularity. If we address only the strict technical point of view, all ROS/ROS2 components can be reused and connected to other components (if that makes sense). This extreme freedom offered by ROS/ROS2 is not, however, what we would like to see classified under the term advanced modularity.
Indeed, behind this infinite potential are hidden real operational difficulties. The key to the vast library of ROS/ROS2 components (we are talking here about components directly managed by OSRF but also all components developed by third parties) is a block diagram. The inputs, outputs (with specific types for both) and main function are known, while the details of the implementation can be hidden.
This information about interfaces and general function is obviously necessary, but it is far from being sufficient in an industrial context. What is the validity domain of the function? For what range of variation, quality, frequency of update of the input data? What level of accuracy do we have on the output data?
This is for us the fundamental notion of the concept of advanced modularity: components carrying elementary functionalities must be able to be reused in large contexts but the "contracts" binding

these components must be clarified. The components should not just be able to talk to each other. We have to make sure they understand each other.

5.1.2 To an automated matching process

Paper documentation of requirements and capabilities would be one way to meet the need expressed in the previous paragraph. Given the volume of criteria to be studied, the risk of error during the examination process will remain high. This pleads for an automation of this matching process, as well as the frequency of the reviews. We know that software is generally subject to frequent updates.

5.1.3 Configuration management

Configuration management is an essential part of the safety management. It is not enough to say that this or that software brick is compatible with this or that other brick. This must be managed at the version level. This can also be managed off-line or live with software directly on the robots. If the offline situation needs to be managed quickly, the second case, which corresponds to installed configuration management, seems to be a second, deferred stage.

5.1.4 Complexity management

The advanced modularity tool must help us to detect the software components that are too complex:

- no clear separation between elementary functions
- poorly formulated code
- presence of memory leaks
- possible endless loop
- bad memory management (unnecessary copies of variable in memory…)
- too many nested loop
- poor documentation of the code
- too many variables
- …

5.2 Pure productivity

5.2.1 To automated links

In the follow-up of the automatic matching (see 6.1.2), an automated code generation to link the nodes and launch the resulting combination will be greatly appreciated because this activity does not really generate added value. On the other hand, the work that this represents can be, if it is done manually, a source of many errors and demotivation for the developers.

5.2.2 Automated code generation

Automated code generation may concern other fields than the links between the packages. For example runtime monitoring code or automated failure recovery, e.g., restarting failed nodes or launching additional ones for load balancing.

5.2.3 More visual tools

ROS2 is an extraordinary tool. However, it is sorely lacking in visual tools. Reducing the level of expertise required in the assembly process by using graphical tools opens up the ecosystem to the benefit of both technology providers and end users.
These graphic tools can be used in a wide range of design phases. Here are the priority fields (not covered by the tools in ROS2 or not covered satisfactorily) that have been identified:

- design of software components (nodes as well as packages);
- managing of configuration parameters;
- design of the software system architecture and data flow;
- visualise the parameters of the components;
- visual representation of deployment structure, i.e., which software part runs and where.

5.2.4 Modularity for safety demonstration

In addition to the acceleration and reduction of development costs, there is another element that can shed light on the notion of advanced modularity. We want to move towards modular robotics. To take full advantage of the approach, modularity must also be applicable to the field of safety demonstration. If a single technical component of a robot has to be replaced, and such replacement requires to perform the safety demonstration from scratch, this will have a major impact on the time and cost of bringing the product to market. This will also reduce the market's openness to multiple players.
The concept of contract mentioned above must include the meta-data related to probate. A tool that would manage the level of safety demonstration resulting from the assembly of several "pre-approved" components would be an undoubted plus. By "pre-approved" component, we expect a component for which approval work has been carried out at the elementary level (at the bottom of the V-cycle if we refer to this design process).
NOTE: the availability of graphical tools (see 5.2.3) for modelling the robotic system can also create synergies with MBSE-type safety approaches.

5.2.5 Compatibility mapping

The next item is quite close to the previous one. The tool must be able to indicate the compatibility between 2 components designated by a user. The same basic functionality could be used to establish a global mapping of the compatibility level between all the components of the platform. This can allow the creation of associations not initially envisaged or to enrich areas where too many dependencies exist.

5.2.6 Separation of concern

Our tool must avoid too much discussion between the different roles. A component supplier must be able to offer a generic product, he must not adapt it in a very precise way to the need of a single other component.

5.3 Support for the business model

5.3.1 Remuneration mechanism

For the time being, we approach our ecosystem from a rather technical point of view, but it will only be viable in the long term if it is economically viable. Some components of the platform will have to be paid for. The contract mechanism mentioned earlier can probably be used to develop and monitor the developer remuneration mechanism.

5.3.2 Propagation of licenses

In the same way, it can probably be useful to better evaluate the propagation of licenses (and especially open-source licenses).

5.4 Live Support

5.4.1 Online monitoring

The tool we are talking about must allow us to manage contracts between the components of our system. It can happen in the life of the system that we enter into degraded modes that take us away from the approval conditions. We can either try to do everything so that the system never moves away from the nominal. This can lead to a drastic increase in the acquisition cost of the system or its maintenance cost. If we can make the system self-diagnose its state and signal the moments when it goes out of its certification domain, this can open the door to other management modes. So far we have talked about contracts in the context of design. We could therefore benefit from a tool whose contract management module could also work "live".

6 Technical elements required for our advanced modularity

In the previous section, we developed our definition of "advanced modularity". We now turn to the technical elements on which we propose to base our concept. In the Appendix C, we present a transposition matrix between the sub-concepts of the chapter 6 and the technical elements of this chapter. It illustrates the contribution of the different technical elements to the different “advanced modularity” sub-concept.

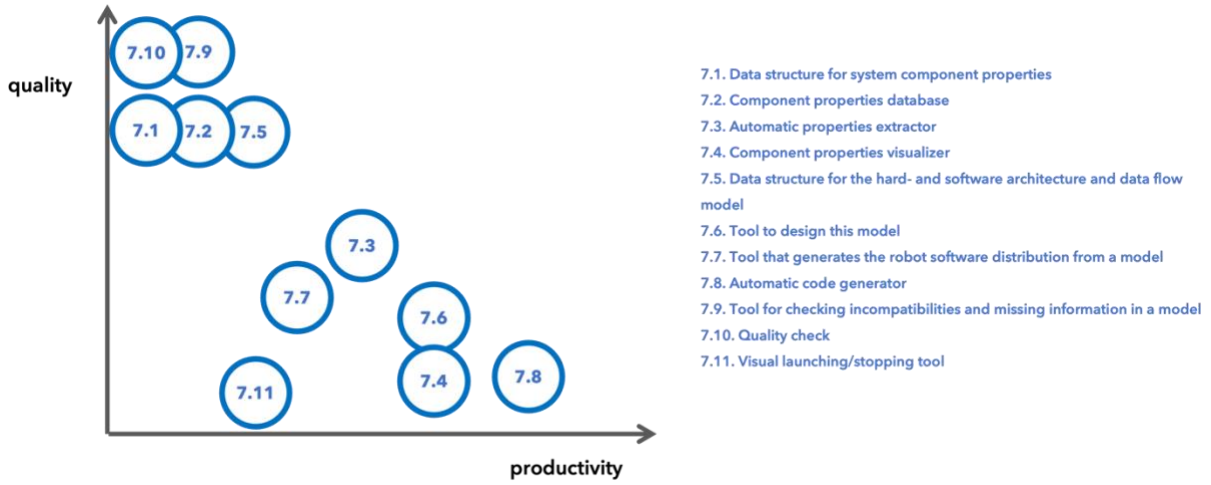

Figure 4 - Tools and their effect

The Figure 2 above positions the tools in relation to 2 (quality and productivity) of the 4 dimensions mentioned in the previous chapter. We have made this choice to simplify the graphic representation, but effects are expected on the other 2 dimensions (live support and business model support). Before going into detail about the tools, we need to identify some of the stakeholders in the ecosystem and the role they will play. The use to which the tools might be put can indeed vary according to these roles:

1. End users: entities carrying out maintenance on behalf of or within railway companies (infrastructure managers or railway operators)
2. Integrators: the name integrator is traditionally used in the manufacturing industry. That's why we use it, even though it only partially reflects reality in the maintenance sector. In our sector, integrators are more likely to be assemblers, using components of various origins to form robots. End-users are more likely to be responsible for integrating robots into their maintenance production processes.[6]
3. Manufacturers: this term designates the companies (robotics technology suppliers) that develop the elementary components - hardware or software - that constitute robots.

[6] Some railways may wish to set up entities with the capacity to take on this role.

6.1 Data structure for system component properties

In the context of modular software, it is important to design an effective data structure for system component properties. It cannot simply be a matter of putting the main characteristics (also known as metadata) of the component one after the other. This would make modelling and data processing tedious and cumbersome. This data structure must therefore make it possible to efficiently link the outputs of one component with the inputs of the next, with a view to matching phase.

The data structure should be capable of accommodating values of different data types, including integers, strings, booleans, arrays, or objects. This flexibility makes it easier to work with and interpret the properties.

These properties will be extremely varied in nature. It may concern the type of measurement provide by sensors (image, point cloud, georeferenced coordinates, velocity vector, etc.), or details of this type (e.g. for an image: RGB, coded on 8 bits per colour channel, width, height, flux, reference frame, optical defects corrected or not...), validity ranges of input or output elements (e.g. min width, max width), requirements other than those based on characteristics (e.g. requires a specific brand or model of Lidar).

In the industry we can find many efforts to get a standard. One important example has been done by the Fiware[7] community. Fiware is an open-source platform that provides a set of APIs (Application Programming Interfaces), standards, and data models for the development of smart applications in various domains such as Smart Cities, Smart Industry, Smart Agrifood, and more. It offers a framework for the development of smart solutions by enabling easy integration of different components and services. In term of data structure, Fiware provides the smart data models[8] that are standardized data structures that represent commonly used concepts and entities in the context of the Internet of Things (IoT) and smart applications. There are examples of smart data models of almost every IoT device we can imagine.

We intent to reuse a data structure already used by Fraunhofer IPA on other internal, national or European projects. It is based on a Xtext DSL already described in the literature[9] [ROSMetamodeling]. This will be used to describe the Meta-Models. It will have tool support to ensure that any concrete component instance model conforms to the meta-model. The component model will include different types of properties, such as properties concerned with interconnection, properties concerned with domain of operation, properties concerned with configurability and more. The meta-model respective DSL will be aimed at both human and machine readability and geared towards support for automated model validation.

For practical reasons, we will likely define individual models per type of properties and a structured way to link or reference this individual models. Such a set of individual models better facilitates the desired separation of concerns between different stakeholders or ecosystem roles.

This structure will be use within the project to describe the developed components. This large-scale test will highlight any weak points or additions to be made. The great flexibility of the technical solution will enable us to manage a wide variety of situations, even if not all have been

[7] https://www.fiware.org/

[8] https://www.fiware.org/smart-data-models/

[9] https://ieeexplore.ieee.org/document/8675668 Bootstrapping MDE Development from ROS Manual Code - Part 1: Metamodeling

anticipated at this stage.
This is a point which mainly concerns manufacturers. We have to be extremely careful to export as few constraints as possible to manufacturers, if we want to benefit from the widest possible range of technologies. This is all the truer given the bigger variable size of the players involved. This data structure in itself will not export any constraint to manufacturer.

6.2 Component properties database

The structure defined in the previous section must be fitted in a database to monitor and control the system. When selecting an appropriate database, several constraints need to be considered: First, different component properties have distinctive life cycles. We have:

- Static properties defined by the component manufacturer. These are properties describing invariant characteristics of the component, that do not depend on the application context
- Semi-static component properties defined by the component manufacturer and set by the integrator.
  These are properties that are fixed at system design time by the system integrator and do not change at deployment or runtime.
- Dynamic properties defined and set by the integrator. These are properties that are defined at system design or deployment time and that change value during normal operation.

Note that we only deal with component properties here that are needed to design and maintain the robot system. Handling application runtime data, such as image data stream or other sensor time-series data is out of scope of this discussion and must be addressed at component level itself, i.e. when designing a data acquisition or data storing hardware/software component.

We will therefore concentrate our efforts on finding the most relevant database for static and semi-static properties. As far as dynamic properties or sensor data are concerned, we can at best formulate recommendations for further developments within the framework of this WP.

6.3 Automatic property-extractor

Some of the properties that need to be managed in the structure mentioned in point 6.1 can be contained in the source code itself. The collection of database information is a tedious, low-value-added task (task that is in the manufacturers' scope of action). But it has a strong impact on the quality of the entire system, which is under the responsibility of the integrator, and which is a major concern for the end users. The integrators should be the main users of this property-extractor.
So, as soon as a piece of information is present in the code, we must make the best effort we can to collect it and avoid requiring an operator to enter it manually. Only in highly processed and monitored organizations is it possible to maintain a high level of quality on manual databases. Our open ecosystem, involving contributors of various natures and sizes, cannot claim to follow this

model.
The effort required to collect data automatically will obviously have to be weighed up against the benefits: we cannot say that we have to be able to collect information automatically at all costs; nevertheless, automatic data collection will probably be a key element in maintaining data sets at a high level of quality.
We support the open-source model. This may not be the case for certain manufacturers in the ecosystem. We cannot yet impose this model, which could cut us off from interesting technical solutions. This tool must therefore adapt to the configuration of a player wishing to open its code, as well as to that of a player locking its source code. The tool should not expose the core of the algorithms. In both cases, the instantiated properties will be identical.
We do not wish to be able to support the components of the manufacturers who would like to make their solution(s) compatible with our environment, for example through a set of APIs, but without complying with native ROS2 rules. They will be responsible for filling in the data manually or with their own tools.
This tool should work with OS associated with ROS2 versions and for CPU platforms managed in our ecosystem (currently Ubuntu 22.04 LTS for X86 and ARM architectures). Minimum hardware requirements will be identical to those for OS.
This tool is approaching the RPA field. As the original elements are lines of code in structured languages, potentially responding to guides, not requiring a priori cognitive interpretation, no major lock exists in this area. A first version of such a tool, developed by Fraunhofer IPA, already exist in the ECLIPSE Framework.[10] [ROSModelextraction].
The ECLIPSE framework is a vast project. It goes beyond the strict requirements identified for the MDE. Fraunhofer IPA has undertaken a migration of its tools to a VS-CODE plugin. The main challenges of this migration are finding replacements of some of the Eclipse Modelling framework core libraries (which are Java libraries) in the VS-Code ecosystem. Another challenge is the visual representation of component models and especially an interactive visual canvas. Promising candidates to overcome both challenges exist but need further investigation.
Here again, we want to test the tool proposed by Fraunhofer IPA after migration to VS-CODE to consolidate a list of required and possible evolutions.
This tool will export constraints to the manufacturers' developers. A study has just been launched with the help of students from the University of Stuttgart. The aim is to list these induced constraints in greater detail.

6.4 Component properties visualizer

The ROS2 environment currently offers few graphical tools provided by the OSRF:

- RVIZ2 visualizes measured data (images, point clouds, accelerations, etc.);
- RQT visualizes more structural information: network of active nodes and data flows, tree structure of robot frames, list of transmitted topics, etc.

[10] https://ieeexplore.ieee.org/document/8906937 Bootstrapping MDE Development from ROS Manual Code - Part 2: Model Generation

- IGNITION (client part) is the graphical interface of a rigid multi-body dynamic simulation tool.

Development under ROS2 relies on direct generation by developers of lines of code (XML variants, Python, C/C++ or, less frequently, other languages) or parameter files (mostly YAML).
A very common practice in the developer community is the use of VS-CODE. Although VS-CODE is a real productivity tool, it is still a text editor. It offers no advanced visualization functionality out of the box. However, several plugins exist to add data visualization. For example, Fraunhofer IPA has developed a plugin to visualize and manipulate kinematic models of robotics systems described by URDF / XACROS files.
This sparing presence of graphical tools can be seen as a virtue. For ROS2 developers and maintainers, it reduces the workload. For users, no effort is required to grasp the graphical interface. But it can be tedious to find the information, buried in dozens or hundreds of lines of code, spread across several files and sometimes managed in several files with notions of priority (the same information may be entered with different values, for example in a parameter file and in a launch file, but the value in the launch file takes precedence over the other). In the end, it is the developer's productivity, or the quality of the code executed that suffers.
In our context, there is therefore a strong need for a graphical interface that is easy to maintain, easy for users to understand and that accurately reflects the structuring of component metadata. Visualization is a first objective, but it would also be advisable for this tool to be able to enter and correct information, ensuring a consistent double backup in the database mentioned in point 6.2, as well as in the source codes for the information contained therein.
Based on previous work, Fraunhofer IPA already offers this tool in the ECLPISE framework for visualization and modification. It is part of the “ROS Tooling”. We already mention the wish to have the MDE tools aside the ECLPISE framework. The migration is VS-CODE is hard to manage. The targeted framework is the Theia framework[11].
This visualizer will be used by both manufacturers and integrators.

6.5 Data structure for the hard- and software architecture and data flow model

In a previous paragraph, we mentioned that the RQT tool, supplied as part of the standard ROS2 distribution, can be used to visualize a robot's active nodes and data flow (topics or services).
This visualization is performed "live" and only concerns active nodes and topics (not services) without any monitoring on metadata.
This tool can only be used fairly late in the development phase, as the nodes need to be executed. Even when this tool is monitoring metadata, the discovery of an incompatibility at this late stage can have a huge impact: a significant proportion of the work carried out may turn out to be totally useless.
For two fundamental reasons (lack of metadata and the need to execute nodes), we can't rely on this tool.
We therefore wish to rely on the characteristics of each node documented thanks to the structure and database described in 6.1 and 6.2 respectively.

[11] https://theia-ide.org/

However, integrators need to be able to indicate how nodes and properties are organized.
Systems and nodes properties can be defined at different levels:

- Logic level: definition of inputs and outputs, comprehensive of data type, allowed values, minimum/maximum data input frequency required.
- Deployment level: once an application, as a combination of blocks, is defined, it must be deployed. The deployment involves the definition of the computing architecture, meaning that components must be assigned to computers in which they will be deployed. Incompatibilities may arise if a certain node does not support a CPU architecture, or if the combination of more nodes together in the same computer creates a high workload on the CPU. For this reason, each component should come with a definition of the supported CPU architectures, typical CPU usage and power consumption, and the peripherals interfaces needed.
  This also means that computing platform should be coherently implemented into the advanced modularity tools, including their relevant specifications – CPU, GPU, RAM, and other relevant data.
- Mechatronics level: depending on the level of the component used – it may be a simple node or a complete robot – integration may require hardware connections, both mechanical and electrical. The definition of these properties may be quite hard, especially on the mechanical side, and may be explored in a subsequent phase of the project. Definition of communication interfaces – (Ethernet, USB) – and power supply requirements should be implemented per each component, to define if connections between components are allowed.

This information must be stored in a database. It can be a shared database, such as the one defined in paragraph 6.2 or a local database, sauch as a single file. While it is vital for the ecosystem that the static properties of individual components are exchangeable, therefore accessible in a shared database, this is less the case at the level of the complete system.

With the resources available in WP18 and based on previous developments at Fraunhofer IPA, we can handle at least the logic level. For the more advanced levels, we'll try to establish recommendations for further activities.

Here are some examples from the market. They may be a source of inspiration, but not all will be usable as they stand; some elements, for example, are based on proprietary formats.
20-sim[12] is another example of a graphical tool that enable the multi-domain modelling from the mechatronic perspective. The interface admits the equation-based modelling as well. This software also has simulation capabilities and code generation for real-time applications. However, as its counterparts (Labview and Simulink) are proprietary.
But we can also find other open-source examples, such as OpenModelica[13] and Scilab[14]. The purpose of OpenModelica is mainly for modelling and simulation, while Scilab is more focused on

[12] https://www.20sim.com/
[13] https://openmodelica.org/
[14] https://www.scilab.org/

numerical computation and data analysis. OpenModelica also provides the Modelica modelling language[15] which is an open standard for modelling complex physical systems, enabling users to represent both the structure and behaviour of systems.
Eclipse Modelling Framework (EMF)[16] is a powerful framework for building tools and other applications based on a structured data model. It is an open-source framework that forms part of the Eclipse platform. I ECORE is a metamodeling language, allowing the implementation of the OMG eMOF (essential Meta Object Facility) specification.

Therefore, manufacturers are asked to provide systems and components that are bundled with the information described, in order to allow easy integration of their products into the advanced modularity tool and enforce the MDE paradigm.

This tool will export constraints to the integrators' developers.

6.6 Tool to design this model

As already mentioned, in ROS2 it is the launch file that organizes the launch of nodes and services, matching published and subscribed topics. Although a model can be observed during runtime, there is no a priori structuring of the model.
Let's also reiterate that development under ROS2 relies on-line-of-code development in standard languages with very few tools. The "entry ticket" for developers is therefore very low, more so than with graphical tools. On the other hand, acquiring a high level of mastery, and therefore a perfect understanding of induced code behaviour, is far more costly.
In the context of production use, with possible safety consideration, the emphasis must be on product control. Integrators therefore need to have tools for this model structuring process.
We are confident in our ability to offer this type of tool, as the Eclipse framework already offers them. As for the other tools the migration from ECLIPSE to VS-CODE plugin is going on. This migration is made possible by the Theia framework (the same framework as for Tool 6.4).
It remains to be seen whether these tools can be adopted as they are or if modifications are required. It will therefore be extensively tested in the WP's various developments.
This tool will be aimed primarily at integrators.

6.7 Tool that generates the robot software distribution from a model

Our modular robots will result from the assembly of components. The number of components, especially software components, will increase with the complexity of the task to be accomplished. For a given component, there might be a relatively high number of variants: it may be necessary to generate a version for each type of CPU architecture (X86 and ARM), depending on the distribution of ROS2, the versions of libraries on which the software depends... Versions must coexist, as it is not always appropriate to undertake a migration of all deployed systems.
Nevertheless, it is vital that the software versions deployed on a system are indeed those expected. The manual extraction of executable binaries or containerized elements is a potential

[15] https://en.wikipedia.org/wiki/Modelica
[16] https://projects.eclipse.org/projects/modeling.emf.emf

source of errors. Helping integrators with this task can therefore contribute to the quality of the final product.
However, to be able to provide a tool for this stage, we need a physical model of the system (at least for the communication layer), which we said in 6.5 was not a priority for this first ERJU call.)
This point will therefore be the subject of recommendations for future work.

6.8 Automatic code generator

The goal of automatic code generation is a correct-by-construction final result. The key idea is to leverage formal information about the software system under construction to create as much of its infrastructure code (so-called “boiler-plate code”) as possible to avoid human error in these parts. The human developer is then only tasked to implement the internal application logic of individual components, while all the integration of components is done by the code generation tool.
Generated code in the context of ROS2 can include launch files, parameter and configuration files, and the code skeleton for components as well as package-level infrastructure and deployment infrastructure such as docker files and docker-compose files.
An early example is the BRIDE (Model-to-text) tool from the OROCOS framework. Another example is the various code generator in the SmartSoft toolchain. Both focus on C/C++ and have no or only limited support for Python code and do not support ROS.
During software development, a second aspect is generating documentation. This should have a more important place, which it does not always have currently. A good documentation is hard to develop, maintain and very tedious as well. An automatic documentation of the code is a very important topic to make consistent the relationship between the documentation and the code. There are many tools that normally are tight to the chosen programming language.
Here at least three types of documentation should be distinguished:

1. Code documentation aimed at developers enhancing a components internal workings.
2. Code API documentation explaining how to use a given component in a larger software system
3. Deployment documentation describing the actual system configuration of a specific deployed system, including the full bill of material (e.g. which components in which versions are used) as well as a detailed record of which parts of the software system are running on which parts of the hardware and how are those parts physically interconnected.

Documentation of the first two types is derived from specifically formatted comments inside the source code. While several tools exist to extract such special comments and present the contained information in a variety of way, such as HTML pages, PDF text and graphical class relationship diagrams, only very few support the developer in writing such special comments in the required form. A notable exception is the “autoDocstring” extension[17] for Python in VS-Code.
The most popular tools to generate documentation are the following:

[17] https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring

- Doxygen[18] is a widely used documentation generator that supports various programming languages, including C++, C, Java, Python, and more. It can generate an on-line documentation browser and/or an offline reference manual from annotated source code.
- PyDoc[19]: in the case of Python, Doxygen does not work as good as it does for other programming languages as C/C++. Thus, in this case, a specific tool for Python is preferred.
- Sphinx[20] is a documentation generation tool widely used in the Python community. It can be used to document various types of projects, including software projects, web applications, and more. It supports multiple output formats, such as HTML, PDF, and ePub.
- JSDoc[21] is a tool that generates documentation from JavaScript code. It uses special comments to generate API documentation in HTML format. It is commonly used for documenting JavaScript libraries and frameworks.
- GitBook[22] is a modern documentation platform where teams can document everything from products to internal knowledge bases. It supports the automatic generation of documentation from Markdown files and integrates with various version control systems. Read the Docs[23] is a popular documentation hosting platform that can automatically generate documentation from your codebase. It supports various documentation formats, including Sphinx documentation for Python projects, MkDocs for project documentation, and more.

As part of the work-package, we will not be undertaking any development activities on the documentation part. As we have just seen, the "market" offers tools. We can't argue that these tools have limitations that would justify our own developments. We will take advantage of WP18's developments to gain a better understanding of any limitations and/or establish recommendations for use for manufacturers and integrators.

For the part concerning what we have called "boiler-plate code" we will continue the initial developments undertaken by Fraunhofer. This will benefit manufacturers and integrators alike.

This class of tools will export constraints to the manufacturers and integrators' developers.

6.9 Tool for checking incompatibilities and missing information in a model

During the creation of the robot model, or as a final check, you need to verify that the characteristics of the output elements of one component correspond to the characteristics of the input elements of the next component (see next figure). As it can be seen, when there is a match between the variable type of the output port of the previous component and the input port of the next component, the tool granted the link. In other case, the link is rejected. It is also important to take into account the maturity of the model. In the early stages of development, you may not

[18] https://www.doxygen.nl/
[19] https://docs.python.org/3/library/pydoc.html
[20] https://www.sphinx-doc.org/en/master/
[21] https://jsdoc.app/
[22] https://www.gitbook.com/
[23] https://about.readthedocs.com/?ref=readthedocs.com

have all the information you need to validate a link. The mechanism that prohibits the tracing of a link that has not been fully validated, as illustrated in our diagram, should therefore not be automatic.

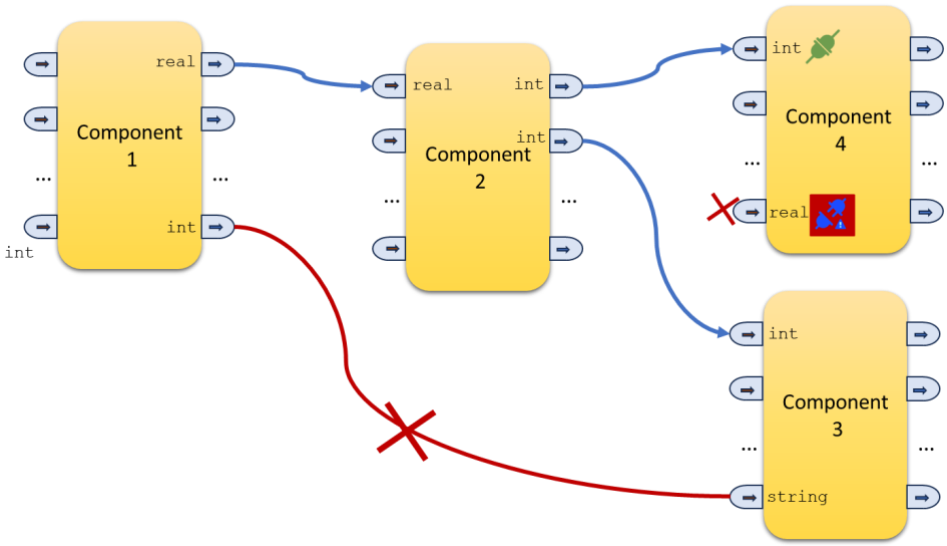

Figure 5 - Example of the input-output check between modules

For shake of clarity, the previous diagram only shows the case of variable type check. However, more examples can be found of an extended check where other types of constraints can be considered. Such extra constraints can be signal frequency update or analogue variable signal limits.
Some information can be expressed in simple terms, but comparisons cannot always be made directly. Sometimes an equation is needed.
Another check should be done on ports that must be connected to other ports, compare to the optional or loosely coupled ports, where can leave them unconnected (see component 4 in the previous figure). Structure101 is a software that not only let visualize underlying structures of the code but also specify APIs for every module and check the incompatibilities among them. It supports Java, C#, C/C++ and python.

This tool will export constraints to the manufacturers and integrators' developers.

6.10Quality check

Checking the quality of source code is an essential aspect of software development. There are various tools available that can help analysing and assessing the quality of your source code. Here are some popular tools used for this purpose:

- Linters: Linters analyse source code to detect potential errors, coding style issues, and suspicious constructs. Popular examples include ESLint[24] for JavaScript, Pylint[25] for Python, etc. In [26] a comprehensive list for linters can be found.
- Static Code Analysis Tools: These tools perform a deeper analysis of source code, looking for bugs, security vulnerabilities, and code smells. Examples include SonarQube, PMD, FindBugs, etc. In [27] a comprehensive list for static code analysers can be found.
- Code Review Tools: These tools facilitate code reviews and collaboration among team members, allowing them to comment on code changes, suggest improvements, and ensure code quality. Examples include GitLab [28] and Bitbucket[29]. In[30], a list of most popular code review tools is presented.
- Code Coverage Tools: Code coverage tools help assess how much of your source code is covered by your test suite. They help you ensure that your tests are comprehensive and thorough. Popular examples include JaCoCo for Java, Istanbul for JavaScript, and pytest-cov for Python. Please check [31] for more details.
- Complexity Analysis Tools: These tools help assess the complexity of the codebase, identifying areas that might be hard to understand or maintain. Examples include tools like Code Climate and Understand for C/C++. Check also Static Code Analysis Tools.
- Dependency Analysis Tools: These tools help in analysing dependencies to ensure that the code is not relying on deprecated or vulnerable libraries. Examples include OWASP Dependency-Check[32] and Snyk[33].
- Security Scanning Tools: These tools focus on identifying security vulnerabilities in the source code. Examples include Checkmarx[34], Veracode[35], and Fortify[36].
- Unit Testing Frameworks: Though not directly for checking code quality, unit testing frameworks such as JUnit for Java, pytest for Python, and Jasmine for JavaScript help ensure that the code functions as expected. An extended list is available at [37] and at [38]

[24] https://eslint.org/
[25] https://pypi.org/project/pylint/
[26] https://github.com/caramelomartins/awesome-linters
[27] https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
[28] https://about.gitlab.com/
[29] https://bitbucket.org/product/
[30] https://blog.jetbrains.com/space/2021/12/15/best-code-review-tools/
[31] https://www.guru99.com/code-coverage-tools.html
[32] https://owasp.org/www-project-dependency-check/
[33] https://www.getapp.es/software/2047389/snyk-1
[34] https://checkmarx.com/
[35] https://www.veracode.com/fix
[36] https://www.microfocus.com/es-es/cyberres/application-security/static-code-analyzer
[37] https://www.browserstack.com/guide/top-unit-testing-frameworks
[38] https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

- Continuous Integration Tools: CI tools such as Gitlab, Jenkins, Travis CI, and CircleCI can be used to automate the process of building, testing, and checking the quality of your code every time a change is made. A comparison study is available at [39] and at [40] and at [41]

As with the generation of documentation, we do not wish to allocate resources to the development of a tool in this area. We will focus on testing existing solutions and producing recommendations for manufacturers and integrators.

This class of tools will export constraints to the manufacturers and integrators' developers.

6.11Visual launching/stopping tool

During the development and specially the debugging phase, integrators may have to launch some specific nodes. Classically unitary launch files are created (to start one node or a limited number of nodes). Then assembly launch files are generated, launching themselves the unitary launch files "in cascade". This cascade is sometimes implemented on several levels. The result is a mille-feuille that is tedious to use, analyse and maintain. In the development phase, if integrators want to launch a limited number of nodes, they need to create a specific assembly launch file or manually launch the relevant unit launch files one after the other.
In the case of ROS, a basic software is rqt_launch. At this stage, we cannot say whether this tool, although basic, meets our needs, or whether it needs to be upgraded. Rather than initiating development a priori, we feel it's necessary to test this tool in greater depth. If we notice during those trials that the limits of the tool are quite for away, we will recommend it use. If not, we will reconsider to add it development to our roadmap. In this situation we will also have to document the possible to cooperate with the rqt_launch maintainers and not only the creation of a novel tool. This can minimize the amount of necessary workforce from our side but also make us more visible in the ROS community.

[39] https://www.atlassian.com/continuous-delivery/continuous-integration/tools
[40] https://www.guru99.com/top-20-continuous-integration-tools.html
[41] https://smartbear.com/blog/top-continuous-integration-tools-for-devops/

7 Impacts for developers

The popularity of ROS/ROS2 has been built around its relative accessibility: by developing in lines of code in standard languages (Python, C/C++...), it's easy to create a prototype.
We do not want to go against this approach by imposing a rigid MDE framework on developers on manufacturers' side or on integrators side. Our intention is to support the development process with tools where quality and productivity are at stake, and to enable developers as far as possible to adapt the balance between the tool-based approach and the "free" approach to their own feelings and experience.

This minimizes constraints, but it's not possible to be completely free of them.
We have already mentioned above a study recently launched at the university of Stuttgart, which will enable us to better materialize the list of constraints exported to developers who are declining to use the tools we need to guarantee quality and boost productivity. The results of this study will be known in early 2024.
While we refer to the results of this study for the precise nature of the constraints, we have described for each tool whether there will be a constraint, and the stakeholder who will be most affected.

Without this constituting a real constraint, we will promote guidelines and good practices.
The ones we can see at this stage take the form of compliance with various guidelines:

- REP 103,
- REP 105,
- REP 144,
- REP 149,
- REP 2004,
- ROS C++ Style GuideLines,
- ROS Python Style GuideLines,
- ROS YAML Overview

8 Conclusions

Most of the tools we propose to develop will benefit from work previously carried out by Fraunhofer IPA. This minimizes our development effort while maximizing our chances of success. However, the tools proposed by Fraunhofer IPA are “work-in-progress”. They will depend for their future development on input from initial use in the project. Basically, the WP18 will build on top of pre-existing experience and initial, workable tools to build a domain-specific toolchain out of the concepts previously developed at Fraunhofer IPA.

The diagram below illustrates the positioning of the tools we intend to develop in the development environment.

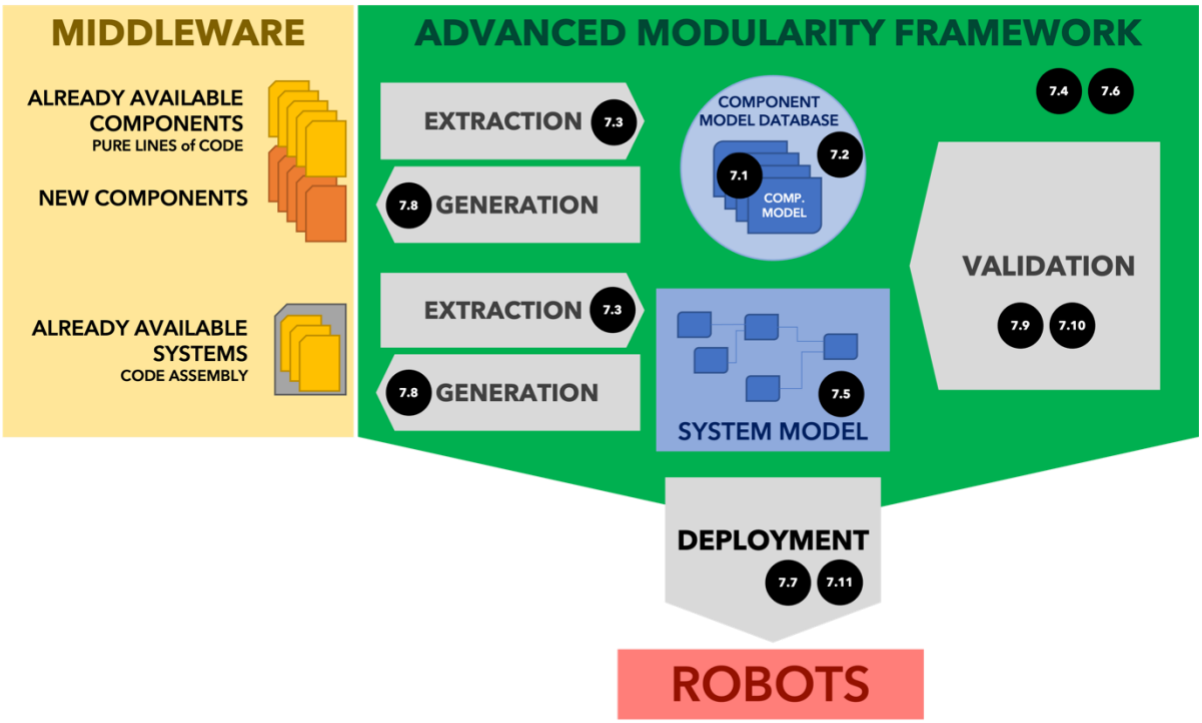

Figure 6 - Positioning of the Advanced Modularity Tools

With this in mind, our development efforts will focus on 8 tools:

- a data structure for component description (tool 7.1)
- a data base of components properties, using the above-mentioned structure (tool 7.2)
- an automatic component properties extractor from code (tool 7.3)
- a component properties visualizer (tool 7.4)
- a data structure for the system – robot – (tool 7.5)
- a tool to model the system as an assembly of components – robot – (tool 7.6)
- a tool that automatically generates code, except for the documentation (tool 7.8) – for documentation we propose to test existing solutions

- a tool for checking incompatibilities and missing information in a system model (tool 7.9)

For 2 tools, we will test existing solutions and make recommendations for use and/or further development:

- a tool that checks code quality (tool 7.10)
- a visual launching/stopping tool (tool 7.11)

For 1 tool, we will draw up recommendations for future developments:

- a tool that creates the robot's software distribution from its model (tool 7.7)

The corpus of tools that will be available at the end of the project will provide a solid, concentrated core of Model Design Engineering tools with a strong impact on product quality and development productivity. Beyond the scope of the project, efforts will have to be continued to increase the perimeter of certain tools (moving the system's data structure from the "logical" scale to the "physical" scale, then to the mechanical scale) or by undertaking work on tools that are not yet covered (software distribution tool).

The 4 tools that export constraints to developers are:

- automatic component properties extractor
- data structure for the system
- tool that automatically generates code
- tool that checks code quality

We have endeavoured to limit this number. The precise list of constraints will be known in January 2024, following the completion of a study conducted at the University of Stuttgart with the help of students.


Europe's Rail


FP3
iam4Rail

9 References

ROSMetamodels: N. Hammoudeh Garcia, M. Lüdtke, S. Kortik, B. Kahl and M. Bordignon, "Bootstrapping MDE Development from ROS Manual Code - Part 1: Metamodeling," 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 2019, pp. 329-336, doi: 10.1109/IRC.2019.00060.

ROSModelextraction: N. Hammoudeh Garcia, L. Deval, M. Lüdtke, A. Santos, B. Kahl and M. Bordignon, "Bootstrapping MDE Development from ROS Manual Code - Part 2: Model Generation," 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), Munich, Germany, 2019, pp. 95-105, doi: 10.1109/MODELS.2019.00-11.

PART C – SAFETY ASSESSMENT

1 Executive Summary

Objective:
Recently, more advanced machines, which are less dependent on human operators, have been introduced on the market. These machines, known as collaborative robots or cobots, are working on defined tasks and in structured environments, yet they can be trained to perform new actions in this context and become more autonomous. Further refinements to machines, already in place or to be expected, include: real-time processing of information, problem-solving, mobility, sensor systems, learning, adaptability, and the capability of operating in unstructured environments (e.g., construction sites). The Commission Report on the safety and liability implications of Artificial Intelligence, the Internet of Things and robotics [1] states that the emergence of new digital technologies, like artificial intelligence, the Internet of things and robotics, raises new challenges in terms of product safety. The report concludes that the current product safety legislation, including Directive 2006/42/EC, contains several gaps in this respect that need to be addressed. Thus, this Regulation should cover the safety risks stemming from new digital technologies.
This means that for a project like FP3 - IAM4RAIL, and more specifically for its WP18, the work cannot be exclusively technical. In parallel with the progression of the TRL level, these additional regulatory issues need to be addressed.

This document therefore describes the elements that need to be compiled to proceed with the Safety Assessment of a collaborative or autonomous railway maintenance robot. It suggests a way of organizing information in a safety case to make it easier to understand for people who will be examining it. This organization is reflected in a template we have called the “Safety Plan”.

Methodology:
This document was conceived in an iterative way. For each theme, one of the participants delivered a proposal to the other members of the working group. This formed the basis for discussion within the group. Then, whenever possible, the proposed piece of methodology was applied to one or more of the WP18 demonstrators. Each iteration was concluded by a feedback phase to refine the proposal.

Conclusion:
A Safety Plan for a railway maintenance robot should be divided into four sections: Purpose of the Safety Plan, System Definition, Safety Proof Concept, and Safety Assessment Report. These sections represent the basic pillars of the safety case for the process change of maintenance measures in the rail system. The Safety Plan should be a covering document which organizes and references the most important documents in the safety demonstration. Our Safety-Proof Concept suggests that the path for such a demonstration should be structured in five perimeters: basic machine safety, information safety, movement safety, inspection safety, and intervention safety.

2 Abbreviations and Acronyms

Abbreviation / Acronym	Description
MAWP	Mutli-Annual Work Plan
WP	Work-Package
ATO	Automatic Train Operation
GoA	Grade of Automation
ETCS	European Train Control System
HAZOP	Hazard and Operability Studies
FMEA	Failure Modes and Effects Analysis
FTA	Fault Tree Analysis

3 Objective/Aim

In the last few years many new technologies have evolved, including autonomous robots, artificial intelligence and new data transmission standards. These technologies have introduced new risks and difficulties in integrating them into the railway sector. This is reflected in the new EU Machine Regulations (2023/1230) [2] and the EU AI Act (2023) [3].
To improve the safety of integration of new robotic technologies and to enhance the transparency between manufacturers, operators and auditors of such systems, one part of this project aims at creating templates and guidelines for creating a safety assessment.

The need to develop modular robots for railway maintenance was outlined in the middleware selection part of this document. The document on overlay for advanced modularity describes the quality and productivity tools to be deployed to make the most of modularity in our context. But there is another area where this modularity can be put to good use, in safety demonstrations.
It would be a waste if development were to proceed rapidly due to advanced modularity tools, but production was slowed down by the necessary safety demonstration phase. This is even more important as the world of robotics is frequently undergoing a renewal of both hardware and software products (this renewal may occur because of obsolescence or because of a steep increase in performance).

It is also important to agree on a common methodology. Consider the case of a robot designed for a new use which results from the assembly of some of the components of several robots used for other purposes. An obvious first step for each of the partners is to try to reuse the largest possible part of the safety demonstrations already carried out within its own organization. But an even more promising, not so obvious second step is for each partner to reuse the largest possible part of the demonstrations already carried out by trusted partners (or by partners following a trusted process). This can minimize the effort it will have to put into its new safety demonstration. While the first, intra-organizational (within a single company) mechanism seems natural, the second, inter-organizational mechanism can be a major habit breaker.
To be sustainable in the long term, it requires balanced contributions from all stakeholders. It should be noted that various balancing solutions exist. These can be a balance in numbers (each contributor provides and withdraws as many demonstration elements as the others) to financial compensation (for those who use the most to those who provide the most). This will be dealt with in subtask 18.1.3 of the work package. Here we are going to concentrate on the more technical aspects of such a system.

We describe the elements that need to be compiled to proceed with a Safety Assessment. We can explain them in detail by referring to the contents of the safety plan shown in Appendix D.

As the industrialization of AI-integrated systems grows, the regulatory framework, particularly at the European level, is in a highly dynamic phase (EU AI act [3], Proposal for a regulation of the European Parliament and of the Council on machinery products – COM(2021) 202). It is therefore important that throughout the project we keep abreast of changes in the regulatory framework to, at the very least, remain internally consistent.

4 Methodology

We worked in an iterative form. For each theme, one of the participants delivered a proposal to the other members of the working group. This basis for discussion was then discussed in the group. Then, whenever possible, the proposed piece of methodology was applied to one or more of the WP18 demonstrators. The next step was a feedback phase to refine the proposal. Finally, this document was reviewed internally and externally before being delivered.

5 Overview of the Safety Plan Content

The following procedure in terms of the safety verification is illustrated with the safety plan. The safety plan is divided into four chapters (Purpose of the Safety Plan, System Definition, Safety Proof Concept, Safety Assessment Report). These constituents represent the basic pillars of the safety case for a process change of maintenance measures in the rail system. Each of them will be described one by one in the following subsections.

5.1 Purpose of the Safety Plan

The safety plan is a high-level or umbrella document which organizes and references the most important documents in the safety demonstration. For this reason, this document does not itself contain the concrete performance of the system components or the scope of the automation. This concretization takes place in the elements of the safety-proof concept.

The safety plan is designed for the following cases:

- initial verification of a maintenance procedure change, regarding the replacement of manual maintenance activities by a defined automated solution, and
- verification of a maintenance procedure change, involving the use of an evolved automated solution (the change may come from the procedure, the automated system, or both).

In all cases, each situation must be properly analysed, even if it follows a documented situation. This means that the document used for the initial verification cannot be used as it stands to support the verification of a subsequent case, e.g., a retrofit of the automated system.

The risk management procedure shall be implemented according to the Standard EN 50128 (Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems), the Commission Implementing Regulation (*EU*) *No 402/2013* (Common safety method for risk evaluation and assessment) as well as NF EN ISO 12100 (Safety of machinery - General principles for design - Risk assessment and risk reduction).
In its most basic form, the risk management system should have a system description and an assessment of the significance of change. If this change to the previous system proposes a significant safety impact, a risk assessment of the new system must be conducted, and measures need to be employed to address the stated risks.

In the context of emerging robot technologies developed for railway maintenance activities, careful consideration needs to be taken in the system definition and the context in which the robots will be functioning (see Section 6.2 for more details). Furthermore, the system itself should be described in terms of the 5 pillars described in Section 6.3: Basic machine safety, Information safety, Movement safety, Inspection safety, and Intervention safety.

5.2 System Definition

The system definition is the fundamental building block on which the safety verification for a change in a railway maintenance process is based. This allows the specific project to be broken down into its most important components and ensures that all aspects of the process change have been considered for a proper safety verification. The overarching goal of the system definition is to create transparency on the purpose, intended environmental context, boundaries, and functions of the system.

Through the course of the project, a template for a system description will be developed alongside corresponding guidelines. To validate these templates, they will be applied to the contexts of each of the 4 applications encompassing this project. In providing these frameworks and guidelines, the FP3 - IAM4RAIL WP18 will provide a starting point for similar robotics projects and reduce the time needed to safely incorporate robotic technologies into maintenance activities.

An example of initial requirements on a system definition that have been already recognized by the working group, is listed below.

- Purpose of the system including a description of the system in terms of the 5 pillars (see section 6.3)
- Operational Design Domain in which the system is intended to function:
    - Could include energy and heat flow, shock, vibration, electromagnetic interference, velocity, weather, environmental restrictions etc.
- System boundaries and interfaces including one or more of the following:
    - other integrating systems,
    - processes (manual/automatic),
    - external systems,
    - physical (e.g., energy and heat flow, shock, vibration, electromagnetic interference),
    - functional interfaces (human, software, AI).
- Internal system components and interfaces including one or more of the following:
    - processes (manual/automatic),
    - physical (interacting),
    - functional (human, AI, software, IT, communication etc),
    - human,

- automation/machine learning,
- data components, processes, and flows,
- technical,
- operational components,
- software systems,
- hardware/communication infrastructure.

A first version of the template mentioned above is available in Appendix F. This version is based on the chassis inspection robot demonstration. Partial uses of this template have already been made on the demonstrators for multipurpose inspection of infrastructure and for disinfection of trains and small stations.

5.3 Safety Proof Concept

It is necessary to structure the path that is to be followed to provide proof of safe operation. Like all technical systems, a railway maintenance robot can be broken down along two axes: the *functional axis* and the *component axis*. The relationships between these two axes can be qualified in matrix form, using Suh matrices for example.
While there are advantages and disadvantages to each decomposition (on the functional axis and on the component axis) for a safety proof, mixing them up can lead to confusion. This includes potential redundancy of information for the reader, difficulties for editors in positioning information in the right place, and other problems. Thus, we choose one axis and stick to it.

Since the use of a robot in a maintenance process includes very various aspects and can involve several components and technologies, we propose to establish categories qualifying the operations (simplified functional axis), to be able to bring a certain genericity to the methodology for achieving a safety approval. However, when it comes to providing proof, it is important to ensure that all the means used (data acquisition, information processing on the hardware and software side, effectors, etc.) are covered by demonstration.
In our analysis, it appeared that one item may be an exception to the rule that we have just mentioned. To better highlight cybersecurity issues in the analysis, we believe that the information transmission between the robot and its environment should be covered by a separate category.

To establish the operations categories, we have drawn on scales created for autonomous systems in other industrial sectors (automotive, aeronautics, etc.). Our division involves the notion of perimeter. It is then within each perimeter that the degree of automation and autonomy comes into play.

The five perimeters we have established are:

1. Basic machine safety,
2. Information safety,
3. Movement safety,
4. Inspection safety, and
5. Intervention safety.

For each of these categories, we are now unable to indicate the precise methods and means that will need to be implemented to measure the effectiveness of the means implemented to guarantee the expected level of safety. Nor will we be at the end of the project. Methods and means are too technology dependent. For example, to detect obstacles in front of a vehicle, different types of sensors can be used: an array of ultrasonic sensors, a laser scanner, or a camera with deep-learning-based image processing software. All these methods have different conditions of use and incur different risks. For example, ultrasonic sensors might fail due to broken cables, the software analysing laser scanner results might contain errors, and the training data for the neural net which processes camera images might be inadequate. To safeguard against a broken wire, its resistance can be measured. Software errors can be detected by systematic testing or formal analysis methods. The accuracy of a neural network can be measured by independently

generated test data. Therefore, no uniform method can exist which shows that “obstacle detection” is safe.
However, we can cite two methods which themselves offer sub-variants. Design based methods are constraining for developers. But as we cannot assign one or other variant to a particular category linked to the product's purpose, we unfortunately cannot be precise about the constraints that will apply to the project's robot developers.

- **Design Analysis and Validation:** There should be documented safety reviews and analyses for the maintenance robot hardware and software, as well as for its design process. For software, the design process comprises requirements analysis, architectural design, module design, implementation and coding, module integration, and deployment on the robot; the result of each phase must be reviewed. Formal specifications and models can help to make requirements more precise. Code review techniques and static analysis tools have been developed to identify safety and security flaws, such as coding errors and vulnerabilities. Furthermore, formal methods such as model checking and program verification can reveal software bugs, and model-based development (e.g., based on semiformal modelling languages such as UML) can help to reduce the likelihood of such bugs. For AI-based systems, amongst other things, training data and hyper-parameters must be reviewed. Furthermore, the software architecture should be reviewed and/or analysed, with respect to critical decisions taken by an AI. If the software is structured into an operating system, middleware and application layer, it may be the case that each layer can be validated separately. For commercially-off-the-shelf components, it might be the case that parts of the safety argument can be provided by the supplier.
- **Systematic Testing:** The main method, but not the only one, for quality assurance of software-based systems is systematic testing. This must be performed at all development stages: unit testing, component testing, integration testing, and system testing. The systematics can be according to implementation (code-based testing) or specification (requirements-based testing). In specification- or model-based testing, test cases are automatically derived from requirements and design models. Ideally, testing of cyber-physical systems such as railway maintenance robots should be first done in a simulation-based testing environment with an automated test execution framework, before actual field testing takes place. For AI-based software, robustness with respect to adversarial examples must be tested, and comparisons with scenario-based user evaluations should be done. Furthermore, it is advisable to perform stress testing for robustness, and penetration testing for intrusion protection.

On the other hand, these categories allow us to specify the nature of the demonstrations that need to be produced. Although we are not in a position to propose a generic approach, the instance of the document corresponding to a specific project will have to specify the means of

proof implemented.

5.3.1 Basic Machine Safety

The most fundamental aspect of a railway maintenance robot is that it is a complex (mechanical and programmable electronic) machine, working in an industrial environment. Therefore, it must conform to the accepted safety rules for such machines. For the basic machine safety of railway maintenance robots, we do not propose a grading scale. The aim is to establish that the machine "per se" is safe for both its users and environment, including third parties: there is no risk of electrocution or electrification, burns, fire, undesirable release of fluids or energy, etc. For this part, we propose to rely on existing norms and standards, such as the CE marking. We have drawn up a list of a priori relevant standards, which we will need to complete and refine throughout the course of the project. The list compiled to date is available in Appendix E of this document. It should be noted that it is important also to define criteria to decide if a certain standard applies to a given robotic system. Applying all available standards to any railway maintenance robot development project may lead to increased development time and cost. The set of standards to apply to a particular project depends on the risk assessment of using the robot for a particular task in the railway environment. This is also the general process that manufacturers follow to apply CE marking to a product.
A categorization of relevant safety standards was given in [9] as follows:

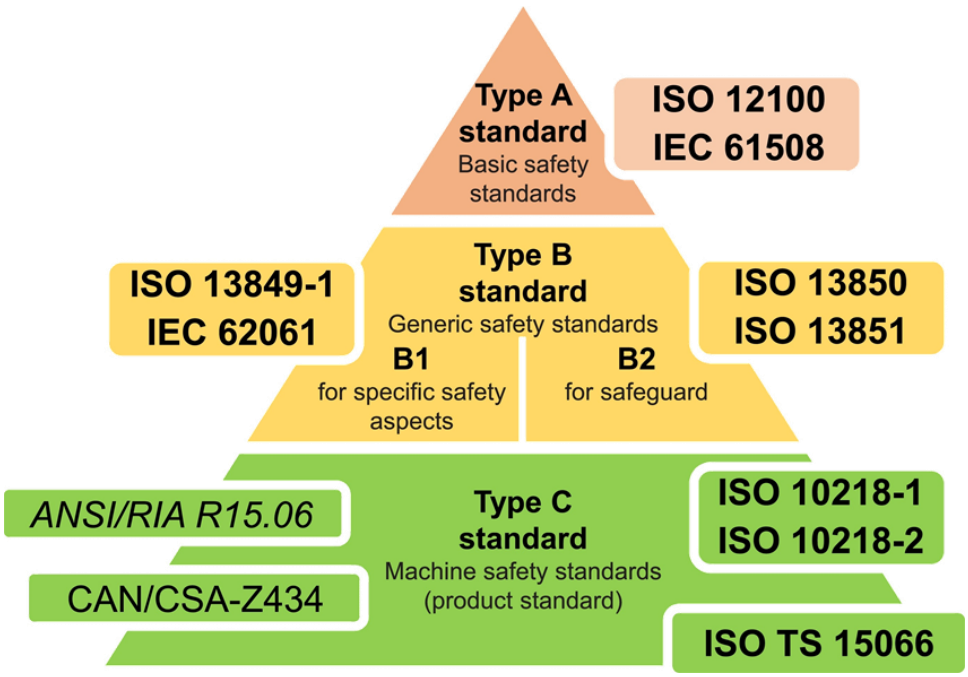

Figure 7 - Categorization of Relevant Safety Standards

The top layer of this pyramid structure is formed by safety standards of Type A, which are basic safety standards for general requirements, such as those given by IEC 61508 or EN 50129. Below that, there are generic (Type B) safety standards – Type B1 standards dealing with specific safety

aspects and Type B2 standards considering specific safeguarding techniques. Even deeper, there are Type C standards considering safety measures for specific machinery such as industrial (stationary) robots. Type C standards take priority over Type A and Type B standards, since they are more specific.
It is important to note that this categorization applies not only to the safety of railway maintenance robots concerning movement and electric shocks, but also to other threats such as IEC 60825-1 for the use of laser beams not to hurt human eyes.

In the further steps of the project, we will use our expertise of the railway context to identify the relevant risks and define the appropriate requirements for a robotic solution.

5.3.2 Information Safety

A more detailed view of a railway maintenance robot considers it as a computer-controlled mechanical machine. Thus, both the safety of its information flow and the safety of its mechanical movements must be guaranteed. For the information flow part, the detailed levels may be classified as follows:

- Information acquisition (i.e., sensing),
- Information processing (i.e., computation), and
- Information transmission (i.e., communication).

As mentioned in the introduction to Chapter 6.3, proof demonstrations falling within the scope of sensing or computation must be managed at the level of each purpose (movement, inspection, intervention) that uses those items.

For a classification of the *information transmission (communication)* in railway maintenance robots, we consider only information which is transmitted from the robot to the outside. Having this specific focus could indeed help us to take better account of cybersecurity issues as previously mentioned. Information transmitted to the robot, e.g., telecommands for movement, inspection or intervention, are considered in the respective perimeter. For the communication of information gathered by the robot, we propose a scale with three grades:

IC0. Information is not shared by the robot,
IC1. Information is shared with non-safety-relevant external systems, and
IC2. Information is used by safety-relevant processes.

Communication type IC0 *(information is not shared by the robot)* characterizes robots which are "closed systems"; they perform maintenance tasks without external intervention. For such robots, it may not be necessary to safeguard the communication.

Communication type IC1 *(information is shared with non-safety-relevant external systems)* subsumes remotely controlled and autonomous robots in non-safety-related contexts. In a remotely controlled robot, it must be assured that the receipt of control commands by the robot is dependable. Even in non-safety-related contexts it may be necessary to safeguard the

communication of the robot with external systems, for the protection of privacy and/or intellectual property rights. There are standards like the IT Baseline Protection Manual of the European Union Agency for Cybersecurity, or the EU's GDPR dealing with these topics.

Communication type IC2 *(information is used by safety-relevant processes)* is used if the information which the robot gathers and transmits is used in safety-related processes. Here, special measures for safeguarding the information transmission between the robot and other systems or humans must be implemented, according to the rules which are applicable for the respective process. Typical concerns include data integrity and timely availability, confidentiality of information, authentication and authorization of actors, non-repudiation, etc.

5.3.3 Movement Safety

A defining criterion for each robot, which distinguishes it from other information processing machines, is that it can move in space, as a whole and/or with different "body parts". The movement of physical masses poses a potential threat to the environment. We can distinguish different criticalities of movement according to the level of autonomy in different application scenarios. For classifying the movement safety of railway maintenance robots, we are proposing a scale with five grades:

MS0. Remotely controlled movement,
MS1. Supervised autonomous movement,
MS2. Driverless movement in controlled areas,
MS3. Driverless movement in dedicated areas, and
MS4. Free and unattended movement.

Movement safety type MS0 (*Remotely controlled movement):* Movement is controlled by a human driver which is either on board or beside the vehicle. The vehicle cannot move on its own, and responsibility for the safety of the movement rests solely with the human driver. In terms of the Automatic Train Operation (ATO) classification, this comprises the Grades of Automation (GoA) 0 and 1 – on-sight train operation and non-automated train operation.
Safety measures may include automated monitored stop functions, speed and separation monitoring, and power and force limiting. Relevant standards include ISO 10218: Robots and robotic devices — Safety requirements for Industrial Robots, Part 2: Robot systems and integration, and ISO/TS 15066: Robots and robotic devices — Collaborative robots.

Movement safety type MS1 *(Supervised autonomous movement):* Robots of movement type MS1 can move autonomously but are supervised by a human at all times. Supervised autonomous movement includes assisted driving where the driver is on board or monitoring the vehicle right next to it or from a remote site. It corresponds to GoA 2 – semi-automatic train operation. In the operation, we must distinguish between two different situations: Normal runs, where everything works as expected, and abnormal runs or emergencies, where the human takes over control.
For the safety proof, we must additionally show that human supervision is always possible, and

that the robot will always accept and follow supervisory commands.

Movement safety type MS2 *(Driverless movement in controlled areas):* Robots of movement type MS2 are designed for work inside controlled areas, e.g., rail service centres, construction and storage halls, maintenance facilities, etc. Within these areas, they perform their tasks autonomously, there is no or only very little need for human supervision. For robots of movement type MS2, interaction with other human workers and third parties can be limited, e.g., by suitable factory safety rules. This way it can be guaranteed that no physical contact occurs between humans and machines. Examples for robots of this movement type would be autonomous rail car underbody inspection robots and autonomous passenger carriage disinfection robots.
There are six main contributors to movement safety in this type: energy level (combination between mass and speed), software, hardware (control), hardware (execution), hardware (sensing), and area control system (that can be subdivided). To show the safety of movement for robots of this type, we need to make sure that the robot will never leave the controlled area during its operation. This may involve the use of redundant sensors and/or external supervision systems.

Movement safety type MS3 *(Driverless movement in dedicated areas):* In contrast to robots of movement type MS2, railway maintenance robots of type MS3 can move on public sectors of the railroad network, inside railway carriages, or in other public spaces like railway stations. They perform their tasks autonomously and potentially unsupervised, but the area in which they work can be dedicated to the robot. That is, the tracks occupied by the robot during operation can be closed for other traffic, the carriage can be emptied of people, and the station (or a dedicated area of the station) can be closed to the public. An example of a robot of this movement type would be an autonomous installation robot which mounts ETCS balises on a dedicated track.
The main contributors to movement safety are the same as with MS2. For movement safety, we need to include measures (e.g., laser scanners) in the robot to survey its environment for humans, other vehicles, animals or other obstacles in its way. Furthermore, we must show that these measures are effective, i.e., will prevent collisions if possible.

Movement safety type MS4 *(Free and unattended movement):* Robots of movement type MS4 can move freely and unattended on public sectors of the railway network or in other public spaces which humans and/or other automated systems are occupying. In terms of automated train operation, this corresponds to GoA4 – unattended train operation. An example for a robot of this movement type would be a fully automatic vehicle for the inspection of the rails and catenary equipment on an ETCS-controlled track.
With this movement type, there are five main contributors to movement safety: energy level (combination between mass and speed), software, hardware (control), hardware (execution), and hardware (sensing). The safety proof for this movement type of robots, which is beyond the scope of the IAM4RAIL project, involves showing all requirements for autonomous vehicles on public roads: traffic rules, safe signalling, safe obstacle detection, etc. It is comparable to the safety proof for autonomous freight trains, shunting locomotives, and other unmanned railway vehicles.

5.3.4 Inspection Safety

Railway maintenance robots not only have to move on the tracks, but they are also designed for a specific purpose. We distinguish two main purposes: *inspection* and *intervention*. An inspection robot is equipped with sensors to measure and analyse its subject, but it will never deliberately interfere with it. In contrast, an intervention robot has actuators with which to repair, improve or modify its subject during the task.
For inspection safety, we are proposing a scale with five grades:

IS0. Non safety relevant inspection,
IS1. Manually inspected safety relevant properties,
IS2. Manual inspection with support functions,
IS3. Semi-automatic inspection, and
IS4. Fully autonomous inspection.

Inspection safety type IS0 *(Non-safety relevant inspection):* Railway maintenance robots of inspection type IS0 are sent out to observe and/or measure certain circumstances, which are important for the operation of the railway network but are not relevant for the safety of humans. Examples of such inspections would be the checking for graffiti on a train, or the wear and tear of the seats in a passenger rail car. Even for robots of type IS0, it may be necessary to ensure that only relevant information is collected, e.g., there might be limitations to video-streaming due to privacy or military reasons.

Inspection safety type IS1 *(Manually inspected safety-relevant properties):* Robots of type IS1 inspect safety-relevant properties. However, they are only used for the collection of data (measurements, pictures, videos, etc.), not for automated data processing. The safety-critical decision is made by humans with the help of this data. Thus, even if the inspection can be considered as still being manual, the means to carry it out have evolved. For example, with the help of an IS1-type robot, operators can read an image on a screen in a comfortable office environment, whereas previously they could view the scene only directly in the workshop. If the modalities are to evolve, it will be necessary to demonstrate that the new conditions for decision-making lead to results of a quality at least as good as the previous ones. Issues which might influence the decision-making process could be the quality and timeliness of a video stream, safety of transmission etc., which must be handled here if they were not yet considered in the safety of Communication type IC2.

Inspection safety type IS2 *(Manual inspection with support functions):* A human conducts the safety-relevant inspection and receives extra, independent information from the machine based on its inspection results. The information provided by the robot (or corresponding robot functionality) supports the task of the human inspector, but it is not necessary to safely complete the inspection. The automated analysis is an “add-on”, which gives additional information but is not in itself safety-relevant. If there is a discrepancy between findings, the human will make the final evaluation and decision. Here it is particularly important to show that there is not an over-reliance on the support function. Similar to inspection safety type IS1, for the safety proof it needs to be shown that the support function does not deteriorate the manual inspection process.
The five potential main contributors to safety are: human, software, hardware (sensing), hardware

(control), and communication.

Inspection safety type IS3 *(Semi-automatic inspection):* With this type, parts of the inspection process are automated by the robot system while other parts are conducted by a human. With robots of inspection type IS3 the processing and analysis of the inspection data obtained by the robot is done by the robot itself, or by some external computer. The machine arrives at safety-critical decisions. However, there is still a human in the loop as a fallback level. There is a threshold associated with each inspection result. If the results of the automatic analysis are beyond threshold, the human is triggered by the robot to do a second evaluation, review a finding or complete the inspection step.
An example would be an underbody inspection robot checking the integrity of brakes. If the robot can localize the brakes and verify that they are in good working condition, the inspection result is positive; otherwise, a human must take over responsibility and inspect the brakes. Another example is an autonomous robot supervising a closed area for intruders. If the camera detects objects which the robot AI with 90% probability classifies as humans, it sends an alarm to the security staff, who can look at the video stream to determine the necessary actions.
The most important task in the safety proof is demonstrating that the threshold is safe, i.e., an automatic decision is reached only if it is beyond reasonable doubt that it rests on firm grounds. For AI-based systems, this may involve the use of automatically generated explanations and their independent automatic checking. If the decision is transferred to a human, it must be safeguarded as for IS1 and IS2.

Inspection safety type IS4 *(Fully automatic inspection)*: When operating correctly, the inspection process with robots of type IS4 is fully automated with no direct human intervention and/or interaction required. Robots of this inspection type perform their work without any human fallback layer. The robot or remote computer decides safety-critical issues on its own, humans do not interfere with the decision.
For robots of this inspection safety type, it needs to be shown in the safety assessment that the automation is equal or better at performing the task than a human expert. This can involve systematic testing, online monitoring, training data analysis, scenario-based user analysis, and other verification methods.

The following picture shows the safety responsibility spectrum between IS2, IS3 and IS4.

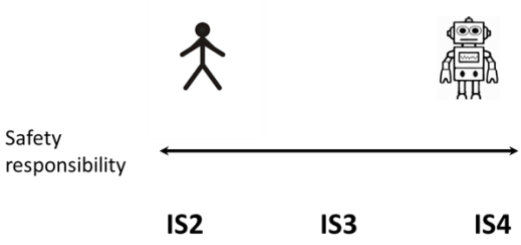

Figure 8 - Human vs Machine Responsibility

5.3.5 Intervention Safety

Intervention refers to tasks where a robot is physically manipulating objects of the rolling stock or railway infrastructure, for installation or maintenance. For the intervention safety, we are proposing a scale with five grades:

IV0. Non safety relevant intervention,
IV1. Manual intervention for safety-relevant properties,
IV2. Manual interventions with support functions,
IV3. Semi-automatic intervention, and
IV4. Fully autonomous intervention.

Intervention type IV0 *(Non-safety relevant intervention):* We say that an intervention is of type IV0, if the manipulations needed to maintain an object are not safety critical. For example, cleaning graffiti off of an asset due to cosmetic reasons using a robot would be an intervention of type IV0. For robots of this intervention type, it may not be necessary to give a proof of intervention safety (however, in most cases it will still be necessary to prove the other perimeters).

Intervention type IV1 *(Manual intervention for safety-relevant properties):* In interventions of type IV1, the human has full control over the robot function/system that is used to fix or intervene in a safety critical part of the system element. Here the human, potentially at a remote location, is relying on the sensor and movements of the robot as a proxy to complete an intervention task. The robot might also augment and amplify the physical capabilities of a human, e.g., to lift heavy weights. As another example, a robot for repairing damaged crossings could be remotely controlled by a human welding expert who does not need to travel to the respective location of the switch. All the welding parameters are set by the human tele-operator.
With intervention type IV1, a major concern is the safety of the command-and-control interface, including (potentially wireless) data transmission. As far as this is not handled by information communication class IC2, adequate measures must be taken that the robot reacts properly to the issued commands, and that the effects are communicated properly to the human operator.
For the safety proof, it must be verified that these measures are effective. Furthermore, the security of the communication must be guaranteed.

Intervention type IV2 (*Manual interventions with support functions):* An intervention process of type IV2 is performed by the human with the robot (or some automatic functionalities) for support. Here the human is always in control and can choose to perform the tasks with or without robot assistance. The support function eases the task of the human operator but is not necessary to perform the safety-critical task. For example, if a human is welding a joint, a support system could check and adjust the temperature while welding, measure geometric tolerances with special tools and provide real-time monitoring of the quality of the welding.
For support functions of type IV2, it must be shown that a malfunction cannot compromise the safety of the system. In particular, the supporting robot should not impose an additional threat to the human operator and/or the task at hand.

Intervention type IV3 *(Semi-automatic intervention):* Intervention processes of type IV3 are characterized by the cooperation/collaboration of human and robot. Certain elements of the intervention process are automated, while other elements are performed by a human. This corresponds to Level 2 – human/machine teaming – in EASA's classification of AI applications in aerospace. For example, a defective infrastructure part could be automatically removed by a robot, and a human could install a new replacement part.
The main additional concern in a safety proof for intervention processes of type IV3, compared to type IV2, is to show that issues from the automated process will be noticed and corrected by the human partner.

Intervention type IV4 (*Fully autonomous intervention):* Intervention processes of type IV4 are fully performed by a robot without any human input or actions. For example, an automated Balise installation robot could travel to a dedicated location, determine the installation points on the appropriate sleeper, drill holes, place a Balise and bolt it down, all fully automatic.
For fully automatic intervention processes, it must be shown that the process execution itself cannot harm people. For example, it must be assured that a fully automated disinfection robot will only turn on the UVC light or spray chemical if there are no humans in its vicinity. This is similar to the proof of movement safety. Furthermore, it needs to be shown in the safety assessment that errors in the process execution are revealed and corrected. Usually, a fully automatic intervention will be followed by an (automated or manual) inspection or testing of the object. This inspection must be proven according to its respective inspection type. For example, the automatic Balise installation robot could use a measurement of the torques of bolts on sleepers and a visual inspection system to see whether they are attached properly.

To close this chapter 6.3, we propose the following table that shows the different categories into which the 4 WP18 in-situ demonstrators fall.

Table 5 - Categorization of the Project Demonstrators

	Basic machine safety	Information safety			Movement safety					Inspection safety					Intervention safety				
	No category	IC0	IC1	IC2	MS0	MS1	MS2	MS3	MS4	IS0	IS1	IS2	IS3	IS4	IV0	IV1	IV2	IV3	IV4
Multipurpose Inspection Robot	X		X	(X)	X	X	X	X	(X)	X			X	X					
Object installation robot	X		X					X					X					X	
Underbody inspection robot	X		X	(X)	X	X	X			X	X	X	(X)	(X)	(X)	(X)	(X)	(X)	(X)
Disinfection Robot	X	X			X	X	X	X											X

In this table the X sign indicates “for accomplishment within the project” and the sign (X) “for future accomplishment after the end of the project”.

The relatively small number of demonstrators in the project means that not all categories can be covered. This is especially true for inspection and intervention safety, as the 4 robots are evenly distributed between these 2 classes.

5.4 Safety Assessment Report

All the elements mentioned above lead to a central safety assessment report.
If the object of detection is significant, obtaining a safety assessment report from the relevant applicable assessment body is required. A template for the significance test will be provided within the project. Depending on the complexity of the project, it is possible to divide the safety assessment into partial safety assessments.

The final deliverable is a template for the safety report that will be developed alongside corresponding guidelines. By providing a framework for the safety assessment report, those integrating robotics into the maintenance field will have a list of requirements fields that need to be addressed to successfully present the safety argumentation. Furthermore, agnostic to the maintenance applications and robot technologies there will exist a common standardization outlining the essential content. This will ultimately reduce the time needed to safely incorporate robotic technologies into maintenance activities for (a) the manufacturer, (b) those integrating technology into their processes, and (c) assessment bodies verifying the adherence to the agreed upon safety processes.

An example of such a safety report would contain clear versioning between drafts and the appropriate signatures on the final draft.

Overview section: The first section would contain a high-level project overview followed by the changes in the processes for the target system. Ending this section would be an explanation of the norms used during the process of risk management.

System definition: A description of the previous system is to be described first following a description of the target/new system. Along with the new system description is a statement describing the purpose of the goal of the new system, the target system environment and system boundaries, functional description of the system, and internal and external interfaces.
Differences between the changes between the old and new systems need to be clearly defined.
This should also include the list of the applicable rules and regulations.
The basis of any safety argument is the identification of potential dangers associated with the operation of the robot. System definition must include a risk analysis that can be done using methods like Hazard and Operability Studies (HAZOP), Failure Modes and Effects Analysis (FMEA), or Fault Tree Analysis (FTA), see references [6], [7] and [8].
If safety features are used, they must be listed. Such features can be integrated into the robot and/or process to enforce safe operation. With respect to mechanical safety, this can include emergency stop buttons, guards, interlocks, and safety sensors to prevent or mitigate accidents. With respect to electric risks, insulations, fuses, circuit breakers, etc. can be used to safeguard against electrical shocks and overloads. For information processing components, redundancy and fault-tolerant hardware, monitors, watchdogs and timers can reduce the risk of failures during operation. For software, coding rules such as “strict exception handling” and “strong typing” can improve safety, and mechanisms such as “strict access control” and “strong data encryption” can

improve security. For data, backup policies can improve resilience to faults.
Finally, the report should contain an overview of the current safety measures and assumptions of the risk assessment; i.e., existing templates/certifications that delimit the boundaries of the risk assessment or operational limitations. It specifies under what circumstances the robot may or must not be used, which skills are necessary for its operation, which precautions must be taken, and which maintenance tasks to the robot itself must be performed. Additionally, it should be specified how incidents are to be handled: where they must be reported, and how it is decided which changes in the process or design are necessary (incidence response plan). Ideally, this document also details how to handle software updates and patches (change management plan).

Evaluation of significance: It needs to be clearly stated what methodology was used to perform the evaluation and the outcome or results of this evaluation. In this section, also details of the involvement of an external assessment body should be described.

Risk Assessment: In the risk assessment portion of the document the methodology used needs to be described along with the results and outcomes of the assessment. The risk assessment will likely that the risk assessment will be done in phases, in which each phase should be described. Example: initial assessment, categorization of risks, consolidation of risks, etc.

Evidence of safety: Here it should be clear what measures are used to ensure the safety of each of the components, as well as the system as a whole. There should be transparency between the identified risks and the corresponding measures addressing said risks. The evidence of the mitigating strategies for the identified risks could take the form of certificates, norms, explicit, tests, measures, and processes.

Change in safety aspects: An identification of known or foreseeable ways safety aspects could change could be also addressed and documented.

6 Towards a Unified Safety Process for Railway Maintenance Robots

While this document is intended to be exhaustive in its presentation of the elements to be supplied for the safety assessment, it does not yet set out how the work is to be carried out. Not all work can be parallelized. Some elements feed on others. These logical sequences must therefore be materialized. Any possibilities for simplifying or bypassing certain stages must be justified and explained. The development of an adequate process and the associated guidelines will be an important part of the future work to be carried out in task 18.2 of the work package. Accordingly, this section of the present document is to be extended at a later stage.

7 Conclusions

In this deliverable we have outlined the basic pillars of a safety case for the process change of maintenance measures in the rail system. We have described a template for a Safety Plan, which is divided into 4 chapters (Purpose of the Safety Plan, System Definition, Safety Proof Concept, Safety Assessment Report).

The first section “Purpose of the Safey Plan” indicates that the safety plan is a covering document which organizes and references the most important documents in the safety demonstration. For this reason, this document does not itself contain the concrete performance of the system components or the scope of the automation. This concretization takes place in the elements of the safety-proof concept. It also presents the 2 cases for which the document is designed:

- initial verification of the maintenance procedure change regarding the replacement of manual maintenance activities by a defined automated solution, and
- verification of the maintenance procedure change involving the use of an evolved automated solution (the change may come from the procedure, the automated system or both).

The second section “System Definition” allows the specific project to be broken down into its most important components and ensures that all aspects of the process change have been considered for a proper safety verification. The overarching goal of the system definition is to create transparency on the purpose, intended environmental context, boundaries, and functions of the system.

The third section “Safety Proof Concept” structures the path that is to be followed to provide proof of safe operation. The path has been organized around 5 categories. Four of those five categories are based on the macroscopic machine functions (basic machine safety, movement safety, inspection safety and intervention safety). A unified categorization would have been more difficult to achieve by working on technologies or components. These can be very varied in robotics. The last category (Information safety) is an exception. It concerns the communication of information between the robot and its environment. This enables us to emphasise cybersecurity issues, which are becoming increasingly important in our society.
For each category, we established what had to be demonstrated. Our original intention was also to suggest ways of establishing the "how" for each category. Unfortunately, it became clear to us that, here too, technological diversity makes it impossible to unify methods for measuring the effectiveness of devices in meeting safety requirements.

All the elements mentioned above lead to the fourth and final section, a central safety assessment report.

This deliverable is the fruit of initial work that needs to be enriched. We will be working on two types of improvements over the coming months:

- We'll be developing or continuing to develop templates and guidelines to help write the safety plan sections themselves.
    - This will be the case for "basic machine safety", where the relevance and contribution of the numerous standards to risk management in the railway context will be highlighted.
    - For the "system definition" a first template has been established. Its application to several of the project's demonstrators should enable it to be enriched.

- A guide to the correct classification of a system in the categories useful for the "safety proof concept" will probably be necessary.
- A template for the safety report will be developed alongside corresponding guidelines.

- While we have specified here the elements to be supplied for the safety assessment, we have not detailed the way in which the various necessary activities are to be carried out. The second axis will be the development of a Unified Safety Process for Railway Maintenance Robots.

8 References

[1] European Commission 2020: Commission Report on the safety and liability implications of Artificial Intelligence, (COM/2020/64 final), https://commission.europa.eu/publications/commission-report-safety-and-liability-implications-ai-internet-things-and-robotics-0_en

[2] European Parliament and Council 2023: Regulation (EU) 2023/1230 of the European Parliament and of the Council of 14 June 2023 on machinery, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02023R1230-20230629

[3] European Parliament 2023: Artificial Intelligence Act. https://www.europarl.europa.eu/doceo/document/TA-9-2023-0236_EN.html

[4] European Agency for Safety and Heaalth at Work: Directive 2006/42/EC - machinery directive https://osha.europa.eu/en/legislation/directives/directive-2006-42-ec-of-the-european-parliament-and-of-the-council

[5] European Parliament: Proposal for a regulation of the European parliament and of the council on machinery products – COM(2021) 202, https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:52021PC0202

[6] EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems

[7] European Commission: Commission Implementing Regulation (EU) No 402/2013 on the Common safety method for risk evaluation and assessment and repealing Regulation, https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2013:121:0008:0025:en:PDF

[8] NF EN ISO 12100 Safety of machinery - General principles for design - Risk assessment and risk reduction

[9] V. Villani, F. Pini, F. Leali, and C. Secchi, “Survey on human-robot collaboration in industrial settings: Safety, intuitive interfaces and applications,” Mechatronics, vol. 55, pp. 248-266, Nov. 2018. doi: 10.1016/j.mechatronics.2018.02.009, https://www.sciencedirect.com/science/article/abs/pii/S0957415818300321

[10] IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems

[11] EN 50129 Railway applications - Communication, signalling and processing systems - Safety-related electronic systems for signalling

[12] IEC 60825-1 Safety of laser products

[13] ISO 10218: Robots and robotic devices — Safety requirements for industrial robots, Part 2: Robot systems and integration

[14] ISO/TS 15066: Robots and robotic devices — Collaborative robots.

PART D – VISION OF ROBOTICS' IMPACT

1 Executive Summary

Objective

Today, the robotics sector invests little in maintenance applications, and even less in railway maintenance applications. The modularity is a technical policy able to make the railway maintenance sector attractive through the massification of robotic components.
By better highlighting the tangible benefits on a large scale that end-users and techno-providers will be able to derive from railway robotics, it is a question of giving confidence so that the necessary investments take place. The benefits we are talking about are not only economic. They may concern the availability of assets, the health and motivation of workers...

Offering a vision of the impacts of robotics on railway maintenance means creating the conditions for the Demand Readiness Level and the Manufacturing Readiness Level to progress together with the Technology Readiness Level.

In the first year of the project, it was not possible to produce a concerted vision of the impact of robotics on railway asset management. Nevertheless, this document presents the methodologies investigated and those selected for subsequent implementation.

Methodology

The methods that can help establish a vision can be bottom-up (concatenation of use cases) or top-down (high-level vision for which compatibility with practical cases is ensured). Bottom-up methods do not seem suitable to us because to be sufficiently robust, they require a significant deployment of resources.
Different top-down methods were therefore researched, proposed to the work-package partners and selected for implementation in the coming months.

Conclusion

Possible approaches fall into 2 families: bottom-up and top-down methods. Due to the difficulties associated with the generalization stage, bottom-up approaches were quickly discarded. Various alternative top-down methods were therefore examined.
The approach adopted is a mix between a top-down analytical approach and a top-down fictional approach.
The analytical approach is based on a breakdown of maintenance into more basic processes. For each of the elementary processes, a short list of relevant indicators (in the context of the introduction of robotics) is proposed. Reference levels are determined. The last step consists of evaluating the evolution of these indicators on a scale of approximately 5 years.
The fictional approach is inspired by Red Team Defense offered by Paris Sciences & Lettres to the French armies. They propose, over a longer time horizon, futures for which the probability of occurrence is not the key point. It is the reactions to be implemented in the face of these new situations that have important value. creating a collective imagination in addition to more traditional commercial relationships can also be a strong glue in a new-born ecosystem.
The total duration of the selected approach is 18 months, based on 4 stages for the analytical approach and on an iterative work of 6 to 9 months for the fictional part.

2 Abbreviations and acronyms

Abbreviation / Acronym	Description
TRL	Technology Readiness Levels (TRL) are a type of measurement system used to assess the maturity level of a particular technology. Each technology project is evaluated against the parameters for each technology level and is then assigned a TRL rating based on the projects progress. There are nine technology readiness levels. TRL 1 is the lowest and TRL 9 is the highest.
DRL	"Demand Readiness Level" is an additional scale to Technology Readiness Level, which will relate to the degree of maturity for the expression of a need by a customer on a given market including the lead markets for eco-innovation.
MRL	The manufacturing readiness level (MRL) is a measure to assess the maturity of manufacturing readiness, similar to how technology readiness levels (TRL) are used for technology readiness. They can be used in general industry assessments,[1] or for more specific application in assessing capabilities of possible suppliers.
KPI	Key Indicator Performance
WP	Work-Package
UIC	Union Internationale des Chemins de Fer – International union of railways

3 Objective/Aim

Today, the robotics sector invests little in maintenance applications, and even less in railway maintenance applications. To obtain the robots they need, the rail industry must therefore finance all the development work. However, the fleets involved in each type of maintenance operation are relatively small (a few dozen robots). Amortising developments on such small populations is complex.

One response to this problem is the modularity policy put forward by this work-package. Although modularity maximizes the chances of profitability for both technology providers and end users, it does not provide all the answers when it comes to the long-term viability of robotics in maintenance. Modularity means that the entry ticket can be passed on to a greater number of applications. It is therefore more likely to be bearable. But at this point, they are still just words. By better highlighting the tangible benefits on a large scale that end-users and techno-providers will be able to derive from railway robotics, it is a question of giving confidence so that the necessary investments take place, so that this entry ticket is distributed among stakeholders.

The benefits we are talking about are not only economic. They may concern the availability of

assets, the health and motivation of workers...

Offering a vision of the impacts of robotics on railway maintenance means creating the conditions for the Demand Readiness Level and the Manufacturing Readiness Level to progress together with the Technology Readiness Level.

In the first year of the project, it was not possible to produce a concerted vision of the impact of robotics on railway asset management. Nevertheless, this document presents the methodologies investigated and those selected for subsequent implementation.

4 Investigated methodologies

Nowadays, the market has extremely get developed, and the introduction of digitalization, new technologies, AI, IoT and so on become a must. In this perspective, companies have introduced digitalization to all their strategic processes including maintenance as it is considered as one of the development levers of many businesses since it has a direct impact on many KPIs continuously monitored.

Defining and monitoring KPIs aimed at getting a global view of the benefit introduced by robotics in the railways environment is not simple, but necessary. Rational approaches to the theme require time to be investigated so as to avoid possible conclusions which are correct for a certain subsystem of possible applications of robotics, but not as a whole.

In addition, Authorities still need to adhere to appropriate specifications that can encompass the use of robotics in the railway sector, especially in strategic processes such as rail maintenance.

4.1 Types of approach

Impact assessments carried out to date in the rail industry have been at the level of a single use case or a small group of use cases. The investigated use cases remain fairly limited compared with the volume of possible applications for robotics in the maintenance sector, or even more broadly in railway asset management.

Therefore, it is difficult to use these documented cases to interest a vast ecosystem over the medium term, and a more global approach is needed.

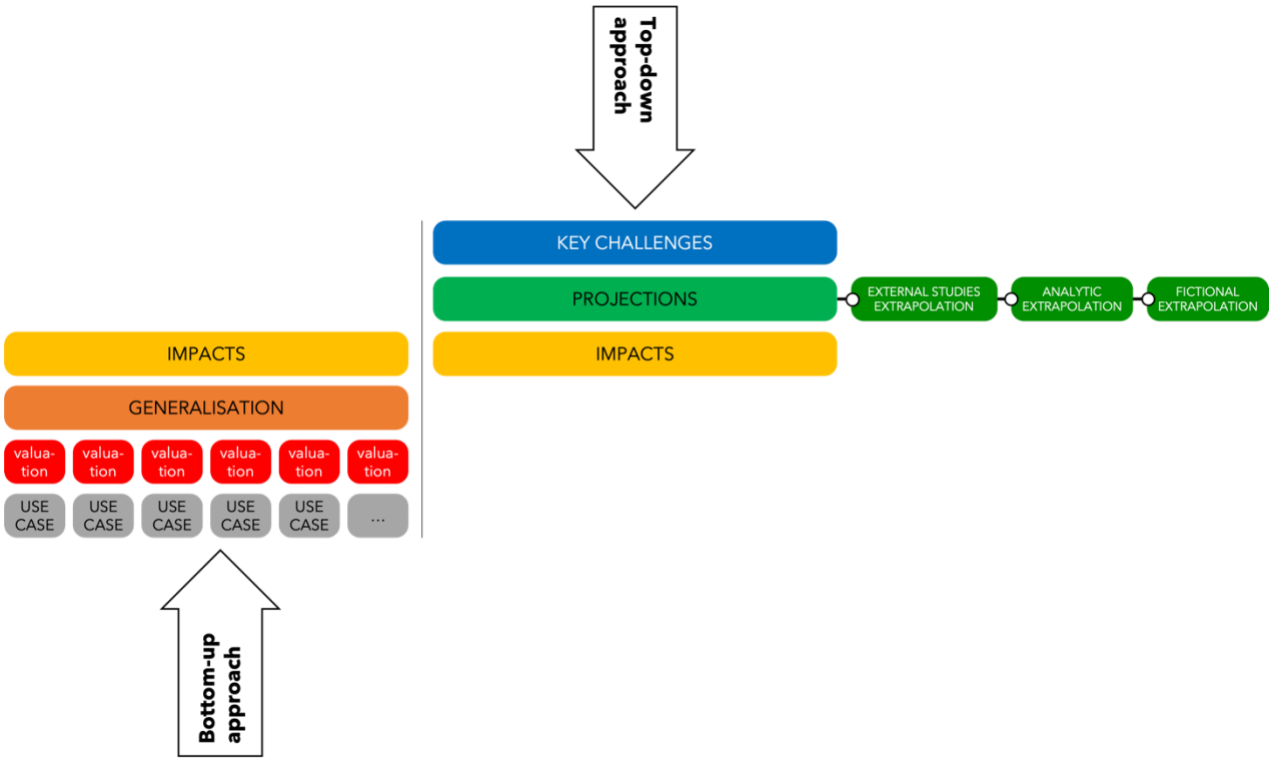

Figure 9 - Types of approach

We could reason on a micro scale for a large number of use cases, then aggregate them to move on to the macro scale (bottom-up approach). However, this would be a long and costly process, so we propose to use direct macro-scale approaches (top-down approaches). In these macro methods, we can use an analytical decomposition or, at the opposite end of the cognitive spectrum, a method involving the imagination and based on fiction.

A final type of top-down approach is the reuse of macroscopic studies already carried out in other industrial sectors, and their extrapolation to the rail maintenance sector.

4.1.1 Bottom-up Approach

We have just indicated that the bottom-up approach is not relevant to establishing a vision. So we won't describe it in detail here. There are, however, elements derived from this approach that can be used in top-down approaches.

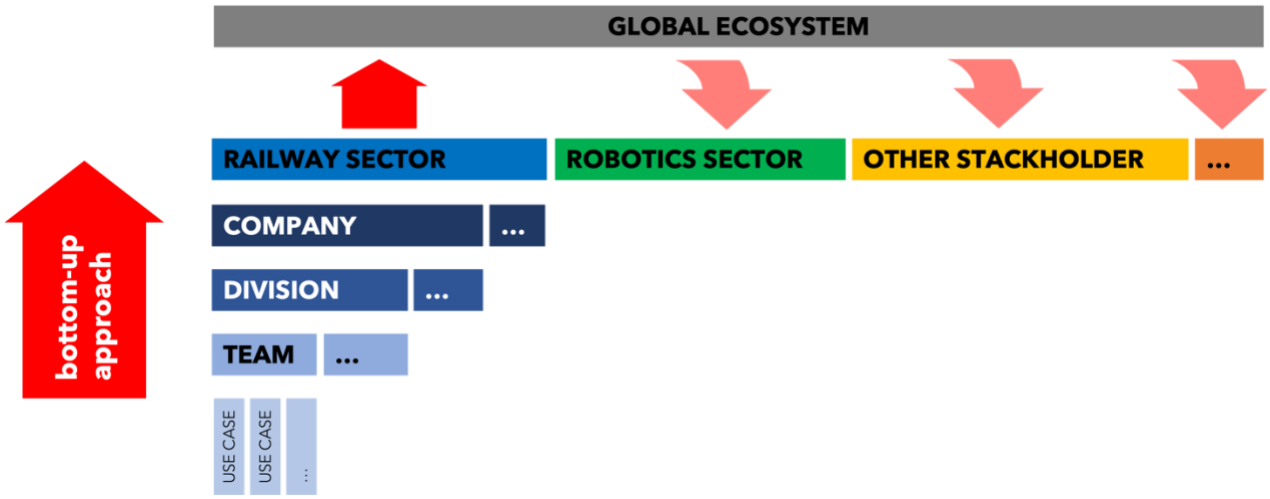

Figure 10 - Bottom-up approach

During the first year of the project, some KPIs have been defined for each use case involved within the WP18, in order to concretize possible gainable results from the experimental campaigns and in order to take into account the fact that demonstrators will not be at the TRL level of a system ready for industrialization. Concatenating and generalizing the elements established for these use cases is akin to a bottom-up approach. Once again, this is not what we are going to do. Nevertheless, the various indicators mentioned at the beginning of this paragraph should inspire top-down approaches.

4.1.2 Top-down extrapolation approach

Macroscopic prospective studies of the impact of robotics do exist[42] [43] [44] [45]. They may concern a particular industrial sector or society as a whole. Their conclusions are not always consistent with each other, and the data and models used to establish them are sometime missing.

Some bodies, such as the International Federation of Robotics, offer annual reports providing general information and trends that could also be exploited[46].

[42] https://www.mckinsey.com/featured-insights/future-of-work/jobs-lost-jobs-gained-what-the-future-of-work-will-mean-for-jobs-skills-and-wages#/

[43] https://www.aeaweb.org/articles?id=10.1257/pandp.20201003

[44] https://mitsloan.mit.edu/ideas-made-to-matter/a-new-study-measures-actual-impact-robots-jobs-its-significant

[45] IFR position papers : https://ifr.org/papers

[46] https://ifr.org/free-downloads/

A major difficulty with this approach lies in the keys to be used for extrapolation. To illustrate this difficulty, let's take an example. The table below compares the number of industrial robots installed on automotive production lines in 4 of the world's most robotic countries (in terms of annual units deployed), as well as the variation between 2021 and 2022.

Table 6 - Industrial robots deployment in the automotive industry

Country	number of cars produced	number of robots installed	change 2022 vs 2021
China	27 M	73 k	+26%
US	10 M	14 k	+47%
Germany	3,7 M	6,7 k	-27%
Korea	3,8 M	5,4 k	-5%

While it's easy to keep track of the number of new robots installed each year (just follow the flow out of roboticists' factories), it's much more complex to know at a given moment the number of robots installed. The number of robots used to produce a vehicle would be an interesting indicator, but it is not readily available. It's more complex to know when a robot has been decommissioned. According to our table, Germany and China are apparently moving in completely opposite directions. Does this reflect a different positioning in the robot acquisition cycle (Germany renews its robots while China builds new factories)? Does it reflect a more intense robotization effort in China than in Germany for an identical positioning in the acquisition cycle (whereas the lower cost of labor might suggest an opposite trend)? Is it simply a reflection of a sector that is growing in China and contracting in Germany, even when positioned identically in the acquisition cycle? It's clear that figures can be used to support a preconceived discourse, but that using them objectively is complex. This is all the more true as the figures used reflect punctual phenomena (number of annual installations) and not a real contribution to the act of production.

4.1.3 Top-down analytical approach

This approach will be based on 4 major steps.

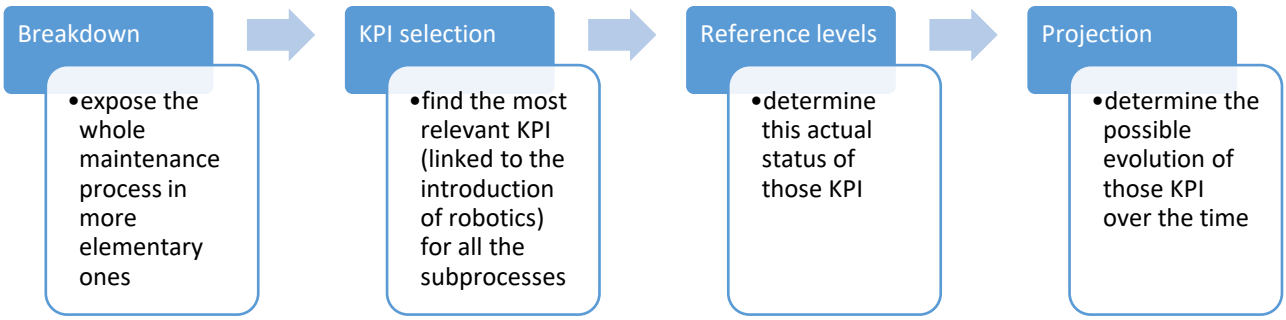

Figure 11 - Diagram of the top-down analytical approach

4.1.3.1 breakdown

As a first step, we propose to break down maintenance, a complex function of the railway system, into a set of elementary functions. Many international norms started to give more interest to maintenance processes and therefore they have deployed specific rules to handle them.
Regarding the notion of maintainability of equipment, the French NF X60-000 standard presents the guidelines for designing a maintenance process to meet its technical and economic challenges. It breaks down in particular the maintenance into processes:

- Ensure the health and safety of personnel and protect the environment during maintenance operations.
- Draw up asset maintenance budgets
- Manage data
- Optimize results
- Design and implement modifications and new work
- Issue operational documentation
- Issue spare parts
- Supply internal and/or external manpower
- Supply tooling, support equipment and information systems
- Provide the necessary infrastructure
- Prepare maintenance operation
- Schedule maintenance operation

- Prevent feared events (preventive maintenance)
    - Steering
    - Realisation
    - Control
- Restore assets to required condition (corrective maintenance)
    - Steering
    - Realisation
    - Control
- Manage Maintenance
    - Manage all the maintenance processes
    - Draw up the maintenance policy
    - Draw up the maintenance strategy
    - Report

This can for example be used to have more directed KPI and to sketch out more direct effects without entering a very micro level.
This breakdown is the result of a first iteration. It has still to be refine with the partners in regard to other relevant norms (see below).

European norm NF EN 13306 addresses the different terminology used in maintenance (Preventive, corrective, predictive, etc...). This is probably one of the most important standards because it describes and gives indications of the maintenance operations to be Implemented. This standard does not directly propose a breakdown. However, we will endeavor to respect the terminology of this standard to ensure that our work is properly understood.

In the realm of operational safety management, the NF EN 60300 series of standards provides the necessary framework. This series addresses the operational safety of products, processes, systems, or services, encompassing human, software, and hardware aspects. It plays a vital role in planning and executing dependability activities, incorporating requirements related to safety and environmental concerns.

The NF EN 16646 standard integrates the management of physical assets within the scope of maintenance activities. It outlines the interactions between maintenance processes and physical asset management processes, emphasizing the importance of maintenance throughout the asset's life cycle.

Additionally, ISO 55001 specifies requirements for establishing, implementing, maintaining, and improving an asset management system that oversees the life cycle of an organization's assets, irrespective of asset type. It is designed for use by those engaged in establishing, implementing, maintaining, and enhancing an asset management system and can be applied to all types of assets across organizations of various sizes. Effective asset management allows businesses to maximize

their potential for achieving objectives, leading to increased customer and stakeholder satisfaction, as well as enhanced trust. Maintenance stands out as one of the prominent tools in assets management, serving as a key component in this integral process.

Quality management system, as described by the ISO 9001 standard, is another critical aspect affected by the maintenance process. The quality management system of a company covers a vast scope, and certain maintenance aspects fall under the ISO 9001 standard, such as ensuring the compliance of production assets and physical assets.

Those standards (NF EN 60300, NF EN 16646, ISO 55001 and to a lesser extent ISO 9001) can be used to enhance and clarify the breakdown provided by NF X60-000.

4.1.3.2 KPI selection

With our final breakdown we will have to find for each subprocess a short set of the most relevant KPI (not more than 3 or 4). The stakes are not the same from one process to another. As an example, let's look at the process of providing operational documentation. The people in charge are not very prone to serious workplace accidents or musculoskeletal disorders. And even if they did, maintenance robotics would have little impact on this risk. On the other hand, operators in charge of production are much more exposed, and maintenance robotics can have an impact on this risk. It remains to be determined whether workplace health is preferred to accidents or musculoskeletal disorders.
It is also conceivable that the set of indicators selected might not be the same for rolling stock maintenance as for infrastructure maintenance.

Some subprocesses may not be directly impacted by the introduction of robots in railway maintenance activities. For example, in the maintenance manage subprocess, the reporting activities will not be concerned. Their results will change, but not really the way things are done. These activities will therefore be identified and excluded from subsequent stages.

To help in the selection of relevant KPIs, we can recall the main motivations that can justify the deployment of projects involving robots:

- **Automated inspection for more precision**: Using robots equipped with high-resolution cameras and sensors allows detailed and accurate inspection of railway rolling stock. These robots can detect even the smallest defects or signs of wear, helping identify issues before they become severe.
- **Predictive maintenance**: By integrating data collected more from sensors on the assets or through robots with artificial intelligence algorithms, predictive models can be developed. These models can forecast when specific parts will need maintenance, enabling timely interventions and reducing the risk of sudden breakdowns. A certain massification of preventive operations makes it easier to ensure the availability of certain critical resources. Maintaining a high level of individual attention may require new resources to take action.

Today, robots are the only technology that makes this possible. There may therefore be a strong link with the "optimization of resources" issue below.

- **Preventive maintenance**: Robots can perform preventive maintenance tasks such as lubricating specific parts or adjusting components. This helps prolong the lifespan of assets and avoids unexpected service disruptions.
- **Component replacement**: Some component replacement tasks can be automated using specialized robots. These robots can be designed to handle heavy or complex parts, improving efficiency, and reducing the risk of workplace injuries.
- **Resource optimization**: Automating maintenance processes allows more efficient use of human resources. Operators can focus on high-complexity tasks while robots handle repetitive and physically demanding activities.
- **Detailed reports**: Robots can generate detailed reports for each maintenance operation performed. These reports can include data about the status of components, tasks performed, and overall conditions of the rolling stock. This detailed documentation is valuable for future maintenance planning.
- **Reduced downtime**: Automation of maintenance processes can significantly reduce rolling stock downtime. Maintenance activities can be performed more quickly and efficiently, allowing rolling stock to return to service faster.
- **Adaptability and scalability**: Robotic systems can be designed to be highly adaptable and scalable. They can be configured to handle different types of rolling stock and can be easily tailored to the specific maintenance needs of various railways.
- **Infrastructural efficiency:** novel robotic systems may allow railway undertakings to carry out inspection tasks by overcoming costly infrastructures, hence allowing a global optimization of the maintenance plans.

Standardization can also help the approach. European norm NF EN 15341 allows to establish comprehensive standards for maintenance indicators which incorporates indicators into a dynamic maintenance process, even able to merge pros of conventional processes. Once appropriate indicators are defined, their implementation involves the use of dashboards for monitoring and associated corrective actions. The primary objective is clear: enabling a thorough assessment and enhancement of the performance of your machinery fleet.

Finally, as already mentioned, the KPIs established in the WP18 use cases will also be used there.

4.1.3.3 Reference levels

The next step is to collect the current reference conditions for all the selected KPIs. These elements will be collected from WP partners or, where appropriate, from organizations federating rail players (e.g. UIC).
These reference levels are fairly high-level information that should not be sensitive for sharing between partners. If any indicators are identified as sensitive during the process, they will be dealt

with only on a relative basis during the rest of the process.

Those reference levels will have to be extrapolated to offer a vision on a European scale. Although the concatenation of partner data covers a significant part of the sector, it is not exhaustive.

The extrapolation mechanisms we use could relate to the size of the network (in km of track), the size of the train fleet, the number of passengers carried, the number of employees in the maintenance sectors, etc. The keys will be clearly outlined.

4.1.3.4 Projections

Taking into account our knowledge of the current situation and the actions currently underway in industry and research, we will attempt to establish one or more possible trajectories by evaluating the evolution of the KPIs for each.
The time horizon targeted here is relatively limited (around 5 years). In addition to assessing the indicators, the conditions for success and the obstacles to implementation will need to be identified.

Attempting to establish the future is always a perilous undertaking. While we cannot be certain about the course of events that will be proposed, we will endeavour to establish a highly probable future, at least given the state of our knowledge.

Figure 3 shows the process not in terms of its temporal organization, but rather in terms of information flows.

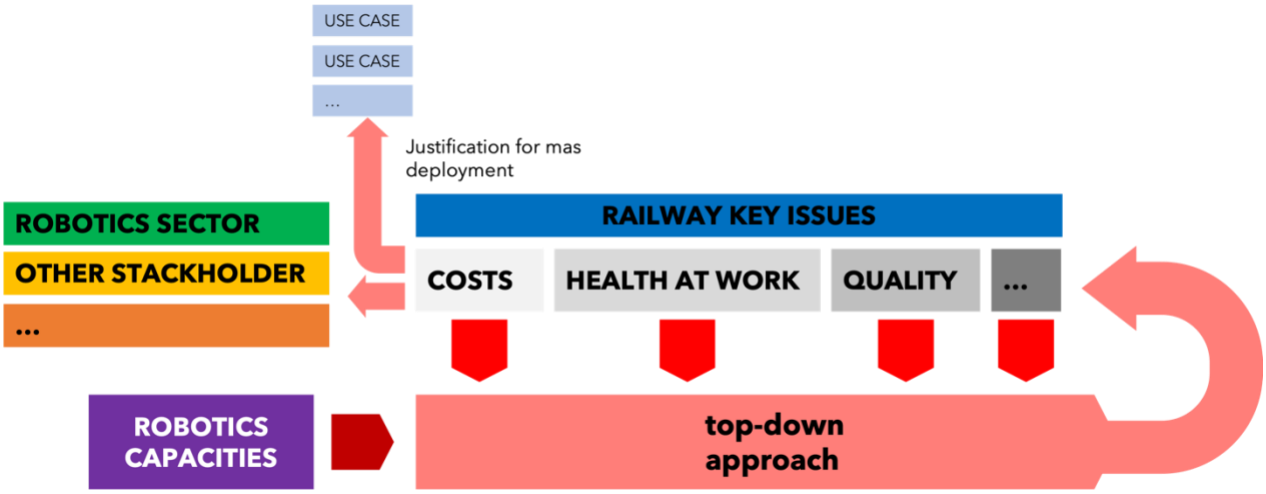

Figure 12 - Impact diagram

4.1.4 Top-down fictional approach

We propose a second, less analytical but more forward-looking approach. For this, we draw on the "Red Team Defense" approach proposed by PSL to the French armed forces.

More precision can be find on the web site of the project[47] and in their publication[48].

The Red Team Defense is composed of science fiction authors and scriptwriters. It's only a part of the group, but an emblematic one. The other parts are composed by researchers and military experts (this last part is sometime cold "Blue Team").

For the initiators, there are 3 essential components to this approach:

- the frontier object: this is a fictional story developed by the Red Team, but not an end in itself. It serves to link two universes which, a priori, don't talk to each other. It doesn't matter whether the future described comes true or not. What's interesting are the new thinking mechanisms it will induce.
- Back and forth: the approach is nourished by regular exchanges between the 3 parts of the collective mentioned above.
- Illustration: the website illustrates this very well. It's not so much to help tell the stories as to question the ideas put forward and make their presuppositions visible. This is probably the aspect that will be least accessible to us for budgetary constraints.

The porject does not propose to reproduce this approach identically. It doesn't have the resources to finance the work of a collective of authors and writers over a full year. We can take advantage of the most creative and imaginative minds among the work package partners. We can bring in the scientific component directly, and draw on our colleagues' professional skills.

We could therefore establish fictional scenarios for the year 2024. Scenarii can be different for rolling stock and for infrastructure or they can be the same. A scenario describes an evolution over time, not a fixed situation in a more or less distant future (up to 2050-2060). This evolutionary character is important. It is the way in which robotics adapts to evolution that needs to be described. It's important to document the obstacles, the reasons why this future is possible, and the values that are generated.

This approach was tested at the end of September 2023 with a group of SNCF scientific and technical experts, with promising results.

5 Selected approach

The work-package partners have chosen to implement an approach that includes top-down analytical and fictional approaches.
In addition to the rationality of the analytical approach, which is reassuring for our organisations, we will be able to adopt a more forward-looking approach, capable of giving rise to a shared

[47] https://redteamdefense.org/en/home

[48] Ces guerres qui nous attendent – EAN : 9782382841792 – Éditions des équateurs

imagination. Imagination alone is not enough to create business relationships. But it can be a formidable unifying force for a new or existing community.

The main stages of the process will be as follows:

- Breakdown of the maintenance process (analytical approach)
  - Input: breakdown coming from the NF X60-000
  - Inspiration: listed standards
  - Procedure: 1 or 2 online workshop
  - End of Phase: January 2024
  - Output: finalized breakdown
- KPI (analytical approach)
  - Input: finalized breakdown
  - Inspiration: listed standards, WP18 use cases KPI, list of the high level targets
  - Procedure: 2 or 3 online or physical workshop
  - End of Phase: April 2024
  - Output:
    - List of the unaffected subprocesses
    - KPI shot list for the impacted subprocesses
- Reference levels (analytical approach)
  - Input: KPI list
  - Procedure: internal work for each involved partner + 1 intermediate online meeting + 2 finalization meeting
  - End of Phase: October 2024
  - Output:
    - Refence levels for all the KPI
- Projection (analytical approach)
  - Input: KPI list and their reference levels
  - Procedure: 1 physical meeting to launch the work, 3 intermediate online meeting and 1 physical meeting to close the sequence
  - End of phase: February-march 2025
  - Output: evaluated scenarios
- Fictional approach
  - It will be an iterative process. The frontier object will be established simultaneously with the technological elements that respond to the new situations that emerge.
  - At this stage we can anticipate 4 iterations, which we will carry out as soon as the reference levels for the analytical approach have been established.
  - End of the phase: ideally synchronized with the end of the analytical phase, but a delay of around one trimester is possible.

6 Conclusion

The approach adopted is a mix between a top-down analytical approach and a top-down fictional approach.
The analytical approach is based on a breakdown of maintenance into more basic processes. For each of the elementary processes, a short list of relevant indicators (in the context of the introduction of robotics) is proposed. Reference levels are determined. The last step consists of evaluating the evolution of these indicators on a scale of approximately 5 years.
The fictional approach is inspired by Red Team Defense offered by Paris Sciences & Lettres to the French armies. They propose, over a longer time horizon, futures for which the probability of occurrence is not the key point. It is the reactions to be implemented in the face of these new situations that have important value. creating a collective imagination in addition to more traditional commercial relationships can also be a strong glue in a new-born ecosystem.
The total duration of the selected approach is 18 months, based on 4 stages for the analytical approach and on an iterative work of 6 to 9 months for the fictional part.

7 References

NF X60-000 Industrial maintenance - Maintenance function
NF EN 13306 Maintenance - Maintenance terminology
NF EN 16646 Maintenance - Maintenance within physical asset management
ISO 55001 Asset management - Management systems - Requirements
ISO 9001 Quality management systems - Requirements

8 Appendices (all parts included)

All the appendix are available aside this document in the ZIP file.

Appendix A

See file Appendix_A_WP18_BANNER.pdf

Appendix B

See file Appendix_B_Selection_Criteria_Analysis_v01.xls

Appendix C
See file “Appendix_C.pdf”

Appendix D
See file “Appendix_D_Template_SafetyPlan_IAM4RAIL.docx”

Appendix E
See file “Appendix_E.pdf”

Appendix F – Draft version of the System Definition template – ARGO example
See file "Appendix_F_System_Definition_Template_-_ARGO_example.pdf".