ROBOMINERS

ROBOMINERS Deliverable D4.1

SOFTWARE ARCHITECTURE
Revised Version

Summary:
The work in T4.1 has produced a SysML model that captures the initial ideas on the ROBOMINERS system architecture. The model maps the miner operational requirements into a set of structural and behavioural elements that realize the system vision. This document reflects this mapping and its realization as a software architecture for the robot controller.

This is a revised version of the deliverable addressing the modifications requested by the review team in the first review report.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement nº 820971.

Title:	Software Architecture
Lead beneficiary:	UPM
Other beneficiaries:	TAU, RCI, TALTECH, RBINS
Due date:	M12
Nature:	Public
Diffusion:	all Partners
Status:	Working Document
Document code:	

Revision history	Author	Delivery date	Summary of changes and comments
Version 01	RS	2020/07/13	Initial version of the document / model
Version 02	RS	2020/08/16	Initial release of the deliverable
Version 1	RS	2020/08/18	Added executive summary
Version 2	RS,EA	2021/06/28	Adapted to review report requirements

Approval status				
	Name	Function	Date	Signature
Deliverable responsible	Ricardo Sanz / UPM	Author	1/7/2021	[illegible]
WP leader	Ricardo Sanz / UPM	WPL	1/7/2021	[illegible]
Reviewer	Stephen Henley / RCI	QC	1/7/2021	[illegible]
Reviewer	Jussi Aaltonen / TAU	QC	2/7/2021	[illegible]
Project Coordinator	Claudio Rossi / UPM	PM	2/7/2021	[illegible]

This document reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

Diffusion List		
Partner name	Name	e-mail
All partners	-	robominers_all@autolistas.upm.es

REVISION SUMMARY

This is version 2 of Deliverable 4.1 that addresses the changes requested in the 1st review report.

Revision Request

"Please elaborate in more detail on the text and diagrams on pages 14 through 78, in particular providing rationale for including the in this report, as they seem not to be relevant to the DoA. The DoA required software interfaces to be developed. Moreover, in Task 4.1 a robot middleware was to be developed. Please elaborate on the status of work in this regard. The scope of the provided functional architecture (pages 111-122) seems very limited. If these are just examples, state this explicitly. Please elaborate of the overall status of work in this area."

Revision Scope in relation with Revision Requests (review report text in red)

"Elaborate in more detail on the text and diagrams on pages 14 through 78, in particular providing rationale for including the in this report, as they seem not to be relevant to the DoA."

We have **simplified** the document removing content not relevant to the DoA. We have **added textual explanations** of its content throughout.

"The DoA required software interfaces to be developed. Moreover, in Task 4.1 a robot middleware was to be developed. Please elaborate on the status of work in this regard. "

The deliverable now describes the software **interfaces** for the robot control subsystems and details the **middleware** to be used.

"The scope of the provided functional architecture (pages 111-122) seems very limited. If these are just examples, state this explicitly. Please elaborate of the overall status of work in this area."

The description of the control system base and functional architecture are now **more focused and deepened** using a scenario analysis and function identification.

DoW Task/Deliverable description

Task 4.1 Robot software architecture (Lead: UPM; Part.: TalTech, TAU, RCI)

This task develops a robot software architecture and will be developed in coordination with Task 3.1 (definition of system requirements) and taking into account the outcomes of the relevant sub-tasks of WP2. Based on the requirements of the computational power, selected sensor modalities and locomotion mechanism, drivers are defined, the robot middleware is developed based on ROS middleware and a GUI is worked out for rapid and easy testing of various sensor configurations and navigation strategies. The architecture is intended to be cross-layer fault tolerant including a health map and diagnostics features. System/Software architecture will be described in SysML/UML (Systems Modeling Language / Unified Modeling Language), standardized visual modeling languages developed for specifying, visualizing, constructing, and documenting complex software systems. This is a key task whose primary function is to allow easy integration of all the software components (from perception to navigation and control) of the robot. The key activities are: a.) architecture framework development; b.) definition and implementation of software interfaces.

EXECUTIVE SUMMARY

Task 4.1 Robot software architecture [M6-M18] is part of **WP4 - Miner software architecture and control** [Months: 6-42]. The objective of this task is the development of a **robot software architecture** and the implementation of architectural software elements.

The robot software architecture shall address project needs (robots, mining capabilities) as identified in WP2 and WP3.The architecture shall be capable of controlling a modular, underground robot to perform mining operations in difficult conditions (e.g. high deep or submerged). The architecture is intended to be cross-layer fault tolerant including a health map and diagnostics features. Adaptation is a key issue in non-attended systems and hence for these robots.

The architecture primary function is to allow easy integration of all the software components (from perception to navigation and control) of the robot to provide the mining mission required capabilities. Functional software components will be developed in many other tasks. Architecture specification will provide the needed framework to make integration possible.

The system/software architecture is described in a system model using SysML (Systems Modeling Language), an standardized modeling language developed for specifying, visualizing, constructing, and documenting complex systems. This language is used to create and document an architectural model of the robot software following established architecture standards.

The model produced so far has been built using the Sparx Enterprise Architect tool. This document (project deliverable D4.1) is an accompanying documentation for this model to help visualising it without the need of the modelling tool.

The initial work in T4.1 [M6-M12] produced a first SysML model that describes the initial ideas on the ROBOMINERS system architecture. The model maps the requirements into a set of structural and behavioural elements that realize the system vision. This document describes the content of this model.

The model is quite complex because it addresses multidimensional aspects that affect the construction of the system in the context of the ROBOMINERS project:

- **Several domains**: The developments in the project/system shall address the needs of different stakeholders in four domains: underground robotics, selective mining, robot bioinspiration and system autonomy. This implies that the model shall capture aspects and provide views that address several contexts.

- **Several robots**: The robotic implications of the four contexts -power robotics, mining robots, bioinspired robots, adaptive robots- imply the construction of different real and simulated systems (system variations). This implies that the model shall address the issues both at a general cross-domain level and a concrete, implementation-oriented level.

The document describes the content of the model: general **organization** and model **resources**; **project**-related elements and domains; system **requirements** from different viewpoints; system **realization**; and system **operation**.

TABLE OF CONTENTS

1. INTRODUCTION

The ROBOMINERS developments involve several teams in the construction of a complex system in the next years. In this process there are some issues that shall be managed to avoid late-minute problems:

- Integration: the different robot parts shall eventually be put together.
- Coverage: all the robot subsystems shall be addressed.
- Evolution: some of them will change along the project.

Organising complex systems construction is the domain of Systems Engineering (SE). For some developers, SE is mere bureaucracy; for others it is a lifesaver. SE is a disciplined way of building systems. In this sense, we included in the proposal the use of SysML/UML as a language for the definition of the control system architecture. SysML is the language advocated by the OMG[1] and INCOSE[2] for the SE of modern systems.

The ROBOMINERS team has decided to employ a *model-based system engineering (MBSE) strategy* to address the software/system developments in the project. Being this project a research-oriented project the role that the model plays is more oriented towards team communication than to automated software synthesis. However, we consider that having a stable, consortium-wide, well documented view of the system pays for the effort in building and maintaining such a model.

Hence the **ROBOMINERS System** is described by means of a semi-formal model that employs the standardized SysML systems modeling language (OMG, 2019). This deliverable captures a snapshot of **ROBOMINERS Software Architecture**. The complete model will evolve during project execution, being a major asset for the different systems engineering activities involved in this project.

1.1. About the Deliverable D4.1

This deliverable (D4.1) describes the software architecture according to initial systems engineering activities (Walden, Roedler, Forsberg, & Hamelin, 2015). The project uses an MBSE approach and a SysML system model is used to organise the system engineering workflow. In case of discrepancy, the model has the authority, not this document.

In principle, the content of the deliverable D4.1 as stated in the project Technical Annex (ROBOMINERS, 2019) was restricted to the software subsystem but it was decided to enlarge it to include not only software but reference to other system elements including hardware (i.e. also the physical robot and supporting subsystems). This enlargement was considered necessary for two reasons:

1. The software part and its architecture is of difficult understandability unless set in an adequate interpretation context (the robot system it controls).
2. Having a more complete model will help other activities in the project (i.e. in other WPs).

The focus of the document is the RoboMiner system software. Figure 1 describes the RoboMiner System **System Context**. The System Context shows the elements in the environment where the RM System performs its activities and that interact with the system:

[1] OMG: Object Management Group (www.omg.org).
[2] INCOSE: International Council on Systems Engineering (www.incose.org)

- The robot mines the **Mine** and delivers slurry to a **Slurry Processor.**
- **Power** and **Water sources** are necessary to support robot operation.
- The whole system is situated in a larger mining ecosystem that determines operational conditions for the selective mining operations of the robot.

Figure 1 content is a SysML Block Definition Diagram (BDD). Tutorial materials on SysML are broadly available. The specification itself is a public document that can be downloaded from:

https://sysml.org/.res/docs/specs/OMGSysML-v1.6-19-11-01.pdf

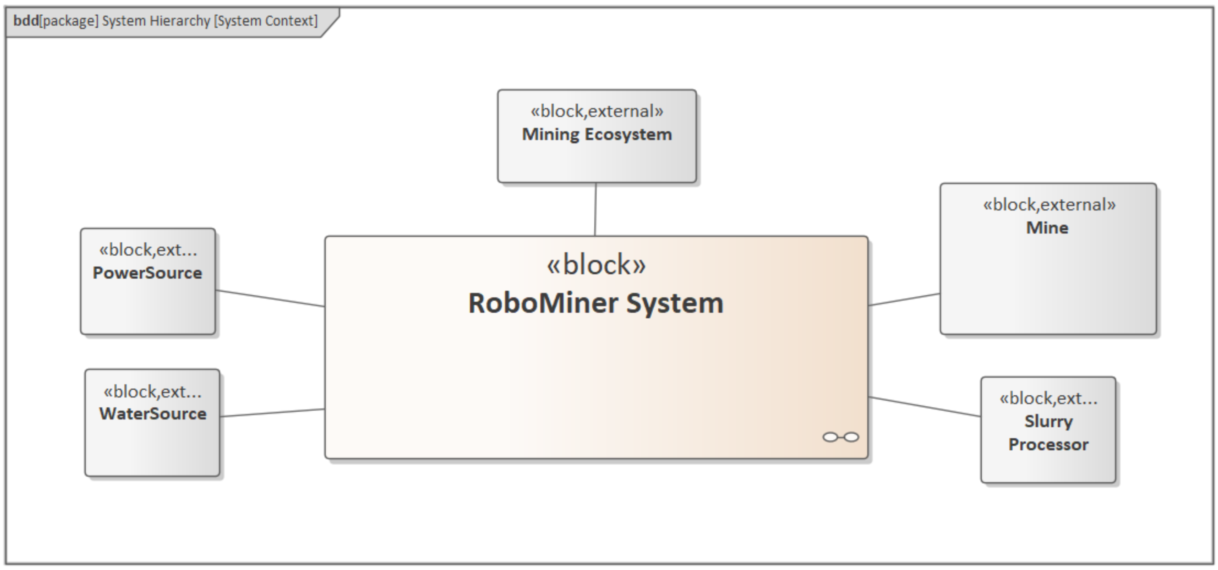


Figure 1: RoboMiner System System Context.

As a brief intro to SysML BDDs, the System Context BDD of Figure 1 shows the following elements:

- A header box: that says this is a BDD that describes the System Context associated to the System Hierarchy package.
- A frame: an external box that sets the context of analysis (the System Hierarchy in this diagram).
- Some blocks: That are labelled with the stereotype <<block>> to indicate that they are systems, subsystems and entities of relevance, and the stereotype <<external>> to indicate that they are not in the scope of the RoboMiner System.
- Some links: That reflect the existence of a relation between blocks.

The diagram describes the entities involved (blocks) that include the RoboMiner system and the surrounding elements that interact with it (in particular the mine, supporting elements like water and power sources, and the entities that use the extracted materials, the slurry processor and the ecosystem).

1.2. Acronyms

These are some acronyms that are used in this document:

BDD:	Block Definition Diagram
MBSE:	Model-based Systems Engineering.

OMG:	Object Management Group (www.omg.org).
QUDV:	Quantities, Units, Dimensions and Values. A predefined library with model elements to represent physical aspects of systems. The QUDV library is introduced in the *OMG SysML Specification Annex E*.
RM:	RoboMiners.
SE:	Systems Engineering.
SysML:	OMG Systems Modeling Language.

1.3. Mandatory References

The model described in this deliverable follows the **SysML 1.6 specification** (OMG, 2019):

- OMG Systems Modeling Language (OMG SysML™) Version 1.6. Object Management Group. 2019.

The architectural modelling approach follows the **ISO standard on architecture description** (ISO, 2011):

- ISO/IEC/IEEE 42010-2011 – Systems and software engineering – Architecture description. International Standards Organization. 2011.

1.4. A Note on Document Content

The ROBOMINERS model is organized as a series of packages that contain representations of the system under development considering the different phases of the project and the different viewpoints involved.

The initial sections of this document contain information about the Model and Project View packages. These packages contain reference material to understand and use the model.

The System Model is captured in four main packages:

- User and System Requirements.
- The Mining Environment that contains the system.
- The System Realization package that describes the construction of the system.
- The System Operational package that describes the operation and use of the system.

The concrete design aspects that are central to the system architecture are contained in the system realization section of the model. This realization of the system is described in five packages:

- The System Hierarchy that describes the structural elements and relations in the system.
- The System Behavior that describes behavioral aspects of the system.
- The System Control package that describes the software and control mechanisms.
- The **System Instances** package that describes the specific robots that are going to be built in the project.

Finally, the **Module Library** contains elements that are used transversally in different subsystems.

Given the early status of the model describe in this deliverable, the most relevant content can be found in the Requirements and description of the System Hierarchy.

Beware that the project addresses different objectives and that there will be different implementations of systems to address the needs of the project (esp. robots and simulations).

This implies that there are architectural descriptions that are general -apply to different robots- and other that are specific of concrete ones. In particular:

- The RM1 robot is the full-size robot that will be able to perform real mining.
- The RM2 robot is a modular, small scale system that is used to explore robot reconfiguration strategies.

The following sections reflect the structure of the Sparx Enterprise Architect system model. Their content is automatically generated from the model itself using the model exporting mechanisms of the tool.

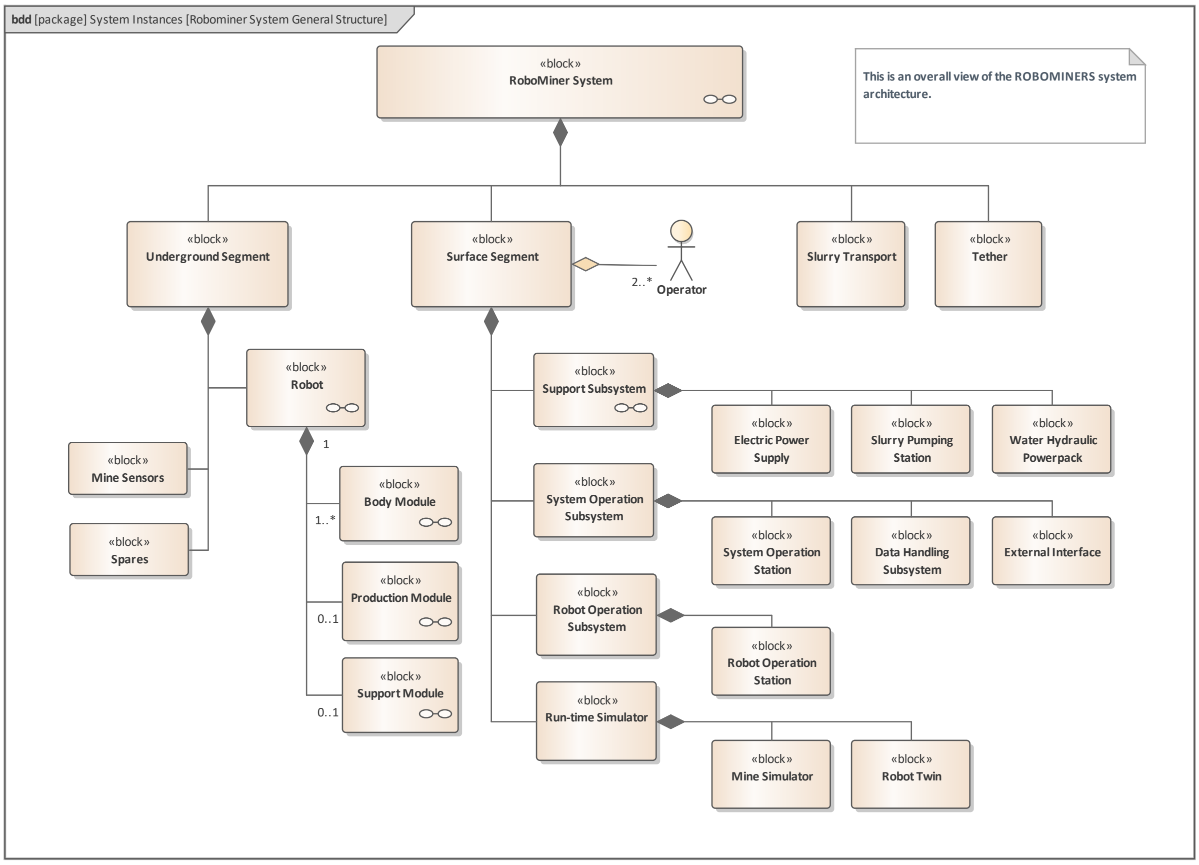

bdd [package] System Instances [Robominer System General Structure]
«block» RoboMiner System
This is an overall view of the ROBOMINERS system architecture.
«block» Underground Segment
«block» Surface Segment
2..*
Operator
«block» Slurry Transport
«block» Tether
«block» Robot
1
«block» Mine Sensors
«block» Spares
1..*
«block» Body Module
0..1
«block» Production Module
0..1
«block» Support Module
«block» Support Subsystem
«block» Electric Power Supply
«block» Slurry Pumping Station
«block» Water Hydraulic Powerpack
«block» System Operation Subsystem
«block» System Operation Station
«block» Data Handling Subsystem
«block» External Interface
«block» Robot Operation Subsystem
«block» Robot Operation Station
«block» Run-time Simulator
«block» Mine Simulator
«block» Robot Twin

Figure 2: Robominer System General Structure.

2. MODEL ORGANIZATION, DOMAINS AND PROFILES

The system model described in this deliverable contains the artefacts associated with the model-based systems engineering approach used in this project. These include use cases, scenarios, requirements, structural diagrams, behavioural diagrams and cross-cutting aspects.

The model is complex due to the different objectives addressed by the ROBOMINERS project and the complexities of robot control in underground harsh conditions.

This chapter describes some common, transversal elements, that are used to organise and label model elements to manage this complexity.

2.1. Model Structure and Dependencies

The ROBOMINERS project is organized as a series of packages (see Figure 3) that contain representations of the system under development considering the different phases of the project and the different viewpoints involved. The About the Model and Project View packages contain reference material to understand and use the model.

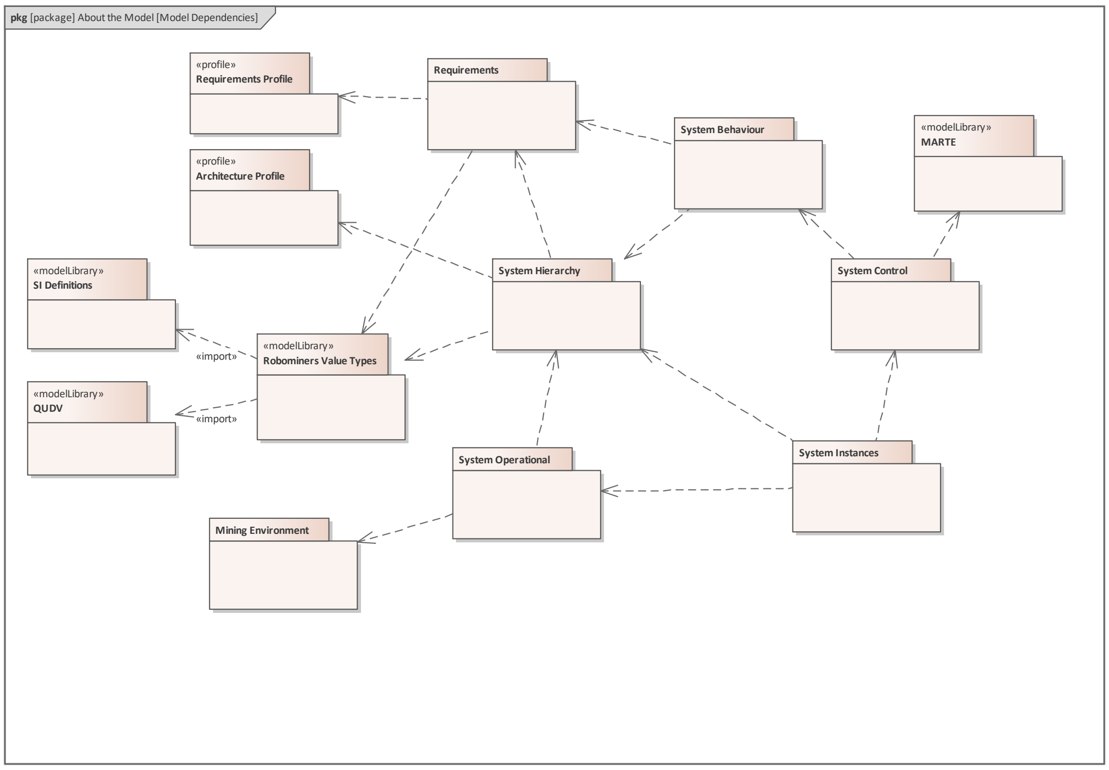

pkg [package] About the Model [Model Dependencies]
«profile» Requirements Profile
Requirements
System Behaviour
«modelLibrary» MARTE
«profile» Architecture Profile
«modelLibrary» SI Definitions
System Hierarchy
System Control
«import»
«modelLibrary» Robominers Value Types
«modelLibrary» QUDV
«import»
System Operational
System Instances
Mining Environment

Figure 3: Model Packages and Dependencies.

The System Model is captured in four packages:

- User and System **Requirements.**
- The **Mining Environment** that contains the system.
- The **System Realization** package that describes the construction of the system.
- The **System Operational** package that describes the operation and use of the system.

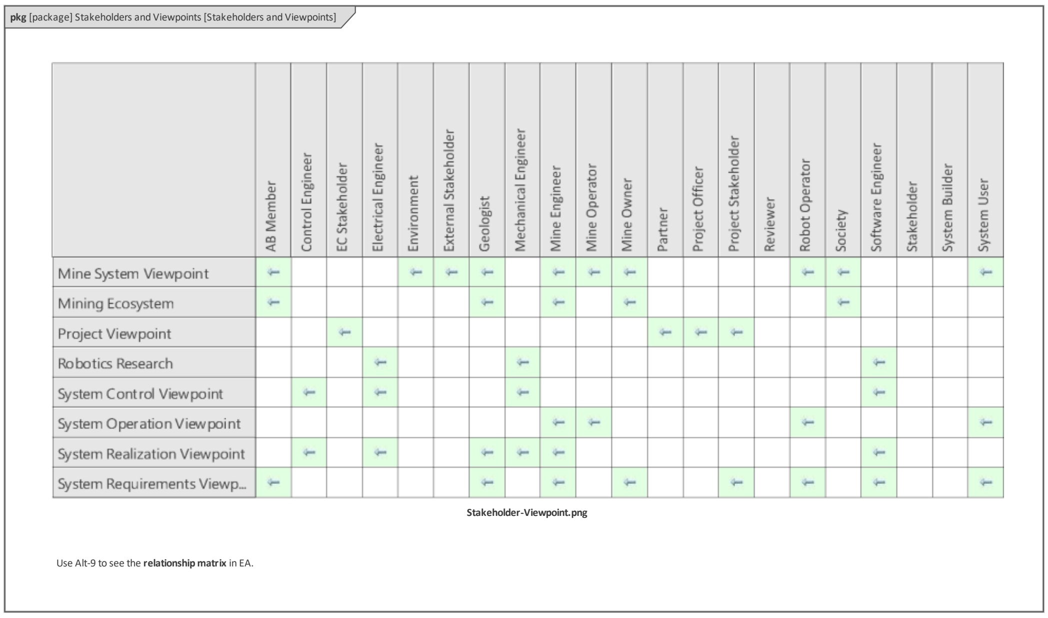
The realization of the system is described in five packages:

- The **System Hierarchy** that describes the structural elements and relations in the system.
- The **System Behavior** that describes behavioral aspects of the system.
- The **System Control** package that describes the software and control mechanisms.
- The **System Instances** package that describes the specific robots that are built.
- Finally, the **Module Library** contains elements that are used transversally in different subsystems.

2.2. Stakeholders and Viewpoints

The ROBOMINERS System Model shall address the needs of all project stakeholders identified in the requirements specification process. This identification of stakeholders is important to present the system architecture model in ways that are adequate for the different stakeholders (ISO, 2011). These stakeholders are used to define different viewpoints that are used to organize the model.

The following table shows the Stakeholders and Viewpoints relationship matrix.



pkg [package] Stakeholders and Viewpoints [Stakeholders and Viewpoints]

	AB Member	Control Engineer	EC Stakeholder	Electrical Engineer	Environment	External Stakeholder	Geologist	Mechanical Engineer	Mine Engineer	Mine Operator	Mine Owner	Partner	Project Officer	Project Stakeholder	Reviewer	Robot Operator	Society	Software Engineer	Stakeholder	System Builder	System User
Mine System Viewpoint	←				←	←	←		←	←	←					←	←				←
Mining Ecosystem	←						←		←		←						←				
Project Viewpoint			←									←	←	←							
Robotics Research				←				←										←			
System Control Viewpoint		←		←				←										←			
System Operation Viewpoint									←	←						←					←
System Realization Viewpoint		←		←			←	←	←									←			
System Requirements Viewp...	←						←		←		←			←		←		←			←

Stakeholder-Viewpoint.png

Use Alt-9 to see the **relationship matrix** in EA.

Figure 4: Stakeholders and Viewpoints.

System stakeholders are grouped into five categories:

- **EC Stakeholders** are stakeholders associated to the EC.
- **Project stakeholders** include the main stakeholders concerned about the execution of the project.
- **System builders** are stakeholders involved in the construction of the robots and mining systems in the project.
- **System users** are stakeholders involved in the operation and exploitation of the system
- **External stakeholders** are external entities that contribute general requirements associated to the potential impact of the technology.

2.3. Common Model Resources

There are some common resources that are used throughout the model:

- **Model libraries**. In the present status of the model, the only library is the QUVD standard library used to capture quantitative aspects in the model (physical magnitudes, quantities, unities, etc.)

- **SysML profiles**. These are collections of specific resources –metamodel elements- that help better capturing concrete aspects of the system model. We have developed two profiles for the project: 1) a **requirements profile** to properly capture the different classes of requirements we have and a 2) an architecture profile to label the core architectural elements of the system (agents, nodes and functions).

2.3.1. QUDV Library

Figure 5 shows a SysML Block Definition diagram in package 'QUDV' that is used to avoid misunderstanding concerning the interpretation of numeric values in the system/model elements.

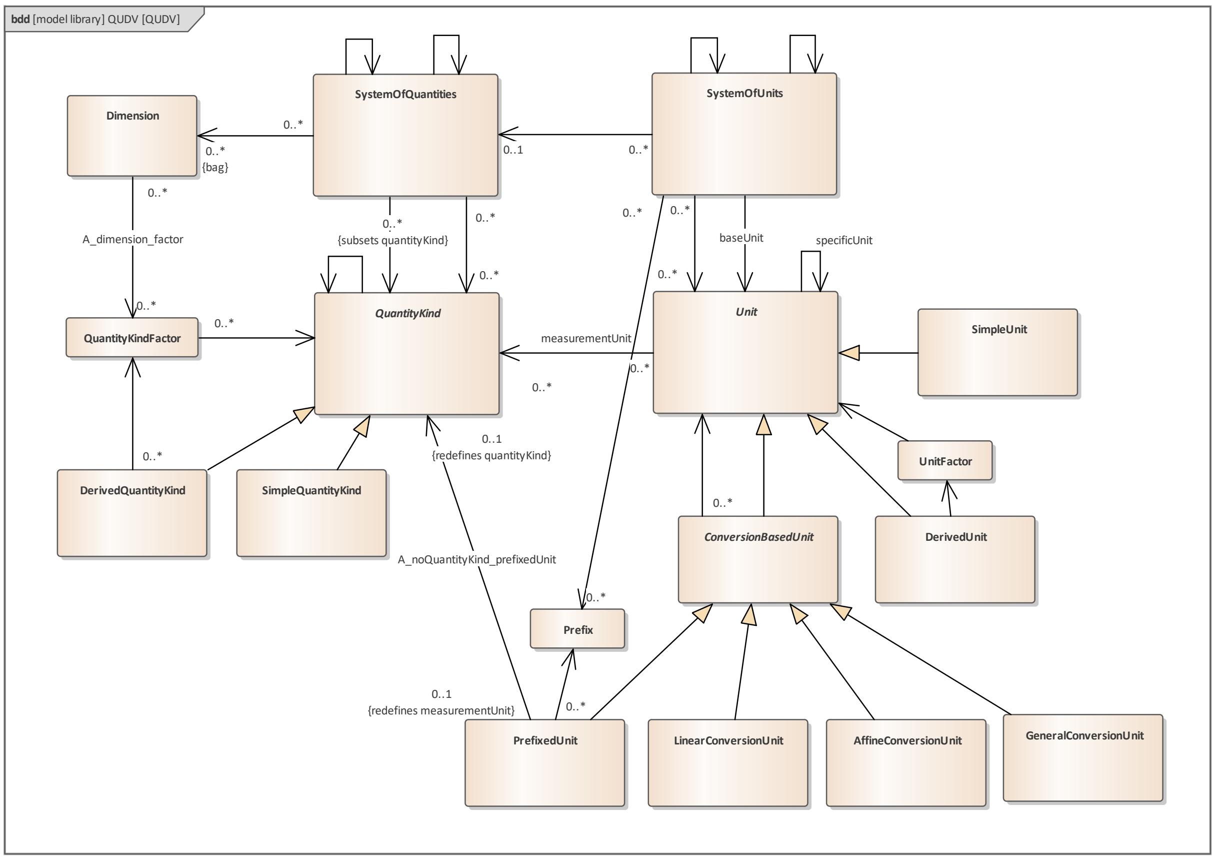


Figure 5: QUDV Library for handling magnitudes and quantities.

The QUVD library is defined in the SysML specification (OMG, 2019) and is used to capture quantitative aspects in the model (physical magnitudes, quantities, unities, etc.). For any system model, a solid foundation of well-defined quantities, units and dimensions system is very important to reduce risks. Properties that describe many aspects of a system depend on it. Such a foundation is a team-wide shareable resource that can be used by all partners in a consistent way.

2.3.2. Architecture Profile

The specification of the Robominer software architecture implies the identification of fundamental elements and their relations. After the specification of the Robot Operating System v2 (ROS2) as base middleware for the system implementation it is necessary to build on it the transversal capabilities that system architectural element will have.

The **architecture profile** is a shared resource that provides stereotypes for the different classes of system core elements (esp. system agents, system nodes and functions).

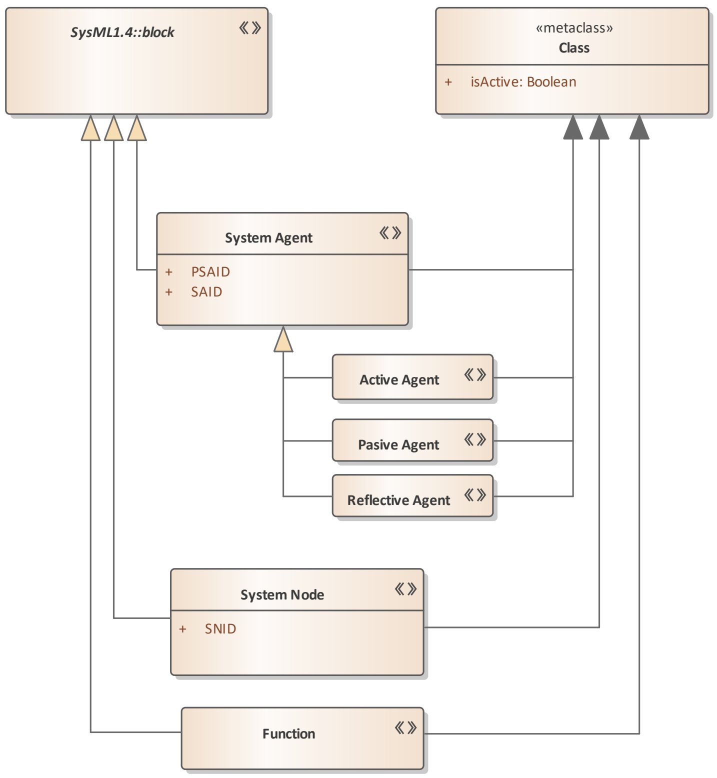


Figure 6: System architecture profile.

These are the basic constructive elements of the software architecture that are used to provide homogeneous capabilities across the model (note that the model is built and used by different partners). These elements are:

- System Agent: The Robominer is a modular robot and as such it requires some level of module-bound intelligence. The system agent provides this capability of *"being part of"* a Robominer aggregate. Some of these agents will be passive, some active, and some will include reflective capability.

- System Node: While the middleware selected for the Robominer software is the version 2 of the Robot Operating System (ROS2) the node concept of this middleware does not fully address the needs of the project (esp. concerning the metalevel to be developed in T4.5). A more concrete and formal specification was in need to properly address the reasoning processes required by the metacontroller.

- **Function**: Is the core element used to organize system capability and realization. This is fundamental for the metacontrol aspect of the system.

2.3.3. Requirements Profile

The requirements profile provides stereotypes for requirements in the context of ROBOMINERS. This is useful given the different nature of some of the requirements coming from the different project stakeholders.

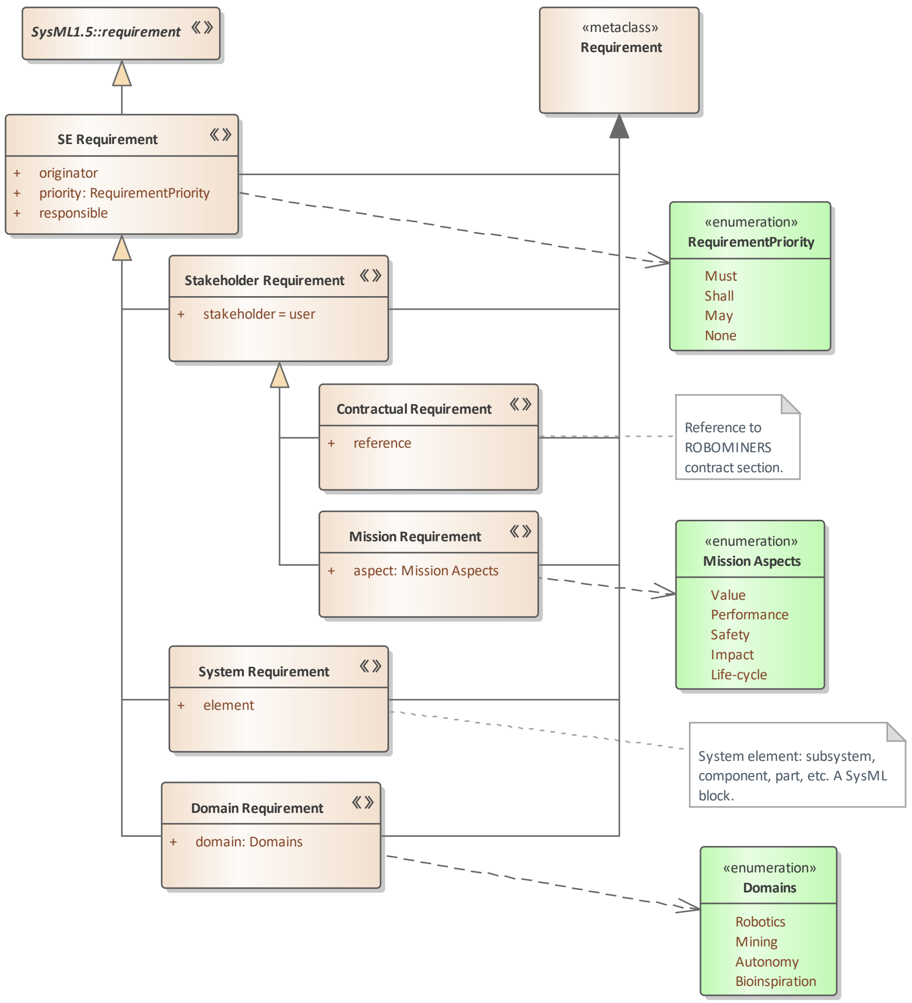


Figure 7: Requirements Profile**.**

The central content of the requirements profile is the identification of both the sources and natures of requirements –stakeholders, domain and system- and the specific domains that address the different RTD foci of the project.

2.4. Project Domains

The ROBOMINERS Project sits at a convergence point of several domains of research and technological development:

- Special mineral deposits mining using selective strategies.

- Underground bioinspired robotics.
- Autonomy and resilience by system adaptation.

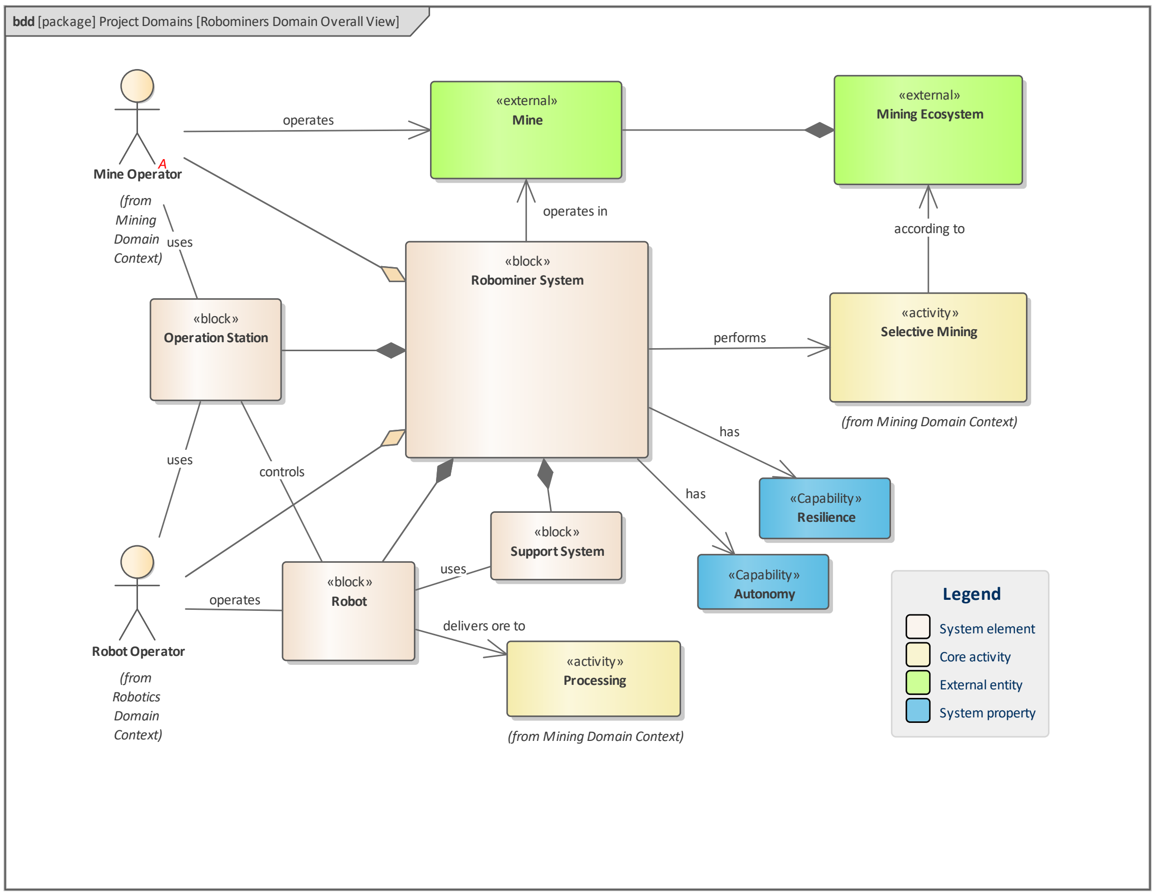


Figure 8: Robominers Domain Overall View

These domains define contexts for understanding the issues related to the construction of the robotized system. They establish different frameworks, business needs, system requirements, terminology and methods that shall be harmonised is the pursue of a coherent system and strategy.

2.4.1. Mining Domain Context diagram

Figure 9 shows the Mining Domain Context diagram. This diagram shows the main elements that surround the Robominer system from the prespective of mining operations.

The Robominer operates in a mine under the supervision of a human operator. The robot may interact with other elements in the mine also under supervision of human operators.

The mine contains some orebody that contains the mineral ore of interest. The robot performs excavation to extract this mineral ore using a selective mining strategy. The ore may suffer other processes after extraction.

The selective mining activity addresses the needs of the mining ecosystem and the characteristics of the orebody like veins, joints, etc.

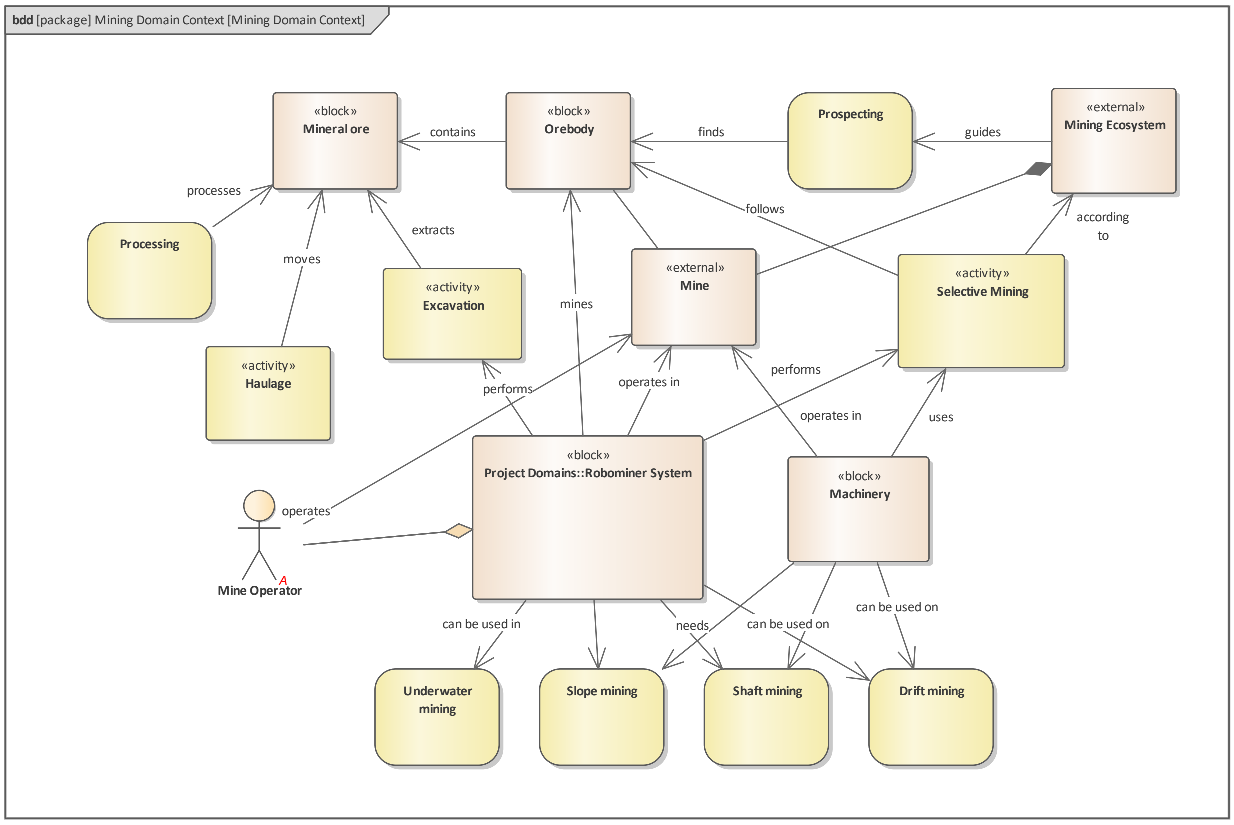


Figure 9: SysML Block Definition diagram in package 'Mining Domain Context'.

The mining excavation operation may be performed in a dry frontal process but it may also have special needs —i.e. special mining activities— for underwater, slope, shaft or drift mining.

The mine may also be actively operated by means of other specialised machinery that the robot system may have a need to know about.

2.4.2. Robotics Domain Context diagram

The selective mining activity is performed by the Robominer System. The robot miner is one of the elements that compose the system. The robot is operated by a human through an operator station. The mine is also operated by a human operator.

The robot has some structural elements —body— and behavioral elements —controller— to perform the system activities. The main activities considered so far are related to robot movement and localization.

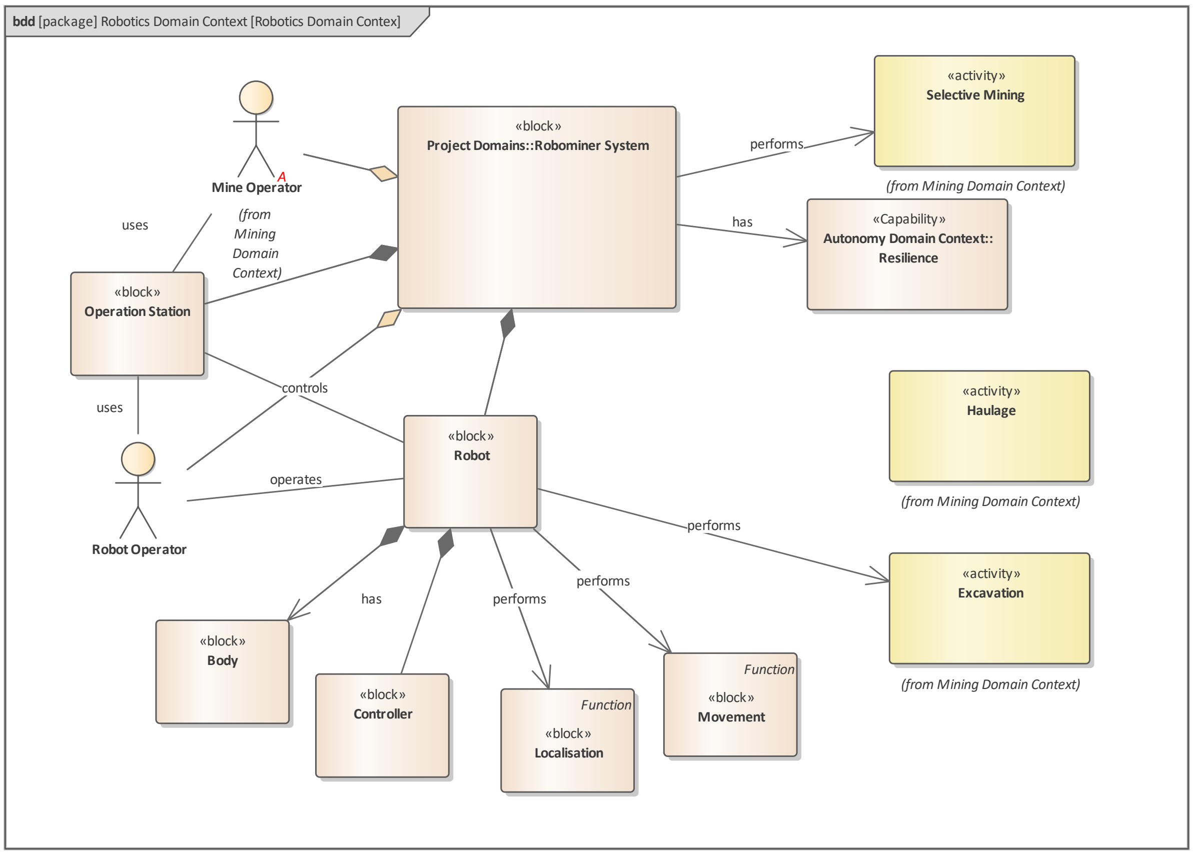

bdd [package] Robotics Domain Context [Robotics Domain Contex]
Mine Operator
(from Mining Domain Context)
«block» Project Domains::Robominer System
performs
«activity» Selective Mining
(from Mining Domain Context)
has
«Capability» Autonomy Domain Context:: Resilience
uses
«block» Operation Station
controls
uses
«block» Robot
«activity» Haulage
(from Mining Domain Context)
operates
Robot Operator
performs
«activity» Excavation
(from Mining Domain Context)
has
performs
performs
«block» Body
«block» Controller
Function «block» Localisation
Function «block» Movement

Figure 10: SysML Block Definition diagram in package 'Robotics Domain Context'.

The excavation activity —maybe the most important one— is also performed by the robot and fundamental for the mining domain.

2.4.3. Autonomy Domain Context diagram

Figure 11 contains a SysML Block Definition diagram related to the autonomy domain context. In essence, this diagram describes that the autonomy-enhancement capability of adaptation is provided by means of two mechanisms:

- Awareness: used by the robot system to observe the functional status of the system.
- Reconfiguration: used by the metacontroller to reorganize the robot system to address disturbances.

These two mechanisms are the basis for the provide capabilities of autonomy, adaptation and resilience.

As an example, the diagram shows that the movement and localization capabilities described before are realized as system functions and managed by the self-awareness mechanisms of the robot.

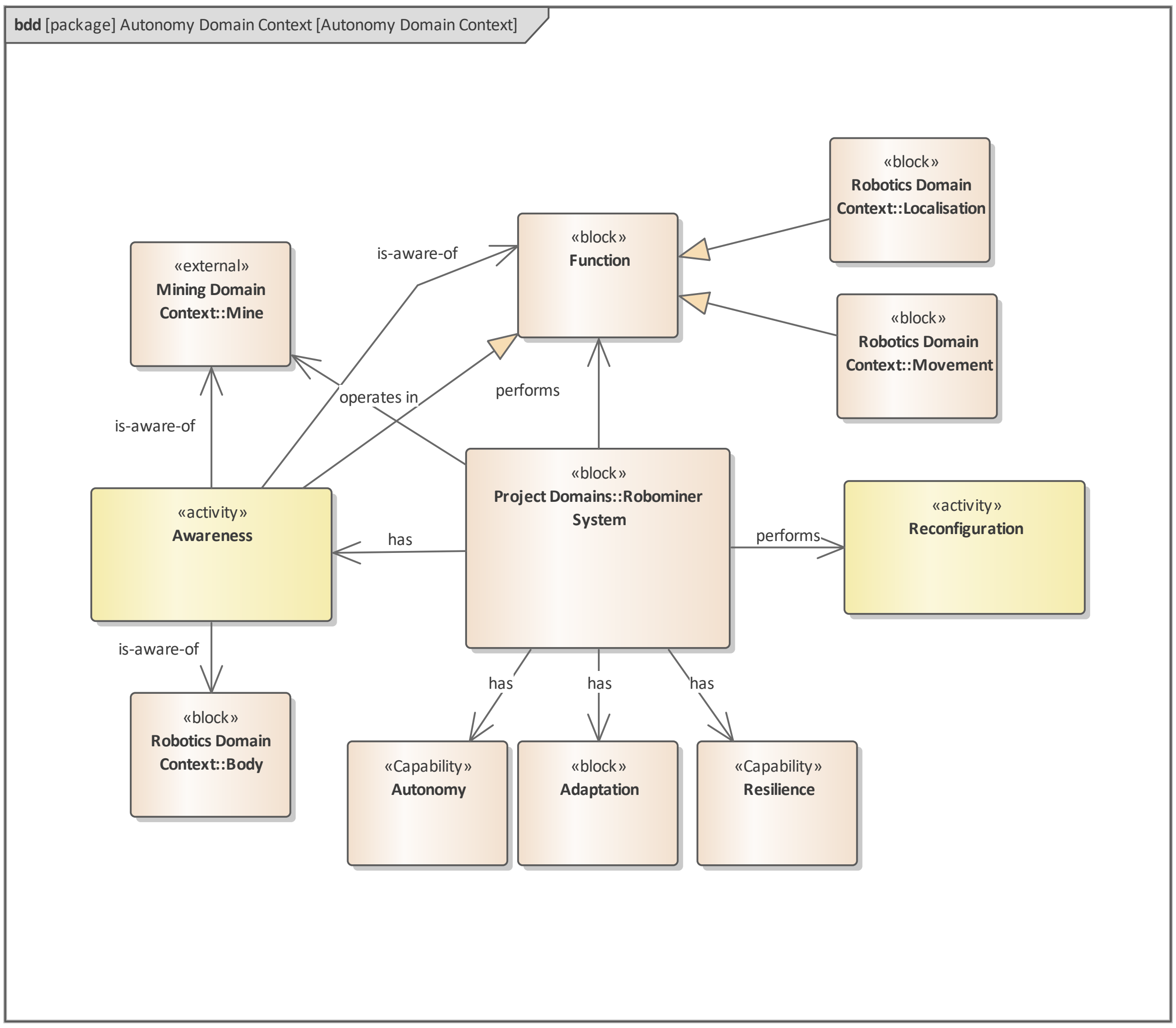

bdd [package] Autonomy Domain Context [Autonomy Domain Context]
«block» Robotics Domain Context::Localisation
«block» Function
«external» Mining Domain Context::Mine
is-aware-of
«block» Robotics Domain Context::Movement
operates in
performs
is-aware-of
«block» Project Domains::Robominer System
«activity» Awareness
has
performs
«activity» Reconfiguration
is-aware-of
has
has
has
«block» Robotics Domain Context::Body
«Capability» Autonomy
«block» Adaptation
«Capability» Resilience

Figure 11: SysML Block Definition diagram in package 'Autonomy Domain Context'.

3. REQUIREMENTS SPECIFICATION

Using the capabilities of the SysML language, the system model contains both requierements and cros-cutting aspects (e.g. allocations). In this section we summarise the main results of the work done in requirements specification and analysis,

Figure 12 shows the model packages that contain the identified requirements for the robominer system. These requirements are related to different sources :

- Contractual requirements: Those stated in the technical annex. Beware that due to the exploratory and TRL4 nature of the project some of these requirements are not mandatory for the full-scale miner robot implementation but are demonstrated in laboratoy prototypes or address considerations of longer horizon.
- User requirements: Related to the mining ecosystem and the use ot the robot for ore extraction.
- Operation requirements: Related to the behavior of the system during operation.
- Domain requirements: Related to the different RTD domains addressed in the project.
- System variations: In the ROBOMINERS project there will be several robot implementations that are used to address specific aspects of the project. This package contains requirements that are specific of any one of such implementations.

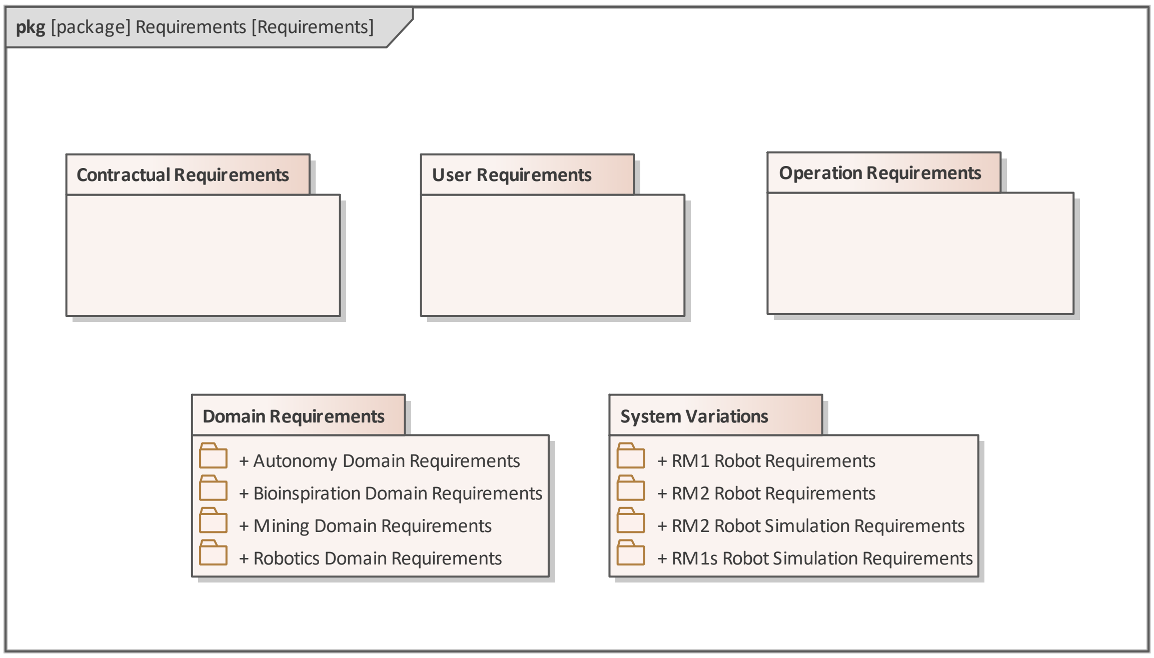

pkg [package] Requirements [Requirements]
Contractual Requirements
User Requirements
Operation Requirements
Domain Requirements
+ Autonomy Domain Requirements
+ Bioinspiration Domain Requirements
+ Mining Domain Requirements
+ Robotics Domain Requirements
System Variations
+ RM1 Robot Requirements
+ RM2 Robot Requirements
+ RM2 Robot Simulation Requirements
+ RM1s Robot Simulation Requirements

Figure 12: Requirements packages.

3.1. Use-case and Scenario-based approach

In order to identify the system requirements a use-case and scenario-based approach has been used.

Figure 13 shows the package structure used to organize the use cases that have been analyzed to identify requirements.

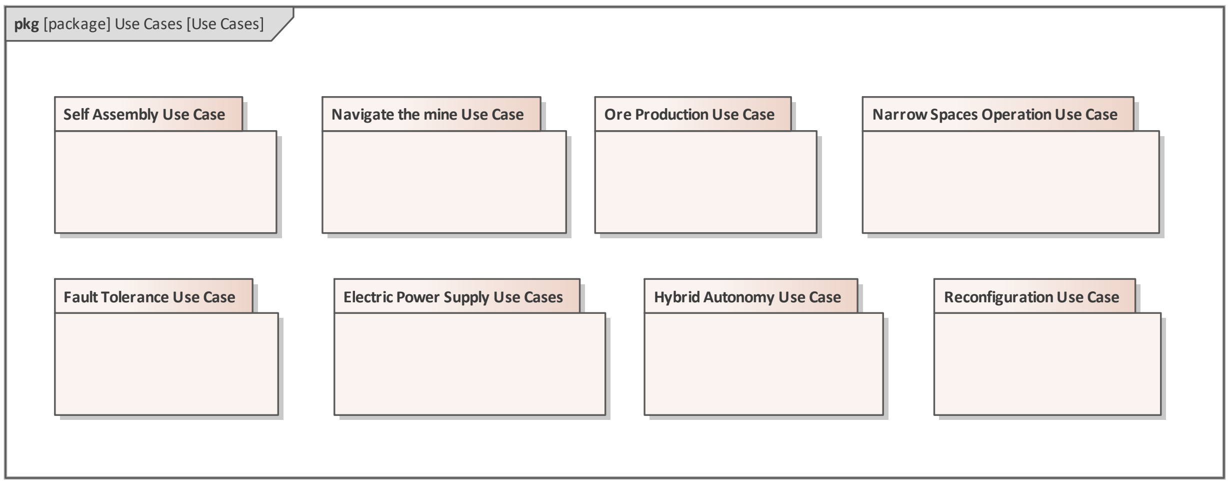

pkg [package] Use Cases [Use Cases]
Self Assembly Use Case
Navigate the mine Use Case
Ore Production Use Case
Narrow Spaces Operation Use Case
Fault Tolerance Use Case
Electric Power Supply Use Cases
Hybrid Autonomy Use Case
Reconfiguration Use Case

Figure 13: Use cases packages in the requirements identification process.

As an example, Figure 14 shows one concrete use case related to the self-assembly of modular robots. See the model to have a detailed view of all use cases.

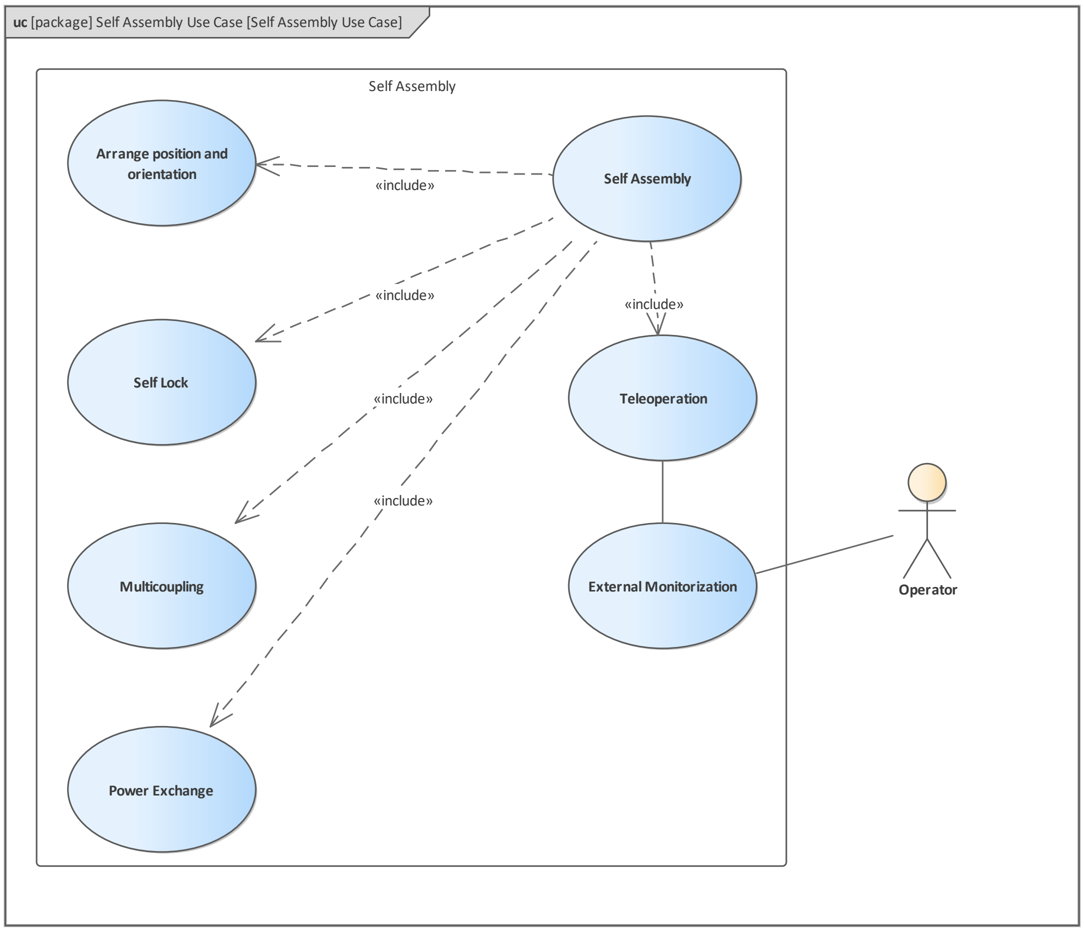


Figure 14: Use Case diagram in package 'Self Assembly Use Case'.

See the system model for details concerning the concrete requirements identified so far. See also Annex 1: Scenarios for a textual, more readable description of some concrete scenarios in these use cases that have been used to further identify system functions and define the system architecture.

3.2. System Variations Requirements

As said before, the ROBOMINERS Project will implement several robot systems (both real and simulated) to address the different needs of the project. These systems are system variations that derive from core models and assets. The requirements identified will be addressed in these system variants.

In the present situation, three + three variations (three robots, three simulations) have been identified:

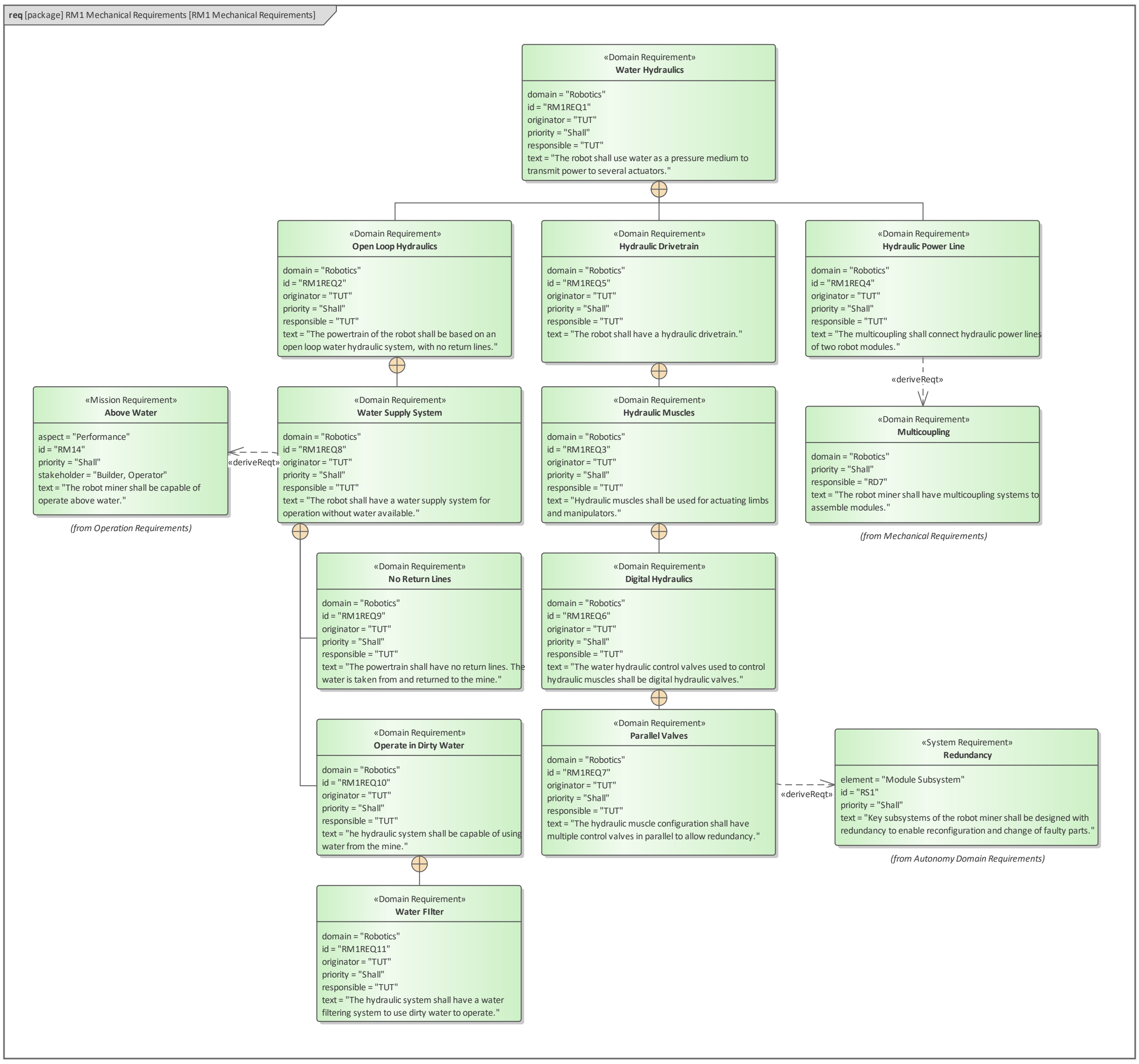
- RM1: a full-scale mining robot able to operate in underground flooded environments.
- RM2: a small-scale robot used to investigate reconfiguration strategies for augmented autonomy and resilience.
- RM3: a small-scale robot used to investigate movement and localization strategies.

These three robots will have three simulated counterparts (RMSx). The need for simulations respond to different classes of needs:

- Some capabilities will be demonstrated in simulation. For example, the RM1 physical reconfiguration may not be demonstrated in the full-scale robot due to the limited budget (RM1 parts are expensive elements).
- There is a need of having experimental platforms —e.g. for controller development— before the physical robot is built due to the concurrent activity in the different workpackages.

Figure 15 below, shows just one of the requirements diagrams available in the system model: the RM1 Mechanical Requirements for the full-scale robot.

Figure 16 shows some of the domain requirements for the RM2 robot. This is a small-scale robot that is to be used in the development of adaptation strategies by reconfiguration.


req [package] RM1 Mechanical Requirements [RM1 Mechanical Requirements]
«deriveReqt»
«deriveReqt»
«deriveReqt»
(from Operation Requirements)
(from Mechanical Requirements)
(from Autonomy Domain Requirements)

«Domain Requirement»
Water Hydraulics
domain = "Robotics"
id = "RM1REQ1"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The robot shall use water as a pressure medium to transmit power to several actuators."

«Domain Requirement»
Open Loop Hydraulics
domain = "Robotics"
id = "RM1REQ2"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The powertrain of the robot shall be based on an open loop water hydraulic system, with no return lines."

«Domain Requirement»
Hydraulic Drivetrain
domain = "Robotics"
id = "RM1REQ5"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The robot shall have a hydraulic drivetrain."

«Domain Requirement»
Hydraulic Power Line
domain = "Robotics"
id = "RM1REQ4"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The multicoupling shall connect hydraulic power lines of two robot modules."

«Mission Requirement»
Above Water
aspect = "Performance"
id = "RM14"
priority = "Shall"
stakeholder = "Builder, Operator"
text = "The robot miner shall be capable of operate above water."

«Domain Requirement»
Water Supply System
domain = "Robotics"
id = "RM1REQ8"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The robot shall have a water supply system for operation without water available."

«Domain Requirement»
Hydraulic Muscles
domain = "Robotics"
id = "RM1REQ3"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "Hydraulic muscles shall be used for actuating limbs and manipulators."

«Domain Requirement»
Multicoupling
domain = "Robotics"
priority = "Shall"
responsible = "RD7"
text = "The robot miner shall have multicoupling systems to assemble modules."

«Domain Requirement»
No Return Lines
domain = "Robotics"
id = "RM1REQ9"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The powertrain shall have no return lines. The water is taken from and returned to the mine."

«Domain Requirement»
Digital Hydraulics
domain = "Robotics"
id = "RM1REQ6"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The water hydraulic control valves used to control hydraulic muscles shall be digital hydraulic valves."

«Domain Requirement»
Operate in Dirty Water
domain = "Robotics"
id = "RM1REQ10"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "he hydraulic system shall be capable of using water from the mine."

«Domain Requirement»
Parallel Valves
domain = "Robotics"
id = "RM1REQ7"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The hydraulic muscle configuration shall have multiple control valves in parallel to allow redundancy."

«System Requirement»
Redundancy
element = "Module Subsystem"
id = "RS1"
priority = "Shall"
text = "Key subsystems of the robot miner shall be designed with redundancy to enable reconfiguration and change of faulty parts."

«Domain Requirement»
Water FIlter
domain = "Robotics"
id = "RM1REQ11"
originator = "TUT"
priority = "Shall"
responsible = "TUT"
text = "The hydraulic system shall have a water filtering system to use dirty water to operate."

Figure 15: SysML Requirements diagram in package 'RM1 Mechanical Requirements'.

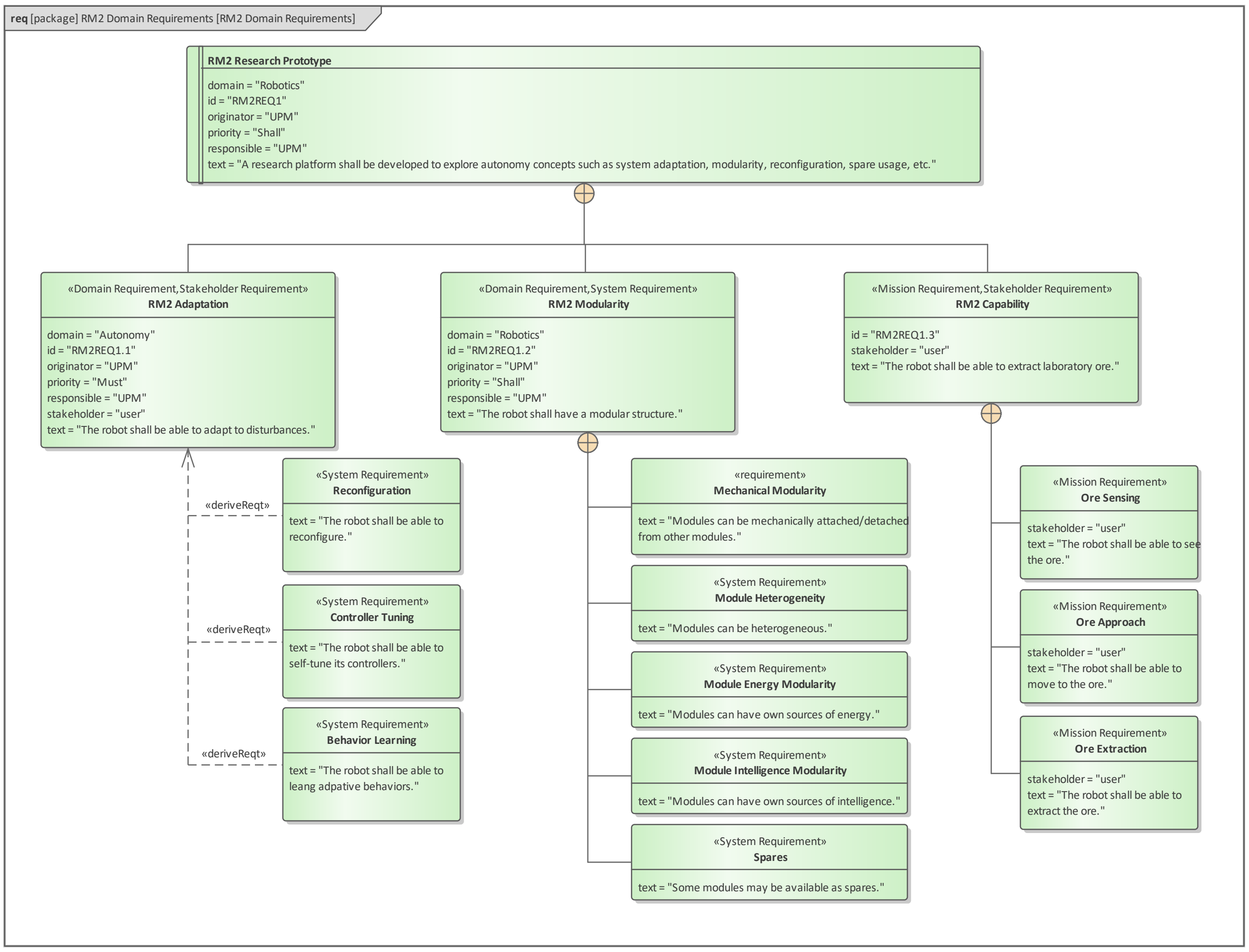

req [package] RM2 Domain Requirements [RM2 Domain Requirements]
RM2 Research Prototype
domain = "Robotics"
id = "RM2REQ1"
originator = "UPM"
priority = "Shall"
responsible = "UPM"
text = "A research platform shall be developed to explore autonomy concepts such as system adaptation, modularity, reconfiguration, spare usage, etc."
«Domain Requirement,Stakeholder Requirement» RM2 Adaptation
domain = "Autonomy"
id = "RM2REQ1.1"
originator = "UPM"
priority = "Must"
responsible = "UPM"
stakeholder = "user"
text = "The robot shall be able to adapt to disturbances."
«Domain Requirement,System Requirement» RM2 Modularity
domain = "Robotics"
id = "RM2REQ1.2"
originator = "UPM"
priority = "Shall"
responsible = "UPM"
text = "The robot shall have a modular structure."
«Mission Requirement,Stakeholder Requirement» RM2 Capability
id = "RM2REQ1.3"
stakeholder = "user"
text = "The robot shall be able to extract laboratory ore."
«deriveReqt»
«System Requirement» Reconfiguration
text = "The robot shall be able to reconfigure."
«deriveReqt»
«System Requirement» Controller Tuning
text = "The robot shall be able to self-tune its controllers."
«deriveReqt»
«System Requirement» Behavior Learning
text = "The robot shall be able to leang adpative behaviors."
«requirement» Mechanical Modularity
text = "Modules can be mechanically attached/detached from other modules."
«System Requirement» Module Heterogeneity
text = "Modules can be heterogeneous."
«System Requirement» Module Energy Modularity
text = "Modules can have own sources of energy."
«System Requirement» Module Intelligence Modularity
text = "Modules can have own sources of intelligence."
«System Requirement» Spares
text = "Some modules may be available as spares."
«Mission Requirement» Ore Sensing
stakeholder = "user"
text = "The robot shall be able to see the ore."
«Mission Requirement» Ore Approach
stakeholder = "user"
text = "The robot shall be able to move to the ore."
«Mission Requirement» Ore Extraction
stakeholder = "user"
text = "The robot shall be able to extract the ore."

Figure 16: SysML Requirements diagram in package 'RM2 Domain Requirements'.

4. SYSTEM SOFTWARE ARCHITECTURE

The system realization section of the system model describes the realization of the robot systems implemented in the project (RM1, RM2, etc.). Figure 17 depicts the major system elements that compose the Robominer system in a hierachical diagram.

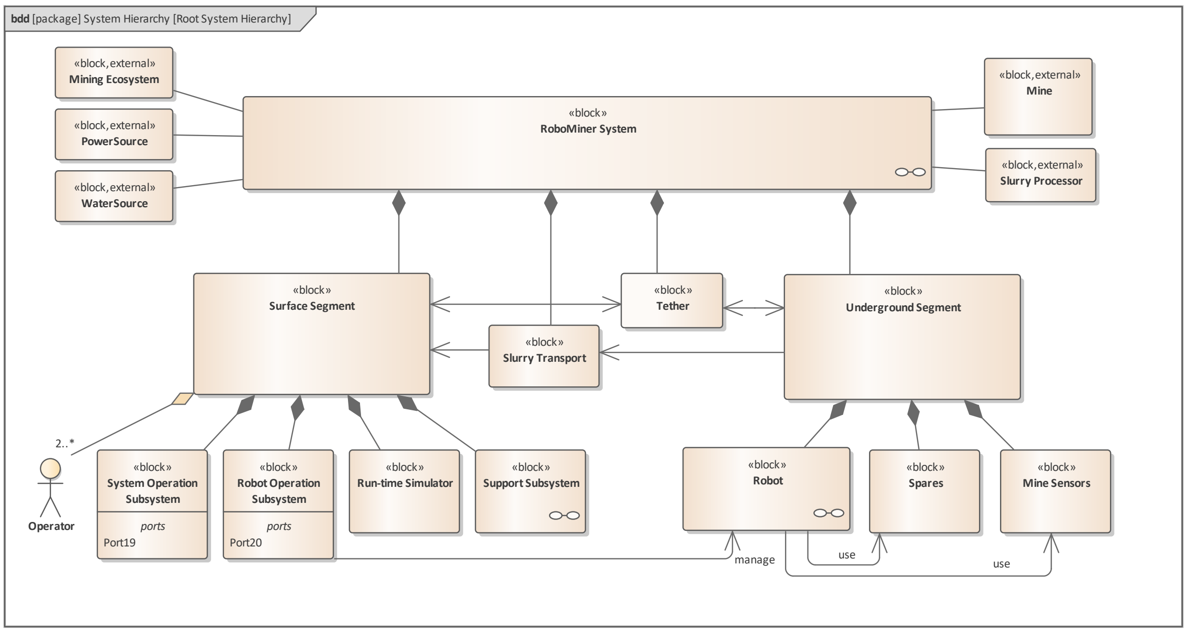

bdd [package] System Hierarchy [Root System Hierarchy]
«block,external» Mining Ecosystem
«block,external» PowerSource
«block,external» WaterSource
«block» RoboMiner System
«block,external» Mine
«block,external» Slurry Processor
«block» Surface Segment
«block» Tether
«block» Underground Segment
«block» Slurry Transport
2..*
Operator
«block» System Operation Subsystem
ports
Port19
«block» Robot Operation Subsystem
ports
Port20
«block» Run-time Simulator
«block» Support Subsystem
«block» Robot
«block» Spares
«block» Mine Sensors
manage
use
use

Figure 17: Major system elements in the Robominer System.

This deliverable however only addresses the specification of the **software architecture** for the robot controllers. In this sense, it contains just the identification of major system elements and the identification of interfaces between them. These software elements are deployed over computing platforms that are parts of the different subsystems shown at the lower layer of Figure 17.

The system software elements described in this chapter are related to two clasess of software:

1) the **common assets** used across the system elements,
2) the **functional assets** identified in the use cases and scenarios analysis.

The packaging of the software (see Figure 18) is organised to address both the needs of the technical system implementation and the workpackage structure of the project.

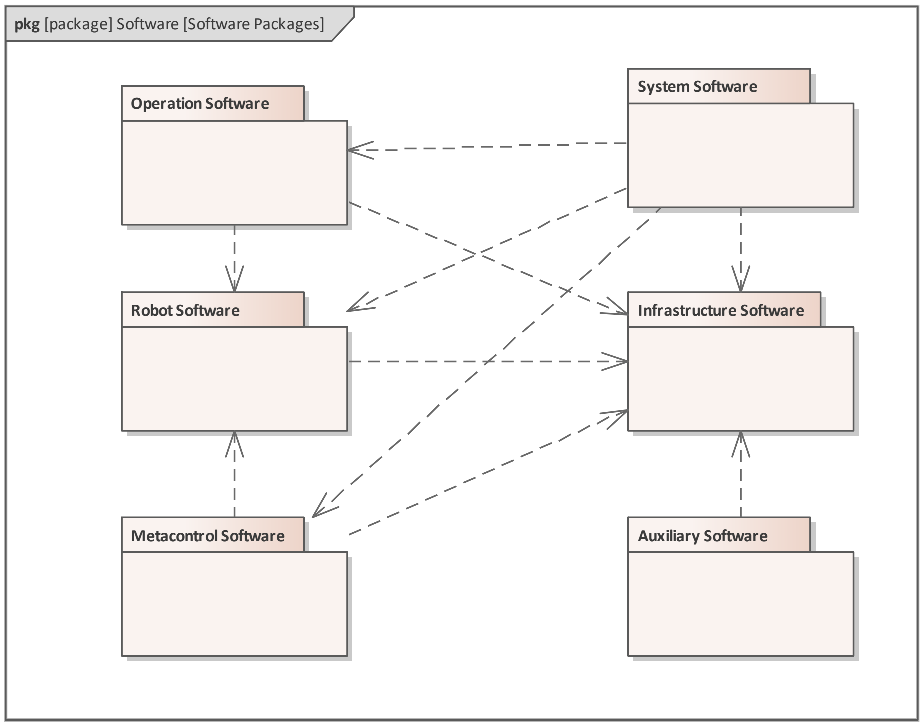


Figure 18: Software packages for control software assets.

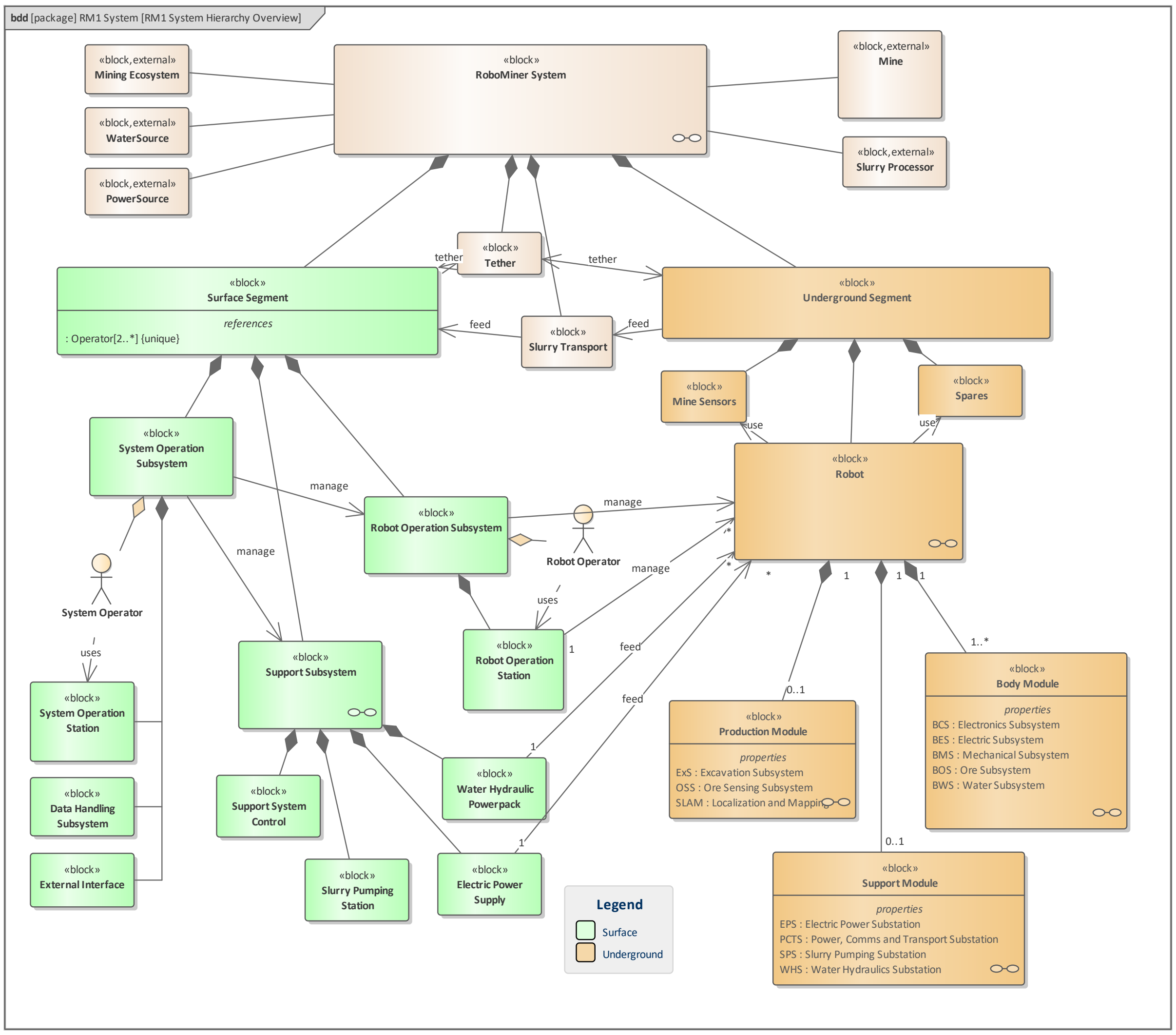

bdd [package] RM1 System [RM1 System Hierarchy Overview]
«block,external» Mining Ecosystem
«block,external» WaterSource
«block,external» PowerSource
«block» RoboMiner System
«block,external» Mine
«block,external» Slurry Processor
«block» Tether
tether
tether
«block» Surface Segment
references
: Operator[2..*] {unique}
«block» Underground Segment
feed
«block» Slurry Transport
feed
«block» Mine Sensors
«block» Spares
use
use
«block» System Operation Subsystem
manage
«block» Robot Operation Subsystem
manage
Robot Operator
«block» Robot
manage
System Operator
uses
uses
«block» System Operation Station
«block» Support Subsystem
«block» Robot Operation Station
1
feed
feed
*
1
1
1
1..*
0..1
«block» Production Module
properties
ExS : Excavation Subsystem
OSS : Ore Sensing Subsystem
SLAM : Localization and Mapping
«block» Body Module
properties
BCS : Electronics Subsystem
BES : Electric Subsystem
BMS : Mechanical Subsystem
BOS : Ore Subsystem
BWS : Water Subsystem
«block» Data Handling Subsystem
«block» Support System Control
«block» Water Hydraulic Powerpack
1
1
0..1
«block» External Interface
«block» Slurry Pumping Station
«block» Electric Power Supply
Legend
🟩 Surface
🟧 Underground
«block» Support Module
properties
EPS : Electric Power Substation
PCTS : Power, Comms and Transport Substation
SPS : Slurry Pumping Substation
WHS : Water Hydraulics Substation

Figure 19: Full system hierarchy.

4.1. Common Assets

The common assets are software elements that are used across all the subsystems of the system hierarchy (Figure 19). There are two major classes of common assets:

- **System Core Software**: This is Robominers-specific software used across the different system elements to provide common capability needed for the operation of the system. This is software developed in the project.
- **Platform software**: Runtime software in computing platforms that support the system deployment. This is open-source software already available outside the project.

4.1.1. System Core Software

4.1.1.1. System Agents and Nodes

The Robominers software system will be composed by a set of interacting System Nodes (SNs) managed by System Agents (SAs).

SNs encapsulate the software functions needed for the different activities of the system (e.g. moving, drilling, etc.) and provide a common management interface and a specific operation interface. SNs are passive and will be managed by a System Agent (SA). SAs provide capabilities for distributed cognitive decision making that synergically contribute to the system operation. SAs endow robot modules with the required capabilities for self-management and collaboration with other modules (the ROBOMINERS robot is a modular system).

The computing hardware for robot controllers will be a distributed network of computers. Each one fitting the needs and constraints of its function and deployment conditions. For example, each leg may have its own computers for local autonomy and hierarchical control. SNs will be deployable on two different classes of computing platforms: base and reduced (e.g. Olimex/Linux systems or smaller OSless microcontrollers). SAs will need a base computing platform capable of supporting ROS2 middleware.

Reduced computing platforms shall provide some kind of connection also available in the base platform that runs the SA or a Node Adapter. A Node Adapter is an SA that runs on a Base Platform managing a SN that runs on a Reduced Platform.

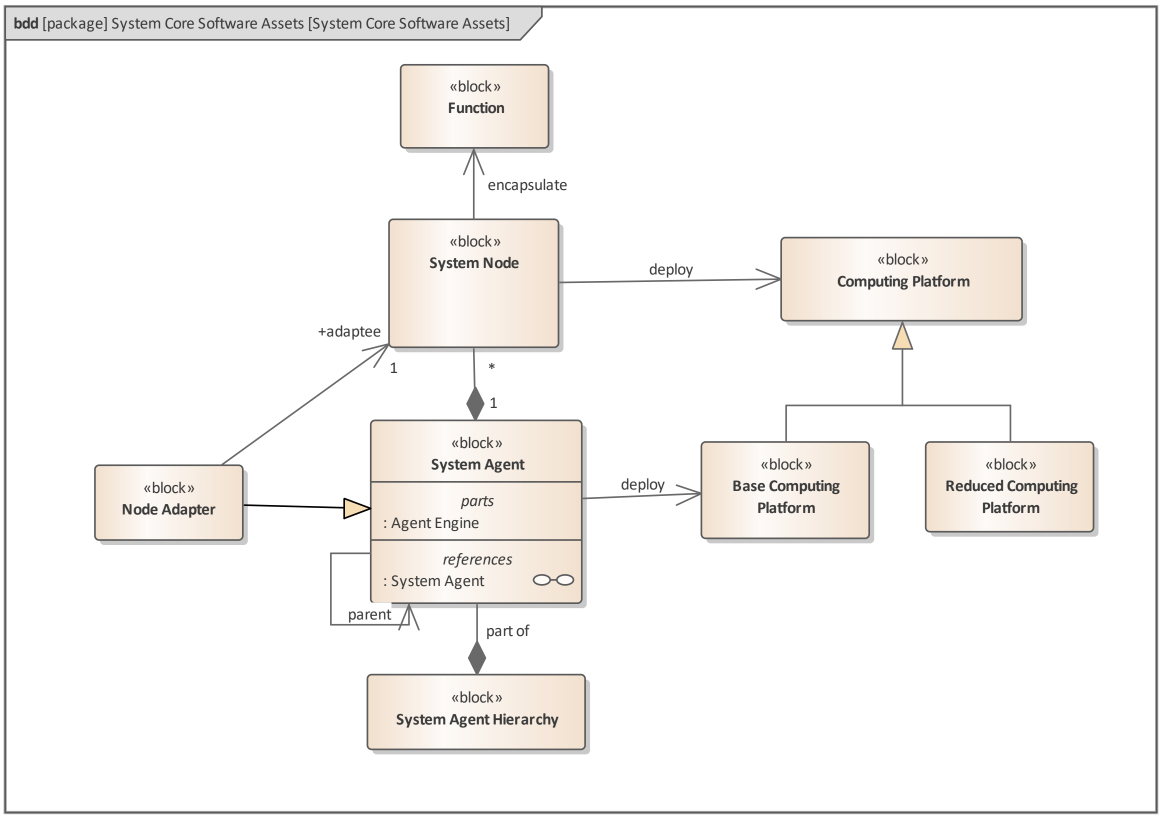


Figure 20: System Core Software Assets.

Active SAs implement an epistemic control loop to pursue goal-directed action using ontology-based representations of the mission, the robot and the environment to: 1) perform mission-oriented directed activity, and 2) address contingencies. The details of the System Agent design are to be developed as part of the activity in T4.2 – T4.6 but some considerations concerning its desired capabilities are summarised in Figure 22 and section 4.3 below. Note that these capabilities are transversal to mission-specific ones (like moving or cutting rock). These are described later in section 4.2 Functional Assets.

Reflective SAs (see Figure 6: System architecture profile.) have the capability of self-reflection and control (metacontrol). This enables the recovery behavior by runtime adaptation to overcome disruption.

The functional state of any function-allocated subsystem is governed by the same state machine. The mission of the metacontrol system is to:

- drive this machine —using the self-awareness function— and
- try to maximise functionality (i.e. return to Nominal Operational State) —using the self-reconfiguration functionality.

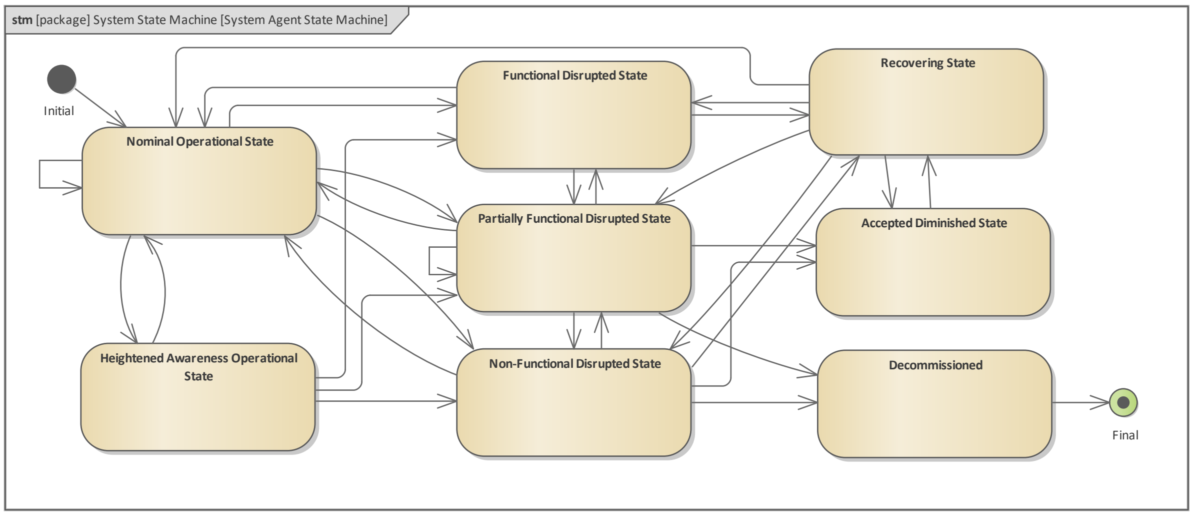

stm [package] System State Machine [System Agent State Machine]
Initial
Nominal Operational State
Functional Disrupted State
Recovering State
Partially Functional Disrupted State
Accepted Diminished State
Heightened Awareness Operational State
Non-Functional Disrupted State
Decommissioned
Final

Figure 21: System Agent state machine.

The SA state machine generalises the ROS2 node lifecycle to specifically address 1) diminished function states and 2) recovery from disruption.

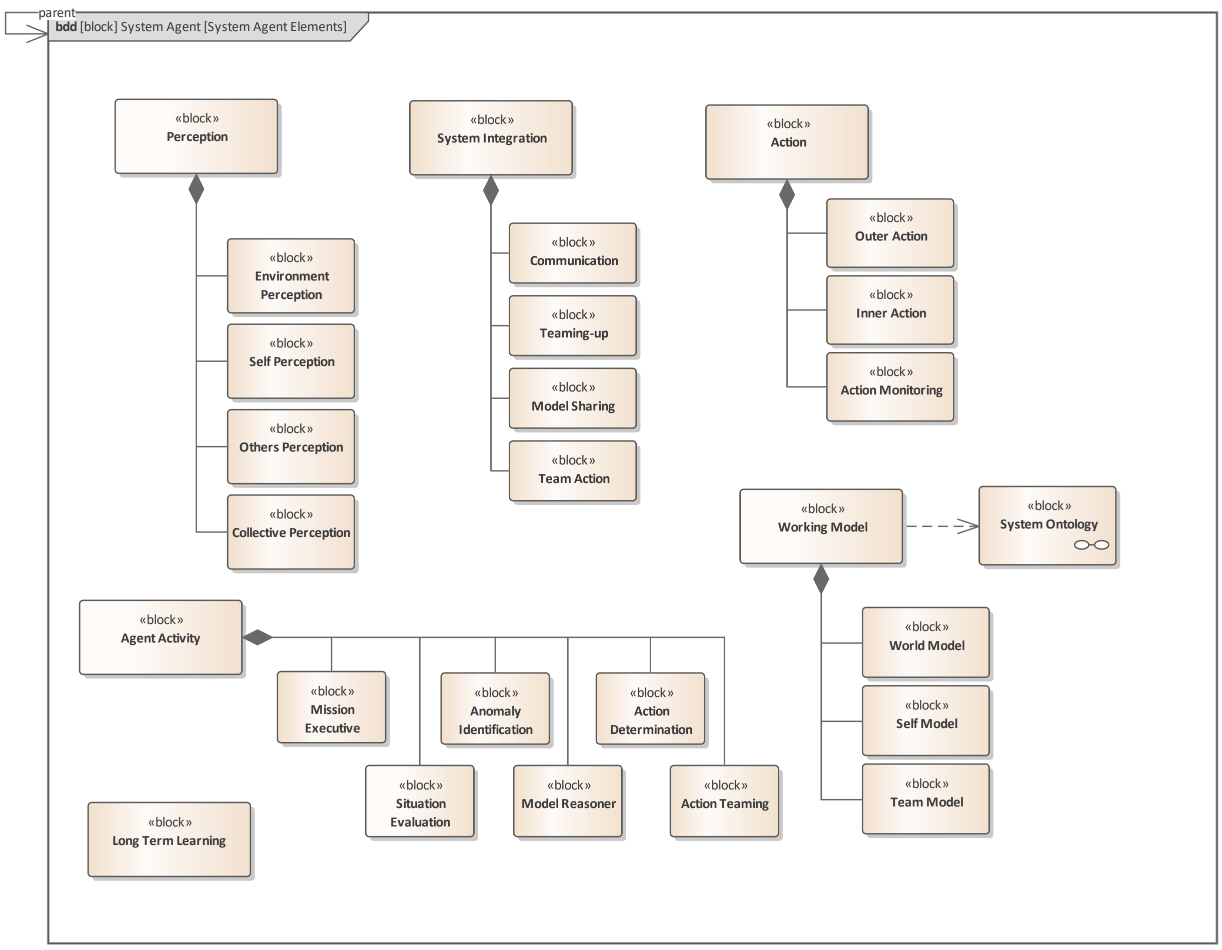


Figure 22: System Agent functional elements.

4.1.1.2. Ontologies

System knowledge is captured in knowledge bases for runtime reasoning. These knowledge bases are being built using formal ontology languages. The ROBOMINERS ontologies are a set of formal specification of concepts that are used to create knowledge bases for the metareasoner to represent and reason about system state and alternate system configurations.

While it is possible to represent and reason about any of the aspects of the system, the current approach is the development of a modular ontology to capture four system aspects (Figure 23):

- Movement — the movement of the robot in the mine using its locomotion system.
- Mining — the extraction of ore using the extraction tool.
- Mission performance — the achievement of the objectives of selective mining.
- Adaptation — the adaptive response to disturbance by means of a metacontroller.

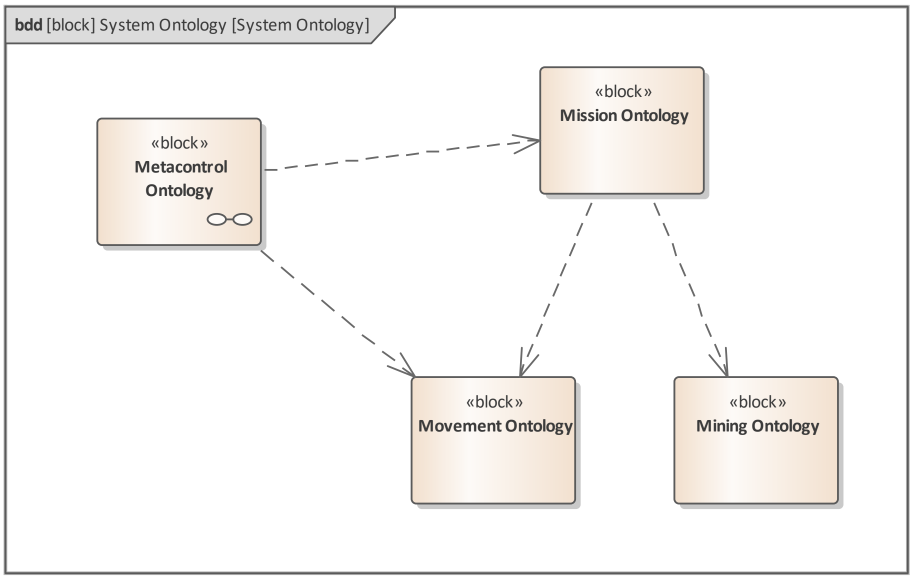


Figure 23: System ontologies to support reasoners.

4.1.2. Platform Software

Platform software is third party software used in the project to build and deploy the robot control system. This software includes operating system software, core runtime libraries and communications software for distributed application construction (Figure 24).

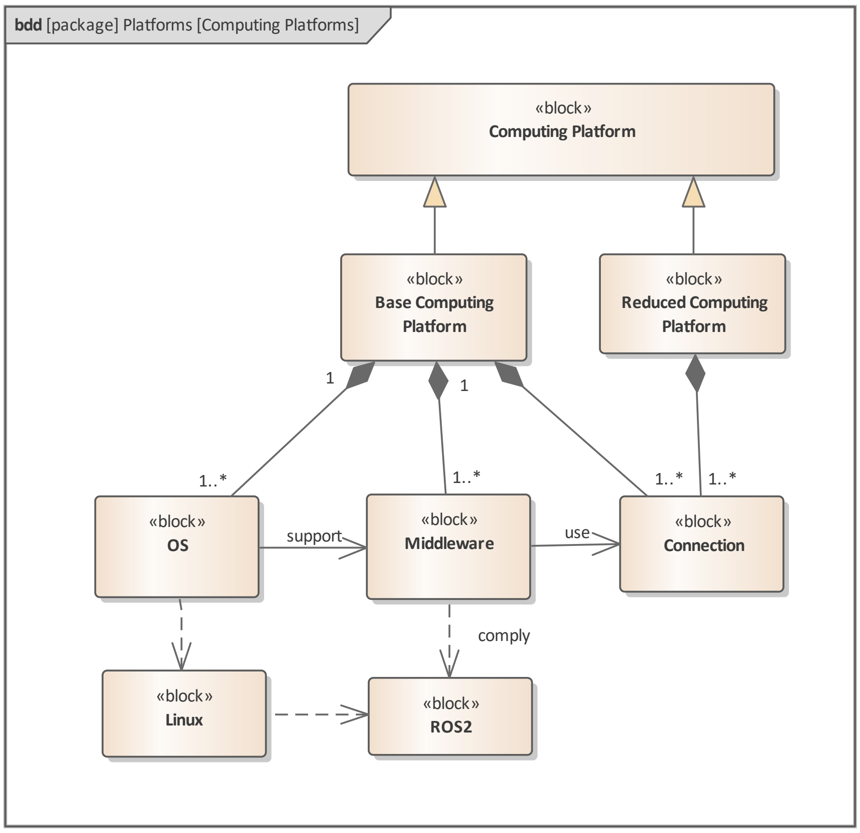


Figure 24: Platform software.

4.1.2.1. Computing Platforms

The ROBOMINERS software assets (system agents, system nodes and functions) will be deployed on two different classes of computing platforms:

- **Base computing platform**: any computing platform capable of running a full-fledged Linux distribution with real-time extensions and a complete ROS2 capability (e.g. Olimex A64-OLinuXino-1G SBCs).
- R**educed computing platform: any embedded computing platform that is not capable of running a real-time full-fledged** Linux (e.g. a small microcontrollers like the ESP32).

System Agents (SAs) will need a base computing platform capable of supporting ROS2 middleware. System Nodes (SNs) need a simpler, reduced platform. Reduced computing platforms shall provide some kind of connection also available in the base platform that runs the SA or a Node Adapter. A Node Adapter is an SA that runs on a Base Platform managing a SN that runs on a Reduced Platform.

4.1.2.2. Middleware

For the system integration, we will use **ROS2/DDS middleware** because it provides the needed capabilities for the controller and enables the use of packages available in the ROS ecosystem.

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has resources that we can leverage in the implementation of the robot control system. And it is open source, that swiftly aligns with project guidelines.

The substrate middleware of ROS2 is DDS, an open specification of data push middleware from the OMG. DDS provides a publish-subscribe transport that easily matches ROS1's publish-subscribe topic model. DDS uses the "Interface Description Language (IDL)" as defined by the Object Management Group (OMG) for message definition and mappings to network neutral formats enable the serialization of the data. Recent DDS implementations have also request-response style middleware, which matches ROS's service system (called DDS-RPC). Annex 2 uses the ROS2 interface definition language to specify some core interfaces of the system.

While we could use just DDS and not use ROS2 for the implementation of the robot controllers, the ROS2 distribution includes also tooling and reusable robot software assets that we can use in the engineering of the controllers. For this reason, we will use ROS2 (and possibly the deeply embedded variant microROS). For detailed information concerning this middleware visit https://design.ros2.org/.

4.2. Functional Assets

4.2.1. Robot Software

The robot embedded controller is a mission-oriented software. It controls the robot to fulfil the selective mining mission that is the core objective of the ROBOMINERS project. In this sense, the robot software controls three fundamental activities:

- Sensing: Perception of the environment in terms of geometry and mineralogy.

- Movement: Actuation on the propulsion system both to go to certain locations and also movement during mining operation.
- Mining: Extraction of ore from the ground and manipulation of extracted materials.

Each of this software elements require specific functional capabilities. Additionally, all subsystems participate in the whole system and require core capabilities that enable this participation. The diagram shown in Figure 25 shows some of these capabilities.

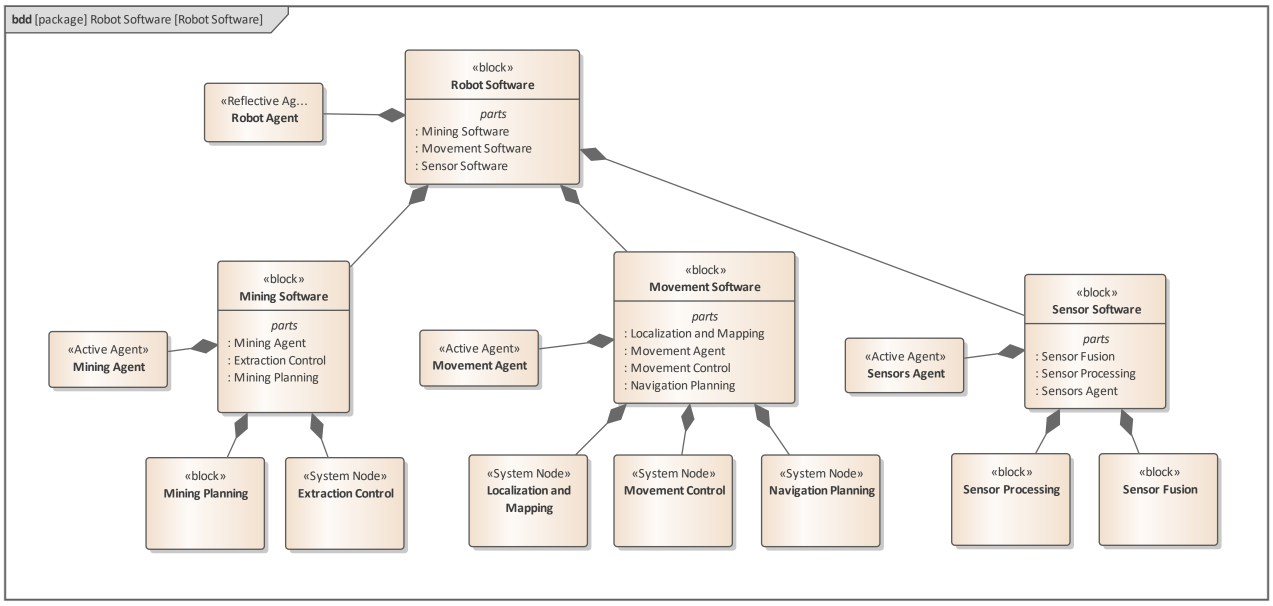

bdd [package] Robot Software [Robot Software]
«Reflective Ag... Robot Agent
«block» Robot Software
parts
: Mining Software
: Movement Software
: Sensor Software
«Active Agent» Mining Agent
«block» Mining Software
parts
: Mining Agent
: Extraction Control
: Mining Planning
«Active Agent» Movement Agent
«block» Movement Software
parts
: Localization and Mapping
: Movement Agent
: Movement Control
: Navigation Planning
«Active Agent» Sensors Agent
«block» Sensor Software
parts
: Sensor Fusion
: Sensor Processing
: Sensors Agent
«block» Mining Planning
«System Node» Extraction Control
«System Node» Localization and Mapping
«System Node» Movement Control
«System Node» Navigation Planning
«block» Sensor Processing
«block» Sensor Fusion

Figure 25: Robot Software.

4.2.2. System Operation Software

While the robot shall operate autonomously, it does so under the supervision of one or more human operators that also handle other mine systems besides the robot. Figure 26 shows some of the software elements involved in this high-level operation. The detailed design of these systems will be done in the future.

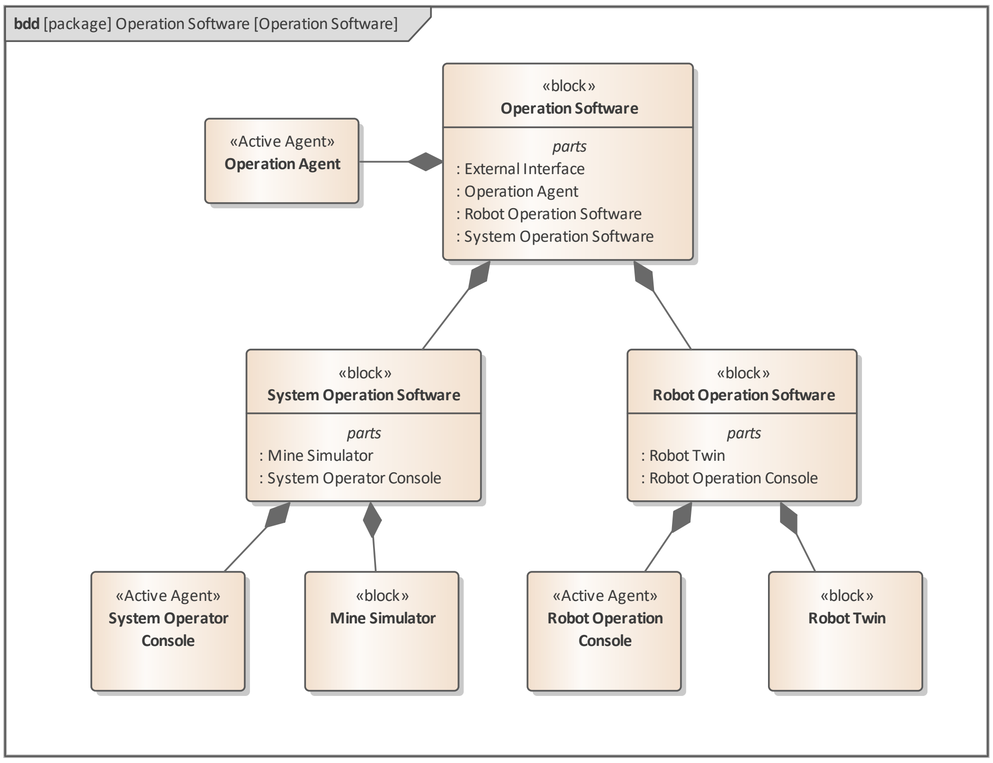


Figure 26: System operation software.

4.2.3. Auxiliary Software

This is software related to the operation and control of auxiliary elements at the system level. This includes water and power supplies, slurry management and tether control.

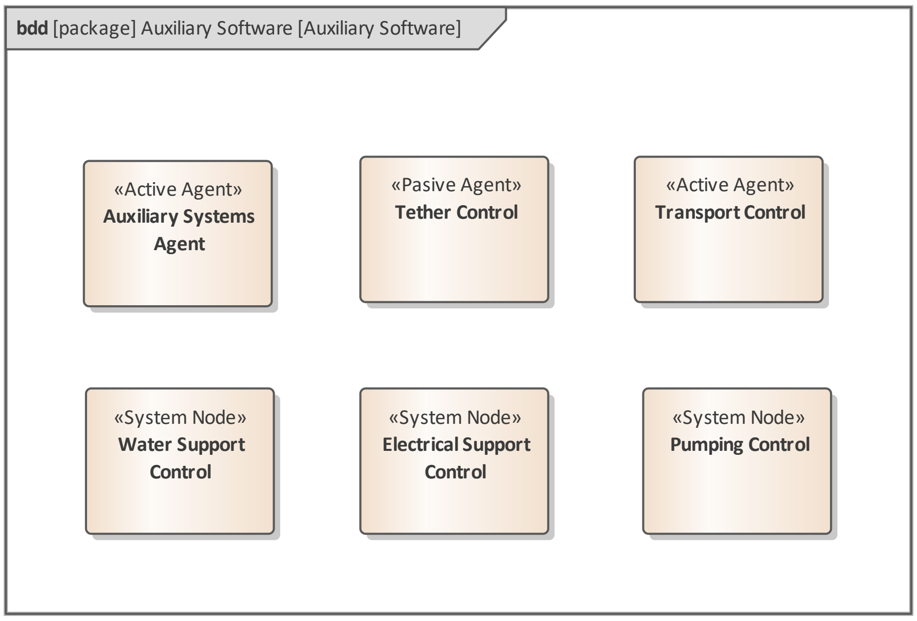


Figure 27: Package 'Auxiliary Software'.

4.3. System Control

The system control strategy is a **dynamic federation of system agents** that follow the system agent hierarchy shown in Figure 28. Dynamicity comes from the need of overcoming disturbances that will be managed by means of system modularization and distributed control.

4.3.1. Overall control structure

The top of the hierarchy is the Robominers System Agent that governs the state of the whole system (robot, operation system and auxiliary systems). Each of these subsystems have a managing agent that has its own goal and collaborates with upper level agents in the fulfilment of higher-level goals.

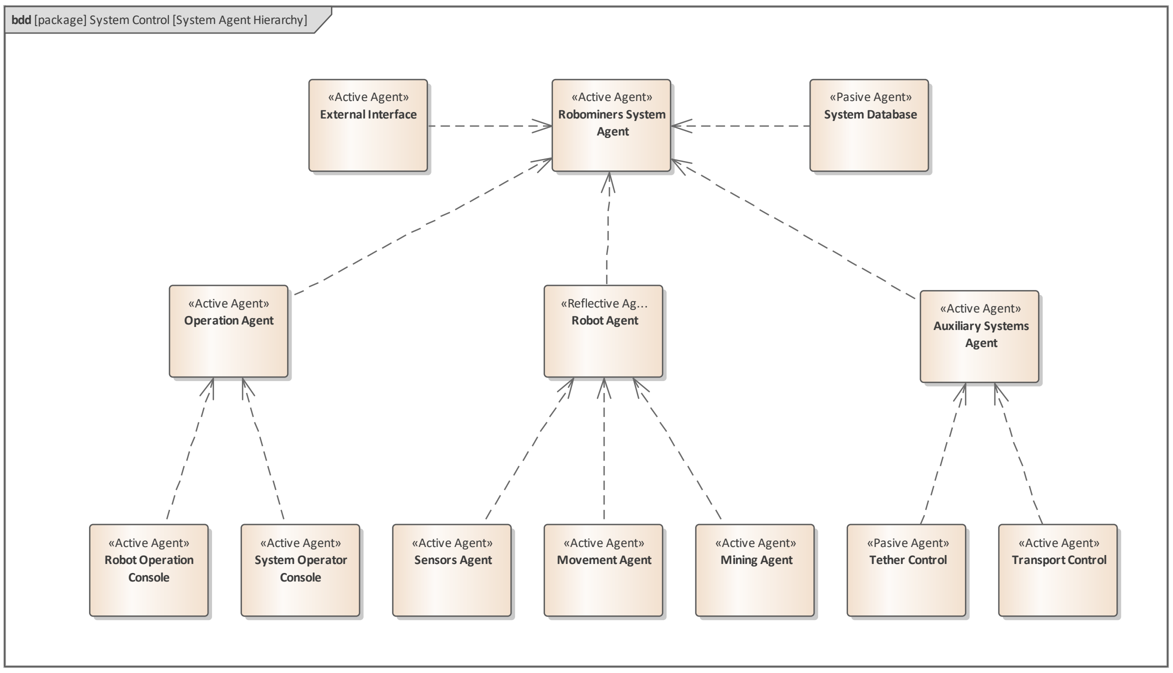

bdd [package] System Control [System Agent Hierarchy]
«Active Agent» External Interface
«Active Agent» Robominers System Agent
«Pasive Agent» System Database
«Active Agent» Operation Agent
«Reflective Ag… Robot Agent
«Active Agent» Auxiliary Systems Agent
«Active Agent» Robot Operation Console
«Active Agent» System Operator Console
«Active Agent» Sensors Agent
«Active Agent» Movement Agent
«Active Agent» Mining Agent
«Pasive Agent» Tether Control
«Active Agent» Transport Control

Figure 28: System Agent Hierarchy.

4.3.2. Behaviour generation and control

The SA overall control mechanism (see SA state machine in Figure 21) shall use a callback mechanism to register the functions used to implement the states' functionality. The nominal operational state is obviously the most important one, and the base strategy is to use a behaviour tree to govern the nominal activity of each agent.

At the time of this writing, for the implementation of the SA core functionality we will use:

- The ROS SMACC state machine library. SMACC is an event-driven, asynchronous, behavioral state machine library for real-time ROS (Robotic Operating System) applications written in C++, designed to allow programmers to build robot control applications for multicomponent robots.
- The ROS BehaviorTree.CPP behaviour tree library. BehaviorTree.CPP is an open C++ library that provides an extensible framework to create behaviour trees in ROS. It was designed to be flexible and fast, properties that match our needs.

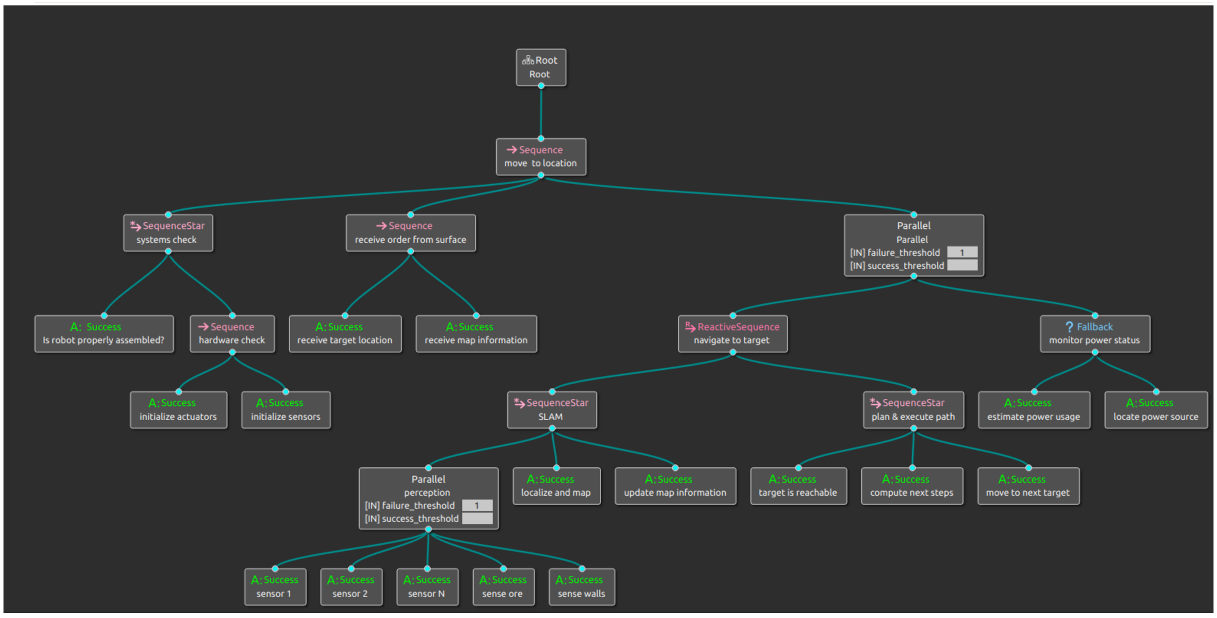


Figure 29: A move-to-location behavior tree visualized using the Groot tool.

4.3.3. System Metacontrol

Besides the nominal activity of the system modules using the described system behaviour generation strategy, the Robominer has a transversal capability for adaptation in the case of disturbances pushing the system beyond the nominal state. The disturbance handling states of the SA state machine are managed by the metacontrol software (see Figure 30).

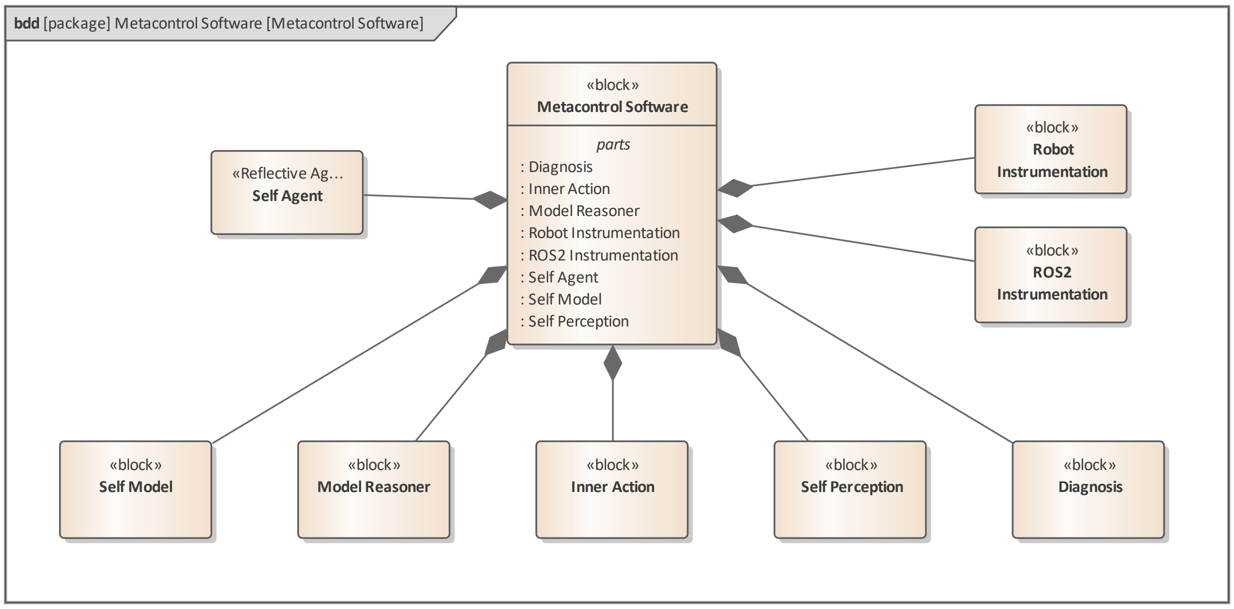


Figure 30: Metacontrol Software.

The metacontrol software is used by reflective agents to implement a self-awareness control loop. This metacontrol system is powered by a formal reasoner (we have selected the Pellet reasoner for the current work). The metacontroller performs runtime reasoning using a module self-model that enables the diagnosis and inner action necessary to transition from disturbed states back to nominal operational state (or a diminished functionality state if full recovery of function is not possible).

The formal ontologies described before are used in the construction of the knowledge bases that constitute the self model. The system-related knowledge base is structured at three abstraction levels to allow reusability and handling of different system agents:

- Teleological metamodel: Common baseline, it coordinates the system mission.
- Domain ontology: Knowledge relative to the main objective of a sub-system, e.g., navigation for the locomotion sub-system or rock mining for the extraction tool.
- Agent individuals: Information relative to the concrete implementation of an agent.

The functional system instrumentation needed to couple the metacontroller with the underlying systems —meta sensors and actuators— will use ROS2 introspection mechanisms as far as possible to reduce the needs for tailoring reused ROS packages from the ROS ecosystem.

5. SYSTEM BEHAVIOR

The development of implementations of behaviors will be guided by the concrete needs of the target operational scenarios (the scenarios that will be used in the system demonstration and evaluation). The analysis of scenarios will help identify the specific functions needed for robot behavior realization and the identification of interfaces and data types.

The objective of the control system is to make the miner robot behave as needed. To analyse this behaviour, we shall describe the operational scenarios of interest –the focus of robot activity– in a way that could be understood by every other project stakeholder.

Given the fact that the project vision points to a real miner robot, the scenarios described below are described in terms of a general miner robot. Concrete demonstration of the scenarios will be analysed later (see section XXXX below). Note that in the project there are several robot implementations.

Some of the structural/functional descriptions in the SysML system model are quite general –they apply to different robots– but others may be specific of concrete ones. For example:

- The RM1 robot –the full-size robot– is able to perform real mineral extraction.
- The RM1 robot can operate under water.
- The RM2 robot -a modular, small scale system- can perform physical reconfiguration.

In the same way, the scenarios described below do not apply to all robots –real or simulated– developed in the project. Besides the concrete robots we refer to, some project activities with specific research interests may be carried out in test pools, mock-up environments or simulations, not necessarily in a real mine.

5.1. Scenario 1: Go to Location

The robot is located somewhere in the mine and shall go to another location (e.g. to the ore extraction front). This happens through free space.

This scenario may develop after the robot being thrown in modules to the deposit via a large diameter borehole (TA description; in this case, Scenario 6: Self-assemble, will ensue), or may happen after a robot repair, or as a result of a higher-level mine operations decision to change the extraction front where the robot is working.

Once the robot is properly assembled, it should prepare itself for the operation. Systems shall be powered-up and started until reaching an operational state.

When in an operational state, the robot may receive orders from the surface segment (or from another robot or system agent?) to perform some mission. Mission start is then ordered from the operator console.

All robot movements within the mine will be recorded on a mine map (a master version of which will be held in the server for access and use by the human operator) and will update that mine map.

There are two important components of the Mine Map: (1) a geological model of the deposit and its surrounding rock, and (2) a geometric map of existing and planned mine workings (tunnels, shafts, ramps, stopes, etc.)

Whether in an old or new mine, there will be information on

1) the 3-dimensional geology (in a geological model of the mineral deposit) and
2) the 3-dimensional map of workings, existing or planned.

The geological model will be derived by numerical modelling methods from data that are obtained from any existing mine workings, drill holes, or other similar sources including geophysical exploration. More detailed description of the deposit modelling is to be included in deliverable D6.6.

First, the robot shall use the sensor systems to establish an initial localization and start mapping the mine. While the initial position of the robot may be known and provided by the human operator, it may also be the case that it is not explicitly stated or included in the mine map. Besides this, the robot needs to bootstrap its self-locating capability.

The geological (deposit) model is generally derived from data with small resolution (5 - 10 metres) compared to the robot's dimensions, so, at the beginning, it will be unable to provide precise positioning information for the robot to start mining, but only generalised guidance. As robotic mining proceeds, the model will be progressively updated to improve its local precision.

As concluded in the deliverable D1.1 Report on sensor performance and navigation strategies for mining environments, robotic **localization** and **mapping** are traditionally based on visual perception. However, in ROBOMINERS one of the challenges is operating in the mining environment blindly. This means following the ore using geophysical methods, sensing the surrounding environment's physical properties and objects or obstacles in the close vicinity. The mapping and localization may be based on proprioception (pressure sensors and IMU (Inertial Measurement Units)), electromechanical whiskers, and using geophysical sensing (XRF, conductivity and induced polarization). These types of data will be used to update and improve the geological model of the deposit (D6.6).

Ore sensing capability is especially relevant when performing the ore extraction (as part of the selective mining capability; see Scenario 2: Mine the ore) but is also relevant in any movement because it can incrementally improve the map info, and can be used for localization purposes

Note that a **mine map** contains several classes of spatially distributed information: mine geometry –esp. actual mine geometry but it may also contain planned future geometry-, mineral composition, flowing currents, etc. The information is gathered by the robot but also from other past and present sources. For example, think about a possible concrete go-to scenario concerning systematic **ore mapping** in an old mine.

The map can contain several geomechanical and geophysical **aspects** (see D.6.6) that shall be aligned and incrementally included in the map:

- Lithology of host rock and ore materials, mineralogy, geochemistry, physical properties
- Rock structure: e.g., joints, variation in rock strength, etc. affecting stability of mine workings
- Presence of hazards such as those identified in deliverable D5.2

Note that the map is not static nor just scalar:

- It is incrementally updated (by all information sources).
- It can contain moving entities (e.g. moving water). The map may contain velocities.

Note that a mine map may contain present, past, and future content. For example, an exploitation plan for the mine as created by a mine engineer (a mine design, a mine layout).

If a mining strategy or mine layout is adopted which includes 'caving', the ore is not immediately transported to surface but forms a temporary heap or stockpile within the mine, usually below or adjacent to the cutting face (as in shrinkage stopping: deliverable D2.2, section 4.1.4). In such a case, this temporary storage needs to be recorded in both the geometric mine map and the geological model.

These aspects are to be used in the decision-making concerning robot operations: by the robot itself, by system-level planners or by human operators. The mine map is shared at a system level. For example, the deposit map may be used by a planner for systematic extraction or by the robot controller for local direction control (e.g., when performing an autonomous bioinspired mining strategy).

The operator or the meta-controller can oversee the robot state and the map that is being generated to monitor or intervene if the data produced has too much deviation. Once the robot has produced a map, it shall locate the most promising mineral vein to start Scenario 2: Mine the ore when operating in fully autonomous mode.

The decision to move to this place may be full autonomous –the robot decides- or following a mixed initiative approach –it is the **robot-operator pair** who decides. We can establish a set of scaled autonomy strategies from fully teleoperated to fully autonomous and decide what to use in a concrete situation.

The robot moves from its location to the cutting front using its navigation methods. An operator monitors the operation. During movement, the robot updates the map. During movement, the robot avoids obstacles. This procedure may be entirely operator-controlled or may be semi-autonomous, with the operator identifying intermediate "way-points" between the current robot position and its destination, but the robot making decisions on how to travel between those points.

The scenario ends when:

- The Robot reaches the target location (e.g. a cutting front or a specific destination).
- The Robot decides it cannot reach the location.
- The Operator stops the mission.

5.1.1. Scenario 1 Variants

There may exist specialised, specific variants of the go to location scenario:

- Robot enters the mine. For example, through a horizontal or ramp entry instead of a borehole.
- Robot exits the mine.
- Robot leaves cutting front to go to a storage/tether/maintenance area.
- Robot leaves cutting front to go to another cutting front.
- Robot escaping a collapse.
- Borehole Down: The robot enters a +/- vertical borehole which reaches the ore body, and then descends through borehole and starts mining.

5.1.2. Scenario 1 Functions

- Sense ore – Get map of ore
- Sense wall – Get map of wall to determine free space
- Sense obstacle – Get map of obstacle to determine free space
- Navigate – Establish desired trajectory in free space
- Move – Follow trajectory
- Set mission – Establish go to location mission parameters
- Locate – Determine robot location on mine map
- Update Map – Update mine map using sensed information

5.1.3. Scenario 1 Entities

- Robot
- Mine
- Location – 6D location of all robot elements.
- Motion unit – The robot subsystem that can move the robot
- Sensor data – Information obtained from sensors
- Mine Map – 3D representation of the mine
- Mission – Specification of goals, resources, and constraints in the go to location mission
- Operator – Human operator in surface station
- Obstacle – Object that may interfere with movement path
- Free space – Space that the robot may move through without digging

5.2. Scenario 2: Mine the ore

Once the robot is at the extraction front, it realizes the main value of the project: **selective mining**[3]. The system has to identify the mineral vein or other deposit type, to extract rock in the right direction.

Choosing the *right digging direction* can be made using a path planner based on the geophysical sensor information and other parameters such as tool wear, robot energy, risk, etc. and normally will be largely dependent upon a mine layout design that has been specified in advance. This may be a conventional mine layout or a bio-inspired mine layout, selected from a very wide range of possible layouts such as those described in deliverable D2.2. Mine layout plans are relevant for deciding where to dig (e.g. fully autonomous operation shall not dig-out a pillar).

As the robot collects detailed local information on mineral grades, rock strength, etc., the mine layout itself may be progressively modified, and the planned future workings may be updated in the Mine Map. Deliverables D6.6 and D6.7 give more detailed accounts of the algorithms and mining methods needed for selective mining.

Then the system must establish a steady and stable position to start the production tool. Once extraction starts, the extracted material is comminuted, moved to the rear of the robot to convert it

[3] Note that the *selective mining* concept can be applied to the concrete act of mineral extraction (dig here or there) but it can also refer to more high-level decision-making concerning the decisions related to mining at a higher, ecosystem level (e.g. deciding if mining or not depending on cost of electricity).

into slurries (or slurrified and then moved to the back for pumping). This allows transport (such as pumping) of the extracted material towards the surface base station.

During extraction, the robot is sensing the environment and also using the digestive sensing to get information from the slurry about mineralogical composition. This information can be used both locally and globally to both update the mine map and control the digging activity of the robot.

V-transport can be implemented in a variety of ways, that are out of the scope of the project. . However, it will be taken into account in simulations).

The following figure (from D3.2) shows the processes happening during ore extraction:

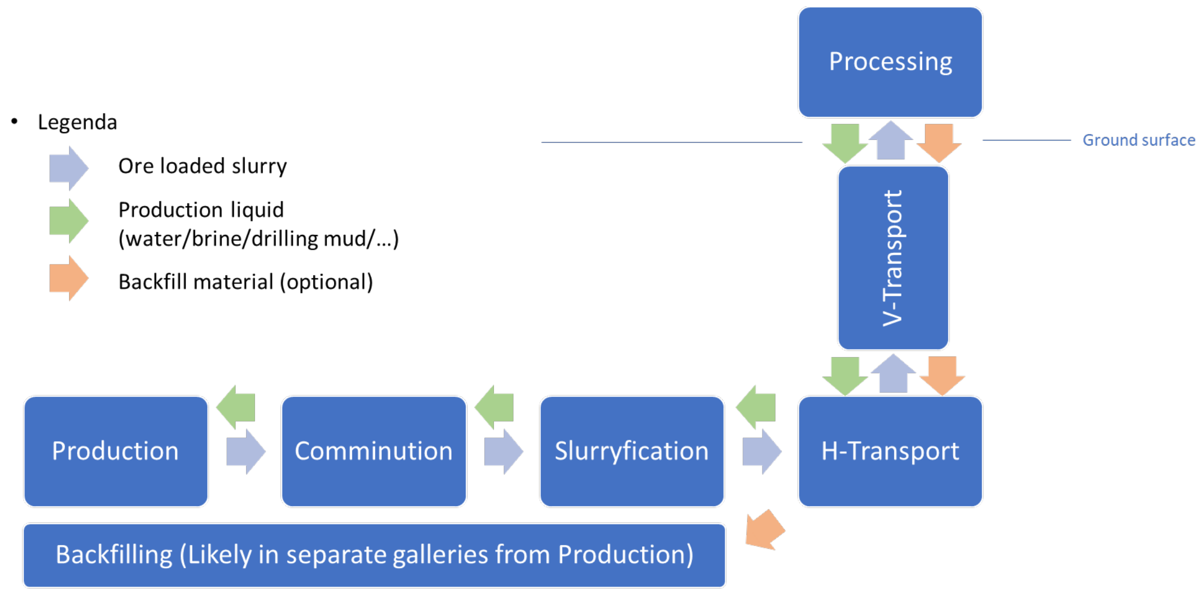

Legenda
Ore loaded slurry
Production liquid (water/brine/drilling mud/...)
Backfill material (optional)
Processing
Ground surface
V-Transport
Production
Comminution
Slurryfication
H-Transport
Backfilling (Likely in separate galleries from Production)

Figure 31: Fundamental activity of the Robominer mining process.

During this process, the robot state is monitored by an operator and the meta-control module. In case of contingency, the system may need to be reconfigured. This could be done autonomously or directed by the operator, triggering Scenario 5: Reconfiguration.

This scenario ends when the robot stops by:

- Completing the mission (reaching a certain amount of rock, reaching a location).
- Ore vein extinguishes/is not profitable – robot stops and reports to operator
- Reaching an open space.
- Failure. Total or partial.
- Operator order.

5.2.1. Scenario 2 Variants

There may exist specialised, specific variants of the mine the ore scenario:

- Robot digs to create free space.
- Robot digs escaping a collapse.

5.2.2. Scenario 2 Functions

- Plan dig – Establish a dig trajectory
- Produce – Dig with the tool to extract the ore
- Sense ore – Get map of ore
- Sense wall – Get shape of wall to determine tool operation
- Move – Follow dig trajectory
- Comminute – Reduce grain size
- Transport – Move comminuted material to slurrifying unit. Also called h-transport.
- Slurrify – Convert comminuted material into slurry
- Hoist – Move slurry to surface. Also called v-transport.
- Set mission – Establish mine the ore mission parameters
- Locate – Determine robot location on mine map
- Update Map – Update mine map using sensed information
- Sense mineralogy – Geochemical sensor of the mineralogical composition of the ore

5.2.3. Scenario 2 Entities

- Robot
- Mine
- Operator
- Extraction front
- Production tool
- Extracted material
- Ore moving unit
- Comminution unit
- Slurry
- Slurrifying unit
- Pumping unit
- Mine map

5.3. Scenario 3: Power assurance

During the robot operation, the robot may use hydraulics to power some subsystems, but other subsystems must have an electric power supply. Handling of electric power is critical for all robot operations (not only ore extraction or robot movement).

For example, to extend capability the robot may be operating at low-energy conditions. For this, a change in how the robot is operating may collaborate to reduce energy consumption and hence extend its energy autonomy. The authority for this mode setting may come from the operator who is monitoring the operation or the robot meta-controller, that may inform the system that it should adapt to the energy available to achieve some goals. This system change may imply involving Scenario 5: Reconfiguration or may be based on simple parameterization of subsystems. The reconfiguration may be done autonomously or aided by the surface operator.

Reconfiguration could target a change in robot and control parameters such as robot speed, sensor updating rate, etc. to operate under not optimal but sufficient conditions. Besides, if the robot is performing Scenario 2: Mine the ore, it can adapt the production tools parameters (e.g. to reduce power peak consumption).

Furthermore, reconfiguration may also imply a large-scale change in the robot's structural organization. If a module does not have sufficient power –e.g. during navigation or material extraction– it can search other modules with spare power. This power seek capability includes e.g. reaching a backup power module or power transferring from other robot modules. In some cases, power assurance may require to couple with another module.

In the long-term view, these attachments/detachments will be done autonomously or teleoperated by the surface operator. In the framework of the project, this will imply a human performing the reconfiguration in the physical prototypes (RM1 and RM2) while it could be automatic in computer simulations. Once the robot is connected to a power supplier, it can recover its normal operation.

In this scenario the robot power monitor determines that the available local power is insufficient for the planned activity –e.g. go to location or mine ore– and seeks an extra source of power. The robot finds a spare, fully charged robot module in the mine map, goes to it, and uses the self-assemble capability to incorporate the module.

The scenario ends when:

- The robot is back to the original situation with enough power to continue its mission.
- The robot cannot find an available power source that fulfills the requirements for the mission.
- The operator stops the scenario.

5.3.1. Scenario 3 Variants

- Ordered shutdown
- Go to recharge station

5.3.2. Scenario 3 Functions

- Monitor power status
- Predict/schedule power usage
- Find power source
- Share power

5.3.3. Scenario 3 Entities

- Robot
- Operator
- Robot devices
- Battery
- Power source
- Module

5.4. Scenario 4: Fault detection

To augment the robot autonomy, diagnostic methods must be implemented. The data produced by self-diagnosis may be monitored by the surface operator and/or the meta-controller.

During the robot operation -navigation phase, or material extraction phase- an observer control node can be running. This observer could monitor the energy consumption, the reliability of the sensor readings, control the production tool and navigation performance, etc.

The monitoring loop can find errors at a hardware level, software level, or in the communication between modules. In the first case, for example, a drilling leg may break during the navigation phase or the production tool may be worn out to keep extracting material. In this case, the system will receive a notification message, and the operator or the meta-controller may decide to start Scenario 5: Reconfiguration.

Similarly, in case of communication or software failure, the robot may have some self-repair actions such as re-running failure nodes or trying other communication channels. However, if the failure cannot be addressed, Scenario 5: Reconfiguration may be needed.

In this scenario a set of faults –both hardware and software– will be injected in the system during operation. The robot shall:

1. Detect them.
2. Diagnose.
3. Update the system KB.
4. Inform the operator.
5. Initiate corrective action (note that the execution of the corrective action is not part of this scenario; e.g. see Scenario 5).

5.4.1. Scenario 4 Variants

Fault tolerance can be achieved with different strategies. The RM metacontrol strategy uses dynamic system adaptation based on system knowledge. A variant could be achieving fault tolerance by simple module redundancy.

5.4.2. Scenario 4 Functions

- Detect fault
- Diagnose fault
- Update system knowledge
- Inform

5.4.3. Scenario 4 Entities

- Robot
- Operator
- Fault
- System knowledge base
- Diagnoser
- Propioception

5.5. Scenario 5: Reconfiguration

One of the capabilities sought in ROBOMINERS is robot resilience. This scenario addresses the capability of the robot to reorganize itself to overcome circumstances and be mission-level resilient.

This scenario starts when the surface operator or the meta-controller decides a reconfiguration is needed. This may be due to a variety of circumstances: a robot structural failure, a communication failure, energy needs, mission changes, etc.

Reconfiguration can take place at any system element (esp. it can happen both at the hardware and software levels). In any case, the first step is to establish the best system configuration given the current objectives, the cause that triggered the reconfiguration and both the robot and the mine state.
This selection may be aided with combined strategies: i) a system knowledge base with engineering and mission knowledge, ii) a digital twin to test different reconfiguration strategies and evaluate its impact in the operational scenario, iii) transfer learning to apply simulation-trained techniques to the run-time situation, etc.

Once the new configuration is selected, we must identify which changes need to be made according to the current situation and plan how the changes should be executed to impact as little as possible the operation.

If the reconfiguration required is at hardware level, the robot shall detach the failing or non-required modules. Then then new modules shall move to the correct position, or the robot shall move towards the modules. Once the modules are close to the robot, Scenario 6: Self-assemble may be triggered. The surface operator will monitor this phase, teleoperating the reconfiguration if necessary.

If the reconfiguration is at a software level, the operator or the meta-controller may decide to end some ROS nodes and launch some new ones or change its operating parameters. This change in the control software may be caused by a change in the module's availability, e.g., a sensor is no longer available, so similar data is obtained by some (probably less optimal) sensors. As this change in the sensor data quality may affect security, the robot shall change its operating conditions such as speed or drilling parameters.

Another possibility is that hardware reconfiguration provides some new capabilities to the system, so the software is reconfigured to take advantage of such capabilities, e.g., adding another body so the robot doubles its size and quantity of sensors. Then the software shall recalculate the robot locomotion strategy to reach an optimal use of more legs to move.

In the concrete reconfiguration scenario, we will address the failure of some system element: the robot will react a to –simulated– failure on a mechanical propulsion element.

5.5.1. Scenario 5 Variants

- Software only reconfiguration: the robot software is adapted to address new mission.
- Incorporation of new capability: the robot will react to the incorporation of extra sensors.

5.5.2. Scenario 5 Functions

- Reconfigure
- Meta sensing
- Meta acting

5.5.3. Scenario 5 Entities

- Robot
- Operator

- System configurator
- System knowledge base
- ROS
- Meta sensors
- Meta actuators

5.6. Scenario 6: Self-assemble

In the TA description, the robot is assembled from parts. In the self-assemble scenario a single robot module is attached to an extant robot. While the base scenario considers assembling just one module, in some sentences we may refer to several modules to consider the possibility of assembling several modules and what are the potential implications.
Scenario 5: Reconfiguration
The robot module to be attached is located somewhere –in a small area– close to the robot and decoupled from it. This scenario may occur, for example, when a required module that is not connected to the robot body must be attached to it, or when a new robot module has been delivered in-situ (see TA description). This may happen also if an agent (e.g., the meta-controller or surface operator) decides that a hardware reconfiguration is needed. In this case, a module can be assembled to overcome a failure or provide new system functionality. See e.g. scenario 5 variants.

Choosing the *right robot hardware architecture (structure)* to adapt to **selective mining requirements** and its **operating contingencies** can be made by closing a structural control loop on top of the miner robot operation system. This control loop may be closed by the surface operator, monitoring the mission development. In deposits in which its characteristics allow a higher level of autonomy, the meta-controller subsystem may be in charge of selecting the required robot architecture.

As mentioned in section 1, scenario 1: go to location, decision-making may be fully autonomous –the robot decides- or follow a mixed approach –it is the **robot-operator pair** who decides. We may establish a set of scaled autonomy strategies, from fully teleoperated to fully autonomous, to use accordingly to concrete situations.

This self-assemble scenario may start before or after a decision is made on what is the robot architecture to be enacted; i.e the assembly will be directed by a pre-specified architecture or the assembly will be directed by a mission specification (this involves some self-organization capabilities described in scenario 5).

First, the system shall **locate the required module(s)**. The modules shall be identifiable, locatable, and approachable. Depending on the complexity of each module –a sensor, a production tool, or a large robot module– it may have self-identification, self-location, and self-movement capabilities. If not, robotics commonly uses identification tags (bar codes, RFID, etc.) or/and artificial vision to recognize and locate objects and this may be used for the new module. In a simulation, we could easily use these identification systems but in real operation, we will probably need the surface operator to identify the module (e.g., vision may not be available underground).

Once the required module(s) are located, they must **get closer** to make possible the physical attachment. Depending on the motion capabilities and the specific mine condition, the main robot body may move towards the module(s) or the other way around.

This concept of *getting close* implies placing the **connection port** of the robot at a suitable distance from the connection port of the module. Therefore, the getting close stage triggers scenario 1: **go to location**, in this case, a location close enough to the required module.

Once both modules are near, the **coupling** starts. The coupling includes adequately **orienting** the ports of the robot and module, **approximation** of both ports, and **lock** both modules together. The coupling then needs to be **validated**: evaluated in terms of if both modules can adequately move together, share information, share energy, etc. depending on the specific module features.

It may be the case that the robot needs external help for the assembly process. The self-assembly capability may then be seen as a system-level capability and not a robot capability.

5.6.1. Scenario 6 Entities

- Robot
- Operator
- Mine
- Port: Part of a module or sub-module with coupling capabilities.
    - Active port: A port with motion/orientation capabilities
    - Passive port: A port without motion/orientation capabilities
- Module: Commonly, a type of node that represents an encapsulated, reusable, modular, and replaceable part of a system's software whose behavior is defined by interfaces. At a hardware level, a module is the basic structure of a robot, capable to move, composed of a body and legs. Modules can incorporate other modules or sub-modules to improve their capabilities. For self-assembly capabilities, modules have motion and identification skills.
- Sub-module: Module with an atomic function in the system. At a hardware level, when we think of sub-modules we picture legs, sensors, production tools, etc. which enhances a module's knowledge or capabilities. For self-assembly capabilities, modules have only identification skills.
- Location: 6D location of all module elements. For self-assemble, the main locations of interest are the active port of the module and sub-module.
    - Position: The 3D position part of the location.
    - Orientation: The 3D orientation part of the location.
- Map: The reference mine map shared by all subsystems.

5.6.2. Scenario 6 functions

- Module identification - Functionality and availability of a module or sub-module
- Module location - Identify position and orientation of the active port of a module or sub-module
- Move - Plan and navigate to module location [Go to location scenario]
- Orient - Move and turn mobile module to get an adequate position and orientation of active port respective to other port of a module or sub-module.
- Attach - Connect ports.
- Coupling – Establish functional coupling between the robot and element.
- Validate coupling - test coupling in terms of motion, communication, energy sharing, etc. depending on the specific coupling features

5.6.3. Scenario 6 variants

- All modules and submodules are thrown in the deposit via a large borehole (as in the project summary description).
- An agent (e.g., the meta-controller or surface operator) may decide a hardware reconfiguration is in need:
    - To overcome a failure the robot is unaware of.
    - To provide new system functionality.
    - To share energy
- Self-dissasembly: e.g. to replace a wear-out production tool.

5.6.4. Scenario 6 evaluation

The self-assembly scenario is complex because it may require robot capabilities –esp. motion– at the sub-robot level. This may imply that it cannot be demonstrated in the large robot RM1. On the other side, simulators may enable the complete scenario with the required complexity level. The possibilities of exploring this scenario in RM2 are bigger than in RM1 given its simpler operational specifications.

- RM1/RM2: Will perform **manually-assisted assembly**. A human operator will perform the movements and connections between ports. The meta-controller may decide the best architectural design for the run-time situation but it will be evaluated by the operator. The assembled modules will be the motion subsystem (body and legs) with sub-modules such as sensors or production tools and with other motion subsystems to create more complex miner robots.
- Simulation/RM2: Will perform **autonomous self-assembly**. The meta-controller will decide the best architectural design for the run-time situation. Then self-identification techniques will be used to find the desired (sub-)module. The motion module will autonomously navigate towards it. The assembled modules will be the motion subsystem (body and legs) with sub-modules such as sensors or production tools. The assembled modules will be the motion subsystem (body and legs) with sub-modules such as sensors or production tools and with other motion subsystems to create bigger miner robots. We will also vary the motion subsystem adding or changing the number of legs and its distribution.

5.7. Scenario 7: Mine Mapping

The robot is given the task of creating a mine map. This task may start from an empty map or may start from a prebuilt map created by mine engineers and geologists. In this scenario the robot moves in the mine and during movement it gathers geometrical and geophysical information that is used to update the mine map.

The mapping of the mine may have four base strategies:

1. Fully autonomous reactive mapping. The robot moves freely –e.g. using bioinspired movement patterns– and updates the mine map during the process.
2. Planned mapping. The system generates a movement and mapping plan from the information available –esp. the extant mine map– and the robot follows this plan.
3. Cooperative mapping. The robot uses reactive mapping –as in strategy 1– while receiving authoritative orders from the human operator.
4. Slave mapping. The operator uses the robot sensors for a teleoperated mapping of the mine.

The base scenario for mine mapping will be a **planned mapping scenario**.

The scenario starts in geology / mining that produces an initial mine map.

A planner generates an exploration plan for the extant mine considering robot location and capability.

The plan is supervised, approved, and ordered to the robot by the operator.

The robot receives the order and starts the execution of the task. The robot moves in the mine acquiring information that is incorporated to the mine map. During movement it uses reactive capabilities to handle local disturbances –see scenario 1.

The mine mapping scenario ends when:

- The mapping plan is completed.
- The scenario is stopped by the operator.
- There is an insurmountable circumstance (e.g. a blocking obstacle, a sensor problem, etc.).

5.7.1. Scenario 7 Variants

- Fully autonomous reactive mapping.
- Cooperative mapping. Collaborate with other robot in the updating of the mine map.
- Slave mapping. Create the map under a master-slave relation with other agent (e.g. the human operator).
- Find ore. Decide using mine/mining/geology knowledge were to go to start mining.
- Create spares map / Find spare. While the mine mapping scenario is mostly related with geometrical/geophysical aspects of the mine, the mine map may contain info on other elements of relevance (other robots, machines, spares, humans, recharging stations, etc.). The mine mapping activity may get info on all these elements and report them to the mine map managing agent.

5.7.2. Scenario 7 Functions

- Movement
- Ore sensing
- Geometry sensing
- Planning
- Operator communication

5.7.3. Scenario 7 Entities

- Mine
- Mine map
- Robot
- Operator
- Ore sensors
- Geometry sensors
- Mapping plan

5.8. Scenario Demonstration

As mentioned before, the scenarios described may be too general or not applicable to both RM1 and RM2 physical robots. These scenarios include research interests for different partners in the consortium and hence may address system aspects that are specific to some partner activities.

As the project develops, these scenarios will be particularized for the different activities in which they will be evaluated –e.g., the final demo in a real mine– and the platforms -physical robots and simulations- that will be used as proof of concept of the ROBOMINERS technologies.

The scenarios that have been described here point to some key activities that the system shall perform to achieve selective underground mining capability. These include movement and ore extraction but also some key features to increase the robustness and reliability of miner robots.

It is obvious that the complete scenarios described so far do not apply to all robots because due to the nature of the project –TRL4– some of the results will be only demonstrated in concept proofs. In any case, it is the ambition of the project to demonstrate as much capability as possible, but this shall not induce confusion concerning the concrete developments of each robot. The following table describes what scenarios will be demonstrated in what systems:

No.	Scenario Name	RM1	RM2	RM3	Rig	RMS1	RMS2	RMS3	Sim
1	Go to location	*	*	*		*	*	*	
2	Ore extraction	*				*	*		
3	Power assurance	*				*	*		
4	Fault detection	*	*			*	*		
5	Reconfiguration		*				*		
5.1	SW Reconfiguration	*				*	*		
6	Self-assemble						*		
7	Mine mapping	*	*	*		*	*	*	
7.1	Find ore					*	*		

RM1: Full scale robot (TAU)
RM2: Small scale robot (UPM)
RM3: Small scale robot (TalTech)
Rig: Ad hoc test rig
RMS1: RM1 simulator
RMS2: RM2 simulator
RMS3: RM3 simulator
Sim: Ad-hoc simulation needed for a specific evaluation.

6. ANNEX: SPECIFICATION OF INTERFACES

The selection of ROS/DDS as middleware opens the way to an early specification of system interfaces based on the architectural descriptions of the previous sections and the already available ROS basic specifications.

ROS applications typically use messages, services, and actions to communicate through interfaces. ROS2 is inspired by the Interface Definition Language (IDL), to describe these interfaces using three mechanisms:

- Message: .msg file to describe the fields of a ROS message
- Service: .srv file to describe a service (request and response)
- Actions: .action file to describe a goal, result and feedback

In this annex we summarily describe available ROS specification of interfaces and their use in the specification of some core Robominer interfaces. These descriptions are based on online ROS documentation[4].

6.1. Message description

Messages are defined in .msg files in the msg/ directory of a ROSS package. They are defined as a `field type field name`, for example:

```
int32 my_int

string my_string
```

Field names must be lowercase alphanumeric characters with underscores for separating words. They must start with an alphabetic character; they must not end with an underscore and never have two consecutive underscores. Default values can be defined as a third element in the definition:

```
fieldtype fieldname fielddefaultvalue

string full_name "John Doe"
```

Constant definitions are named in uppercase and the value is assigned with an equal sign

```
constanttype CONSTANTNAME=constantvalue

int32 X=123
```

The field types supported in ROS2 are:

[4] https://docs.ros.org/en/foxy/Concepts/About-ROS-Interfaces.html

Type name	C++	Python	DDS type
bool	bool	builtins.bool	boolean
byte	uint8_t	builtins.bytes*	octet
char	char	builtins.str*	char
float32	float	builtins.float*	float
float64	double	builtins.float*	double
int8	int8_t	builtins.int*	octet
uint8	uint8_t	builtins.int*	octet
int16	int16_t	builtins.int*	short
uint16	uint16_t	builtins.int*	unsigned short
int32	int32_t	builtins.int*	long
uint32	uint32_t	builtins.int*	unsigned long
int64	int64_t	builtins.int*	long long
uint64	uint64_t	builtins.int*	unsigned long long
string	std::string	builtins.str	string
wstring	std::u16string	builtins.str	wstring

6.2. Service description

Services are described and defined in .srv files in the srv/ directory of a ROS package. They are defined by a request and a response msg type separated by ---, for example:

```
string str
---
```

```
string str
```

Any two `.msg` files concatenated with a '---' are a legal service description, for example (if you want to refer to a message from the same package, you must not mention the package name):

```
#request constants
int8 FOO=1
int8 BAR=2
#request fields
int8 foobar
another_pkg/AnotherMessage msg
---
#response constants
uint32 SECRET=123456
#response fields
another_pkg/YetAnotherMessage val
CustomMessageDefinedInThisPackage value
uint32 an_integer
```

6.3. Action definition

Actions[5] are intended for long running tasks, they consist of three parts: a goal, feedback, and a result. Their functionality is similar to services, except the actions are preemptable (you can cancel them while executing). They also provide steady feedback, as opposed to services which return a single response.

Actions use a client-server model similar to publisher-subscriber topics. An "action client" node sends a goal to an "action server" the node that acknowledges the goal and returns a stream of feedback and a result.

Actions are specified by 3 sections, each of which is a message specification:

1. Goal -This describes what the action should achieve and how it should do it. It is sent to the action server when it is requested to execute an action.

[5] https://design.ros2.org/articles/actions.html

2. Result - This describes the outcome of an action. It is sent from the server to the client when the action execution ends, whether successfully or not.
3. Feedback - This describes the progress towards completing an action. It is sent to the client of the action from the action server between commencing action execution and prior to the action completing. This data is used by the client to understand the progress of executing the action.

Any of these sections may be empty. Between each of the three sections is a line containing three hyphens ---. Action specifications are stored in a file ending in .action. There is one action specification per .action file. For example:

```
# Define a goal of washing all dishes
bool heavy_duty  # Spend extra time cleaning
---
# Define the result that will be published after the action execution ends.
uint32 total_dishes_cleaned
---
# Define a feedback message that will be published during action execution.
float32 percent_complete
uint32 number_dishes_cleaned
```

The action server maintains a state machine for each goal it accepts from a client. Rejected goals are not part of the state machine.

There are three active states:

- ACCEPTED - The goal has been accepted and is awaiting execution.
- EXECUTING - The goal is currently being executed by the action server.
- CANCELING - The client has requested that the goal be cancelled, and the action server has accepted the cancel request. This state is useful for any user-defined “clean up” that the action server may have to do.

And three terminal states:

- SUCCEEDED - The goal was achieved successfully by the action server.
- ABORTED - The goal was terminated by the action server without an external request.
- CANCELED - The goal was cancelled after an external request from an action client.

State transitions triggered by the action server according to its designed behaviour:

- execute - Start execution of an accepted goal.
- succeed - Notify that the goal is completed successfully.

- abort - Notify that an error was encountered during processing of the goal and it had to be aborted.
- canceled - Notify that cancelling the goal completed successfully.

State transitions triggered by the action client:

- send_goal - A goal is sent to the action server. The state machine is only started if the action server accepts the goal.
- cancel_goal - Request that the action server stops processing the goal. A transition only occurs if the action server accepts the request to cancel the goal.

6.4. Some Common Interfaces for the Robominer implementation

6.4.1. Navigation

The navigation messages package[6] provides several messages and services for robotic navigation[7].

Messages (.msg)

- GridCells: An array of cells in a 2D grid.
- MapMetaData: Basic information about the characteristics of the OccupancyGrid.
- OccupancyGrid: Represents a 2-D grid map, in which each cell represents the probability of occupancy.
- Odometry: This represents an estimate of a position and velocity in free space.
- Path: An array of poses that represents a path for a robot to follow.

Services (.srv)

- GetMap: Get the map as nav_msgs/OccupancyGrid.
- GetPlan: Get a plan from the current position to the goal Pose.
- SetMap: Set a new map together with an initial pose.

6.4.2. Sensors

The sensors messages packages[8] provide many messages and services relating to sensor devices.

Messages (.msg)

- BatteryState: Describes the power state of the battery.
- CameraInfo: Meta information for a camera.

[6] https://github.com/ros2/common_interfaces/tree/master/nav_msgs
[7] More information about the navigation stack: https://navigation.ros.org/
[8] https://github.com/ros2/common_interfaces/tree/foxy/sensor_msgs. Many of these messages were ported from ROS 1 and a lot of still-relevant documentation can be found through the ROS 1 sensor_msgs wiki.

- ChannelFloat32: Holds optional data associated with each point in a PointCloud message.
- CompressedImage: A compressed image.
- FluidPressure: Single pressure reading for fluids (air, water, etc) like atmospheric and barometric pressures.
- Illuminance: Single photometric illuminance measurement.
- Image: An uncompressed image.
- Imu: Holds data from an IMU (Inertial Measurement Unit).
- JointState: Holds the data to describe the state of a set of torque controlled joints.
- JoyFeedbackArray: An array of JoyFeedback messages.
- JoyFeedback: Describes user feedback from a joystick, like an LED, rumble pad, or buzzer.
- Joy: Reports the state of a joystick's axis and buttons.
- LaserEcho: A submessage of MultiEchoLaserScan and is not intended to be used separately.
- LaserScan: Single scan with a planar laser range-finder.
- MagneticField: Measurement of the Magnetic Field vector at a specific location.
- MultiDOFJointState: Representation of state for joints with multiple degrees of freedom, following the structure of JointState.
- MultiEchoLaserScan: Single scan with a multi-echo planar laser range-finder.
- NavSatFix: Navigation Satellite fix for any Global Navigation Satellite System.
- NavSatStatus: Navigation Satellite fix status for any Global Navigation Satellite System.
- PointCloud2: Holds a collection of N-dimensional points, which may contain additional information such as normals, intensity, etc.
- PointField: Holds the description of one point entry in the PointCloud2 message format.
- Range: Single range reading from an active ranger that emits energy and reports one range reading that is valid along an arc at the distance measured.
- RegionOfInterest: Used to specify a region of interest within an image.
- RelativeHumidity: A single reading from a relative humidity sensor.
- Temperature: A single temperature reading.
- TimeReference: Measurements from an external time source are not actively synchronized with the system clock.

Services (.srv)

- SetCameraInfo: Request that a camera store the given CameraInfo as that camera's calibration information.

6.4.3. Diagnostics

The diagnostics messages[9] package provides several messages and services for ROS node diagnostics.

[9] https://github.com/ros2/common_interfaces/tree/foxy/diagnostic_msgs

Messages (.msg)

- DiagnosticArray: Used to send diagnostic information about the state of the robot.
- DiagnosticStatus: Holds the status of an individual component of the robot.
- KeyValue: Associates diagnostic values with their labels.

Services (.srv)

AddDiagnostics: Used as part of the process for loading analyzers at runtime, not for use as a standalone service

7. BIBLIOGRAPHY

OMG. (2019). *OMG Systems Modeling Language (OMG SysML™) Version 1.6.* Object Management Group.

ISO. (2011). *ISO/IEC/IEEE 42010-2011 – Systems and software engineering – Architecture description.* International Standards Organization.

ROBOMINERS. (2019). ROBOMINERS. Project no. 820971 Contract. Annex 1 - Description of Action (part A).

Walden, D. D., Roedler, G. J., Forsberg, K. J., & Hamelin, R. D. (2015). *INCOSE Systems engineering handbook. a guide for system life cycle processes and activities v4.0. Technical Publication INCOSE- TP-2003-002-04.*