MeROS: SysML-based Metamodel for ROS-based Systems - version 1.1.1 - reference manual

MeROS developers group - https://github.com/twiniars/MeROS
tomasz.winiarski@pw.edu.pl

September 6, 2023

Abstract

The complexity of today's robot control systems implies difficulty in developing them efficiently and reliably. Systems engineering (SE) and frameworks come to help. The framework metamodels are needed to support the standardisation and correctness of the created application models. MeROS is a metamodel for ROS, which addresses the running system and developer workspace. The ROS comes in two versions: ROS 1 and ROS 2. The metamodel includes both versions. In particular, the latest ROS 1 concepts are considered, such as nodelet, action, and metapackage. An essential addition to the original ROS concepts is the grouping of these concepts, which provides an opportunity to illustrate the system's decomposition and varying degrees of detail in its presentation. The metamodel is derived from the requirements and verified on the practical examples. The matter is described in a standardised way in SysML (Systems Modeling Language). Hence, common development tools that support SysML can help develop robot controllers in the spirit of SE.

Important citation notice

If you are to use MeROS in your papers, please first cite the IEEE ACCESS article [32], where the initial version of the MeROS is presented. You can also refer to MeROS project page [20] and mention the actual MeROS version you are using.

1 Introduction

The development of civilisation has led to an increase in the importance of robotics. Many modern robotic systems are complex. To create them as effectively and reliably as possible, it is necessary to follow systems engineering (SE), where metamodels play an essential role [2, 27, 16]. Robots, especially complex ones, are mostly controlled with usage of software. Hence, in robotics, SE is inextricably linked with software engineering, where frameworks have been crucial for many years [21, 29]. Diverse robotics frameworks have been developed so far [14, 30, 13]. Some steps towards standardisation have been made in recent years, and ROS (Robot Operating System) has come to the fore. Stand-alone ROS 1 (ROS version 1) [26] is unsuitable for hard RT (Real Time) systems, so one of the solutions in practical applications (e.g., [18, 5, 23, 28, 17, 8]), is to integrate ROS 1 with Orocos [3, 4]. Over time, ROS 1 has evolved to, among other things, improve its performance. Known and crucial problems in the face of some contemporary applications (e.g. cybersecurity, RT performance) led to the development of a new version of the framework, ROS 2 [19, 25].

ROS 1 has evolved considerably from the initial distributions. According to ROS metrics[1], new ROS 1 distributions have practically replaced older ones in terms of distro downloads stats. Above indications point to the need to formulate an up-to-date, recent metamodel for the latest versions of

[1] https://discourse.ros.org/t/2022-ros-metrics-report/29594

ROS 1 and ROS 2, which this documentation undertakes by presenting the new metamodel for ROS – MeROS.

The robotic models can be subdivided [1] into Platform Independent Models (PIM), e.g., [36, 35, 9, 34], and Platform Specific Models (PSM). The metamodels of ROS, including MeROS, belong to PSM and should answer to the component nature of ROS [10, 31]. MeROS is founded on SysML (Systems Modeling Language) [22, 11], a profile of UML (Unified Modeling Language). Modelling in languages from the UML family addresses a number of important aspects of systems engineering [7]. These include the use cases [UCX]:

- [UC1] Systems' documentation and presentation,
- [UC2] Effective analysis of systems, especially in interdisciplinary teams (graphical language is more understandable for non-specialists in the field),
- [UC3] Defects detection,
- [UC4] Integration of new collaborators into the development team,
- [UC5] Resuming work after a break,
- [UC6] Extension and modification of existing systems,
- [UC7] Support the implementation of new systems,
- [UC8] Migration of systems.

In practice, documentation is created both prior to implementation and, in many cases, through a process of reverse engineering [6] for existing systems. Agile-type strategies involve modifying the documentation as the project develops [12].

The following presentation starts with formulating the requirements (sec. 2) for the MeROS metamodel. These requirements are allocated to the metamodel that is described in sec. 3. The way to present a model of a specific application based on MeROS is presented on a practical example in sec. 4.

2 Metamodel requirements

The requirements [RX] formulation process for MeROS metamodel is multi-stage and iterative. In the beginning, the initial requirements were formulated based on: (i) literature review (both scientific and ROS wiki/community sources), (ii) author experience from supervising and supporting ROS-based projects, and finally, (iii) author experience from EARL (Embodied Agent-based cybeR-physical control systems modelling Language) [34] PIM development and its applications (e.g. [9, 15, 33]). Verification of subsequent versions of MeROS by its practical applications led to an iterative reformulation of requirements and MeROS itself.

MeROS requirements are depicted on a number of dedicated SysML diagrams. The requirements are organised in a tree-like nesting structure, with additional internal relations, and labelled following this structure. The general requirements are presented in Fig. 1. Here, and in the following diagrams, the elements (components, relations) specific for a particular version of ROS (ROS 1 or ROS 2) are labelled with an „rv" tag with the ROS version that the element is specific for. The lack of a tag means the element is general for both ROS versions.

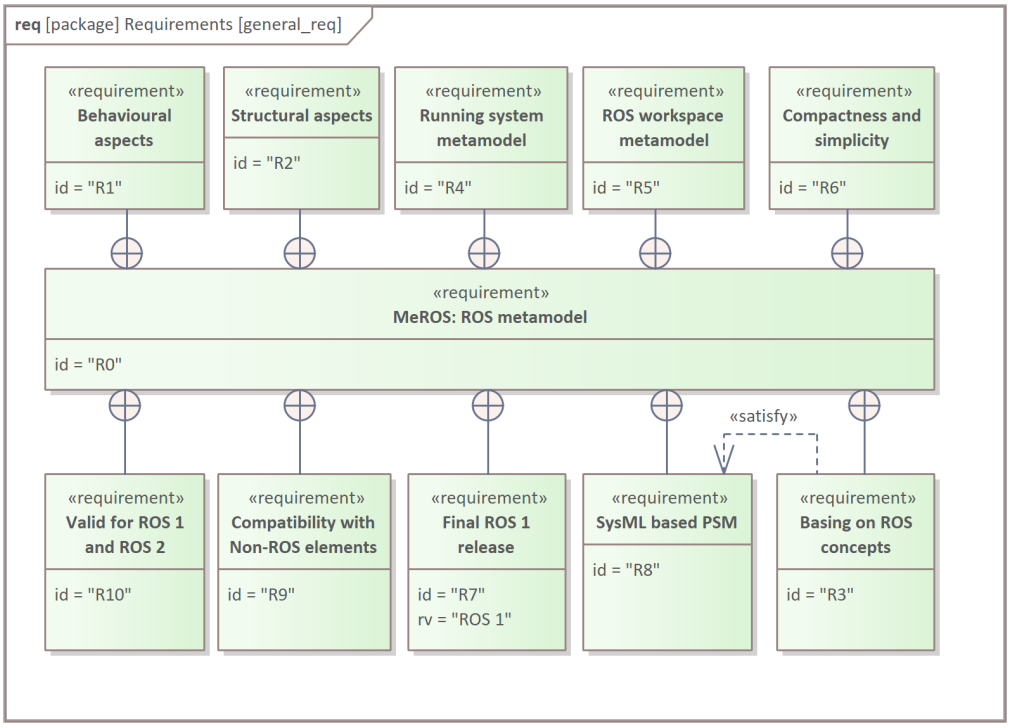

Figure 1: General requirements.

The SysML models have two main parts: behavioural [R1] and structural [R2]. MeROS aims to cover ROS concepts [R3] and not change their labels as long as possible, to maintain conformity and intuitiveness. The ROS system is two-faced. While it is executed [R4], it has a specific structure and behaviour, but from the developers' point of view, the workspace [R5] is the exposed aspect. The model should be compact and straightforward [R6] rather than unnecessarily elaborate and complicated. One of the assumptions that stand out MeROS from other ROS metamodels is conformity with the final ROS 1 release [R7] (Noetic Ninjemys). Although the SysML-based MeROS is classified into PSM [R8], it should be compatible with Non-ROS elements [R9]. Finally, MeROS metamodel should be valid both for ROS 1 and ROS 2.

The system's structural aspects requirements are presented in Fig. 2.

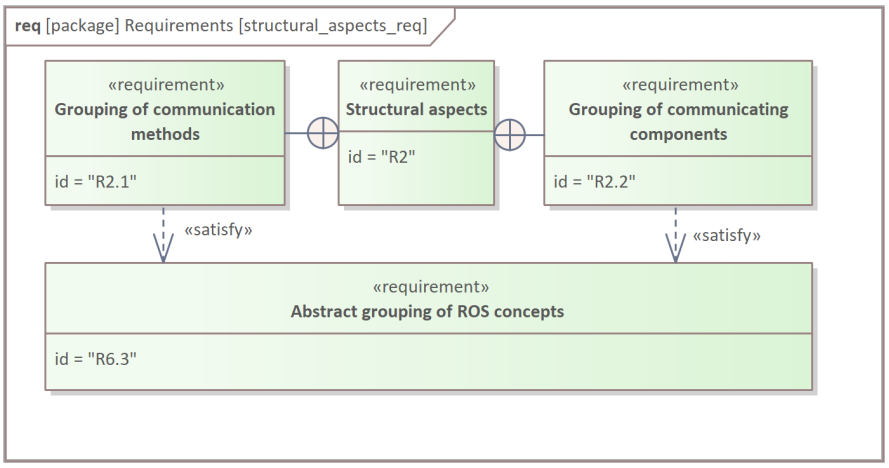

Figure 2: Structural aspects requirements.

A vital addition to the original ROS concepts is the abstract grouping of: (i) communicating methods [R2.1] and (ii) communicating components [R2.2]. The motivation for the introduction of these aggregates is presented further on. It should be noted that several ROS concepts group other concepts in a particular way, especially to deploy the system. Action aggregates Topics and Services (in ROS 2), ROS 1 Node aggregates Nodelets, and ROS 2 Component Container aggregates Nodes. The ROS concepts that MeROS models are organised into four major classes (Fig. 3): (i) Communicating components [R3.1], (ii) Communication methods [R3.2], (iii) Workspace [R3.3], and (iv) Other [R3.4].

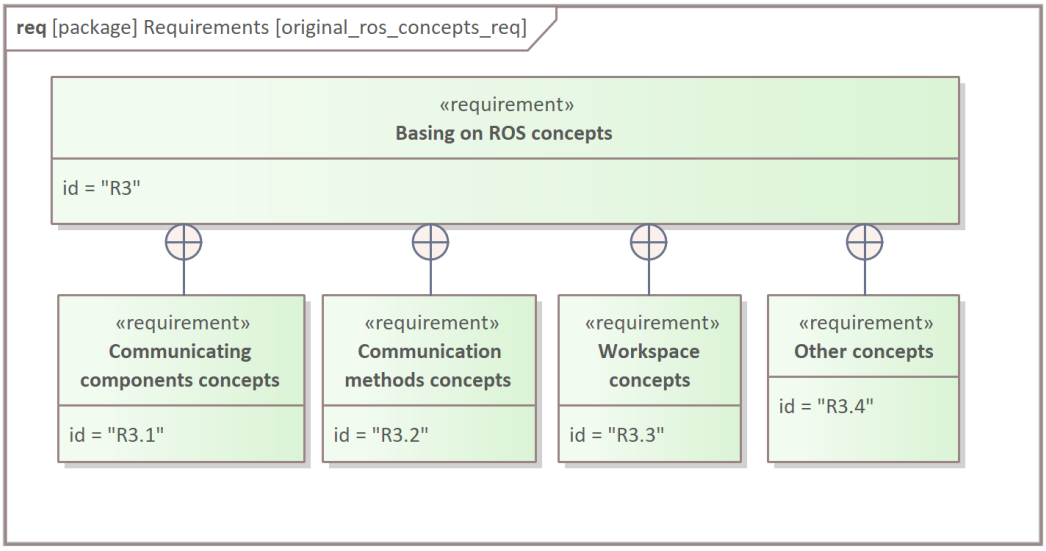

Figure 3: ROS concepts requirements.

Communicating components [R3.1] are: (i) ROS Node, (ii) ROS Nodelet, (iii) ROS plugin, and (iv) ROS library. Both plugin and library let to share the same code between various Nodes or ROS 1 specific Nodelets. Two ROS nodes are mandatory to execute the ROS 1 system: (i) ROS Master and (ii) rosout. Three methods of communication are considered [R3.2] with their inter-component connections and data structures: (i) ROS Topic, its Message and connection, (ii) ROS Service comprising data structure and connection, and finally (iii) ROS Action including data structure and connection.

Workspace concept [R3.3] comprises: (i) ROS Package [R3.3.1], (ii) Metapackage [R3.3.2] introduced in the latest releases of ROS 1, (iii) Group of packages [R3.3.3] and (iv) Repository [R3.3.4]. Other concepts [R3.4] include four elements: (i) ROS Parameter Server manages (ii) ROS Parameters, (iii) roscore forms a collection of programs and nodes that are pre-requisites of a ROS 1-based system. Finally, (iv) ROS Namespace reflects the ROS concept to organise nodes and communication connections. Both ROS Master and rosout are executed with roscore. ROS Parameter Server is a part of ROS Master.

To achieve intuitiveness, MeROS presents a Running system structure (Fig. 4) following ROS rqt_graph pattern [R4.1]. In particular, there are two ways to visualise communication, including [R4.1.1] and without [R4.1.2] dedicated communication components.

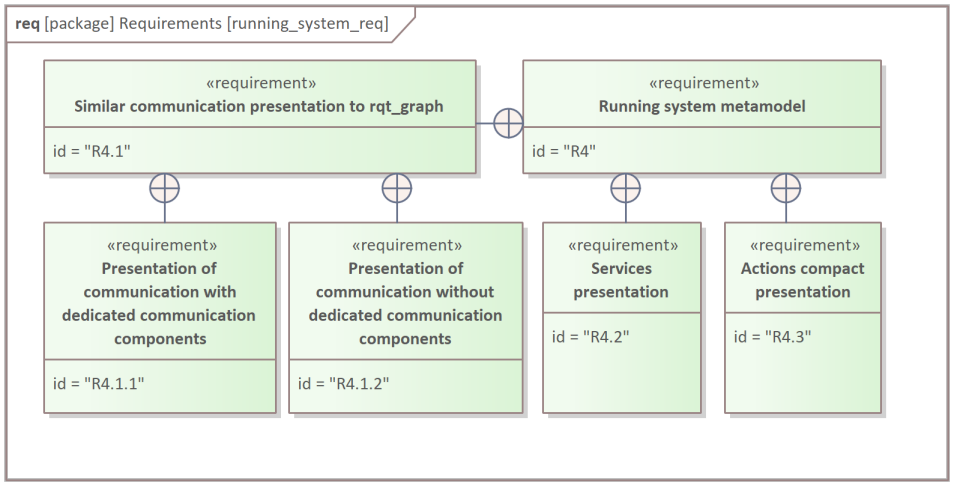

Figure 4: Running system requirements.

The dedicated components are especially useful in the presentation when many communication components use the same topic both on the publisher and the subscriber side. In opposition, the expression of topic names on arrows connecting communicating components, i.e., without dedicated communication components, let to reduce the number of components needed to depict communication for many topics and a low number of communicating components. The other advantage of using dedicated communication components is that the particular connection can be split into several diagrams (e.g. ibd (internal block diagram) or sd (sequence diagram)), where the same object represents this connection in every associated diagram. Services [R4.2] and actions [R4.3] should be depicted as an addition to the presentation of the particular topics. It should be noted that rqt_graph represents actions as a number of topics and services. In MeROS, the topics and services being part of an action can be aggregated, which reduces the number of depicted connections.

The compactness and simplicity [R6] and its nesting requirements are presented in Fig. 5.

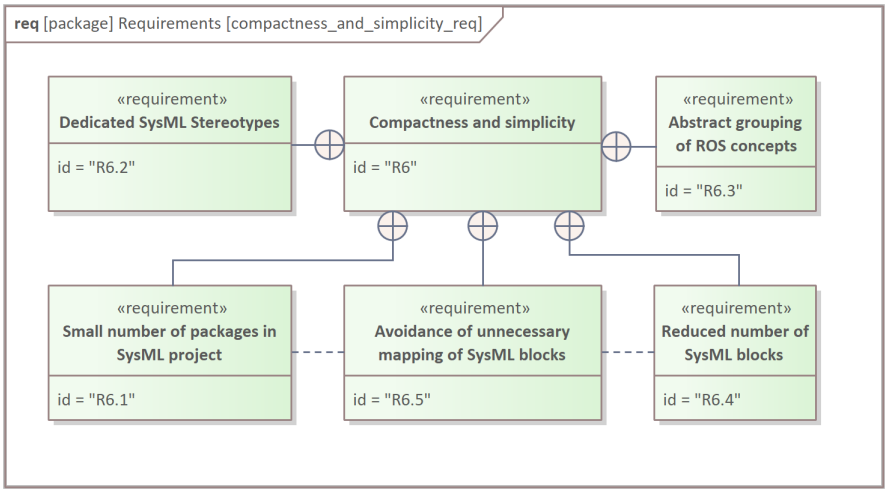

Figure 5: Compactness and simplicity requirements.

A SysML project to develop and represent MeROS metamodel should consist of a small number of packages [R6.1], but still, the packages should distinguish the major aspects of development process: (i) metamodel requirements formulation, (ii) metamodel itself, and (iii) metamodel realizations/applications. Dedicated SysML stereotypes [R6.2] are introduced to MeROS to replace the direct block specialization representation on diagrams and improve the legibility and compactness of diagrams. The grouping of concepts [R6.3] has diverse aims. It enables the presentation of the system part in a general, PIM-like abstract way, on the logical level rather than a detailed, PSM-like implementation one. The aggregation reduces the number of objects represented on the diagram to highlight the essential aspects and stay compact and consistent in presentation. The number of SysML blocks should be reduced to a reasonable level [R6.4]. Both [R6.1] and [R6.4] help in the Avoidance of unnecessary mapping of SysML blocks [R6.5].

There are three elements in the requirements set that satisfy the evolution of the ROS 1 finalised with its ultimate release – Noetic Ninjemys – (Fig. 6): (i) ROS Nodelet (introduced primarily to increase the efficiency of ROS components switching), (ii) ROS Action , and (iii) ROS Metapackage.

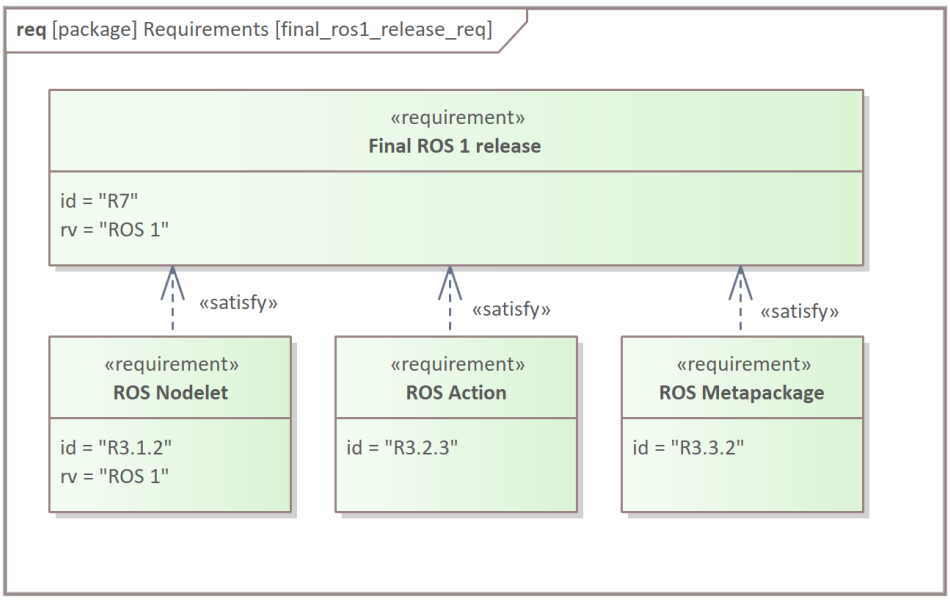

Figure 6: Final ROS 1 release requirements.

3 MeROS metamodel

MeROS metamodel is formulated according to the requirements discussed in the previous section. Sec. 3.1 presents MeROS blocks' structural composition, and sec. 3.2 describes inter-component communication. From the metamodel perspective, the structural aspects [R2] are formulated in both sections, while behavioural [R1] is in the latter. The diagrams comprise selected requirements being allocated to expose the MeROS metamodel development process.

The MeROS diagrams were created in the Enterprise Architect development tool within the SysML project [R8] and organised in three packages [R6.1] (Fig. 7): (i) Requirement Model related to requirements formulation and analysis, (ii) MeROS – the metamodel itself, (iii) Rico Controller – the exemplary ROS 1 application of MeROS described in sec. 4.2. The stereotypes are introduced in MeROS metamodel with the dedicated MeROS profile [R6.2].

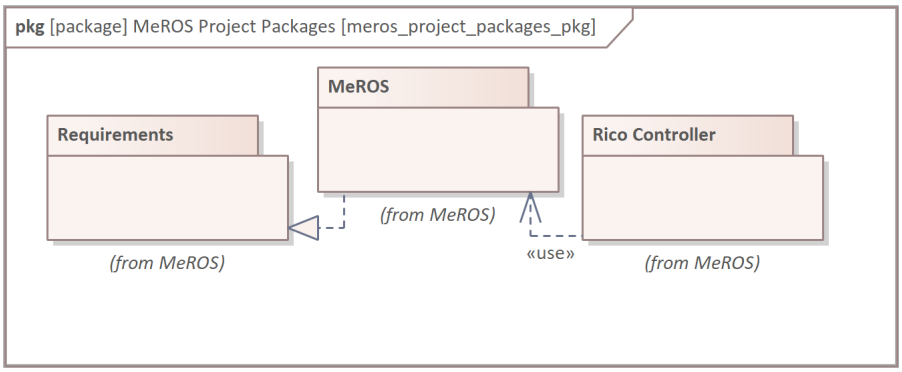

Figure 7: MeROS project SysML packages, where Rico Controller is an exemplary realisation of MeROS metamodel.

3.1 Metamodel composition

The degree of specificity of a metamodel is a compromise between its comprehensiveness (and, therefore, more general formulation) and a more accurate representation of a particular subclass of specific implementations. The metamodel contains compositions of elements and other primary relationships. Attributes and operations range widely, in particular between ROS 1 and ROS 2. Hence, their inclusion would lead to overgrowth and complication of the metamodel [R6]. Models derived from the metamodel can define their operations and new relations specific to a particular system.

The SysML blocks reflect ROS concepts [R3], and their composition is depicted in bdd (block definition diagrams). The metamodel is formulated in a single SysML package. Hence, Groups of Packages (especially Workspaces) and Intrasystems are composed into ROS System (Fig. 8).

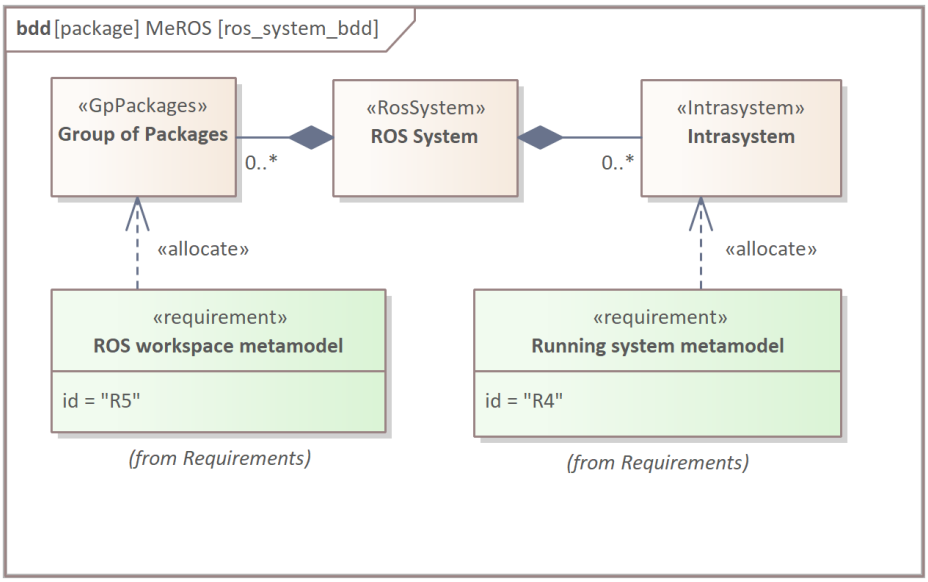

Figure 8: ROS system general composition – bdd.

Consequently, some concepts (e.g., Node) occur in Workspaces and Intrasystems. It reduces the number of SysML blocks in the metamodel [R6.4] and eliminates the need for unnecessary mapping of SysML blocks [R6.5].

In MeROS, a Communicating Component (Fig. 9) is a crucial abstraction of a number of ROS concepts to represent their standardised role regarding communication.

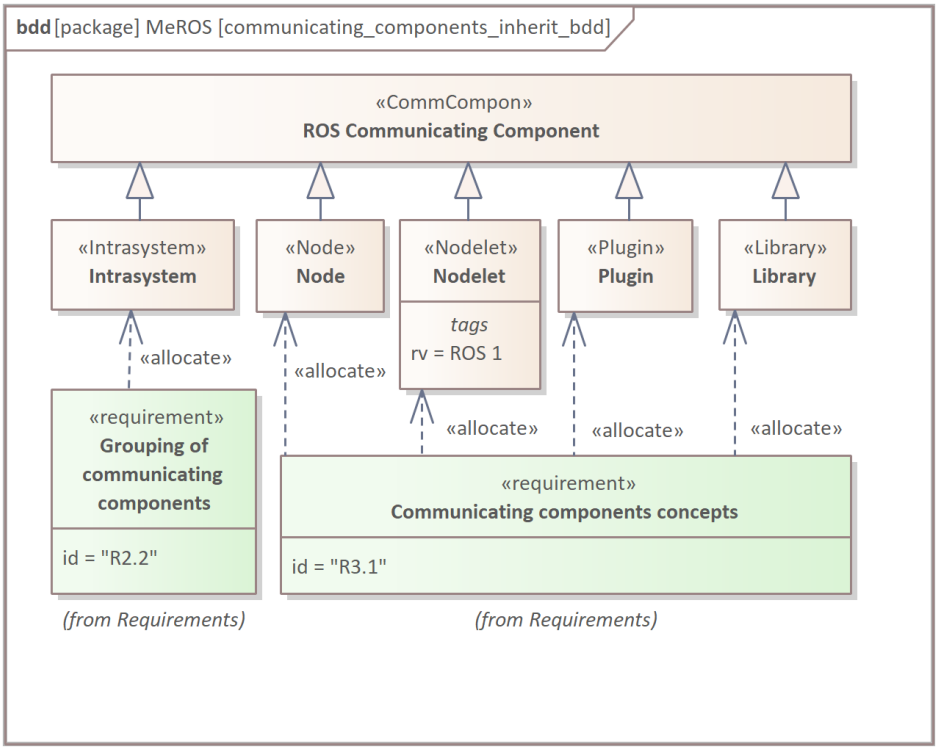

Figure 9: Communicating Component and specialised blocks – bdd.

It should be noted that behavioural aspects of a particular model specified in MeROS can be formulated by operation specification as an act (activity), sd (sequence), or stm (state machine) diagrams. The Intrasystem is one of the aggregates added to the base ROS concepts in MeROS.

For clarity, relations of Communicating Components are depicted in several diagrams. Fig. 10 considers Topics and their Data Structures. Here, the Communicating Component can act as a publisher or a subscriber.

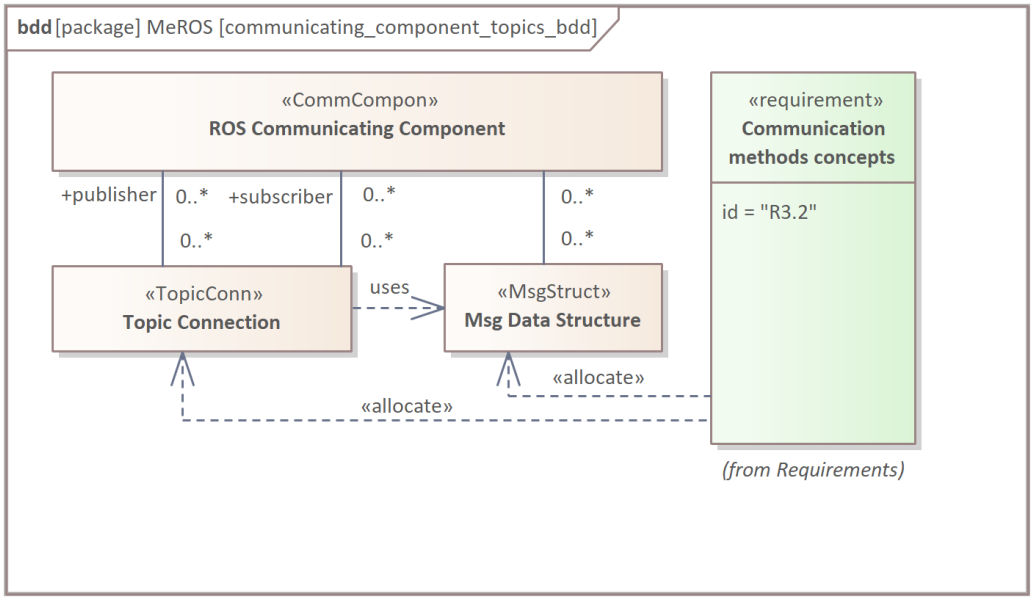

Figure 10: Communicating Component relations – topics – bdd.

Fig. 11 depicts Services and their Data Structures. In this case, the Communicating Component can act as a server or a client.

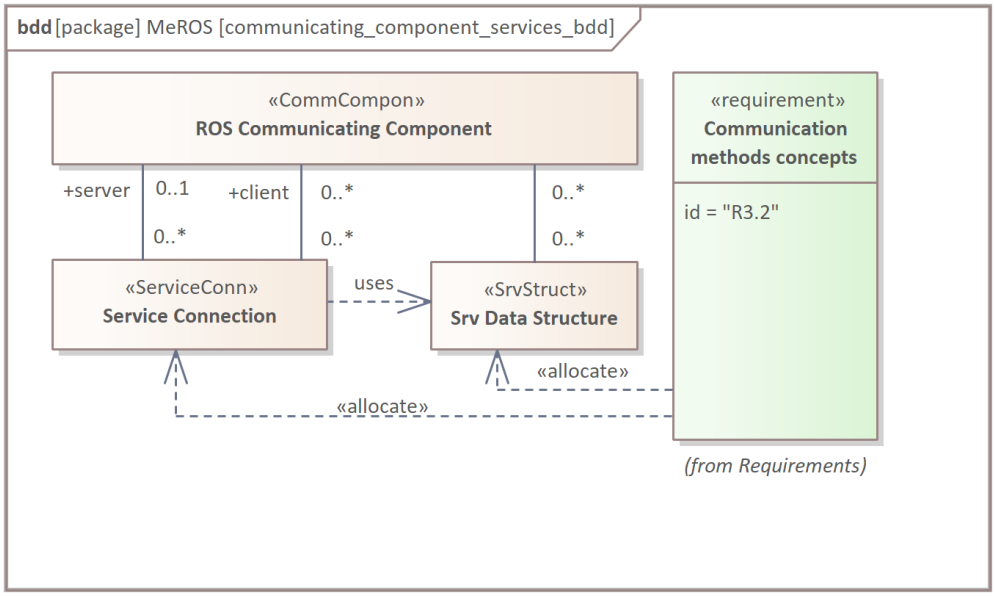

Figure 11: Communicating Component relations – services – bdd.

In ROS 2, an Action bases on Topics and Services while in ROS 1 only on Topics. From functional point of view, ROS 1 specific Topics included in Action have their equivalents in ROS 2 specific Services.

The Actions are depicted in two diagrams – Fig. 12 and Fig. 13. Similarly to Services, the Communicating Component can act as a server or a client.

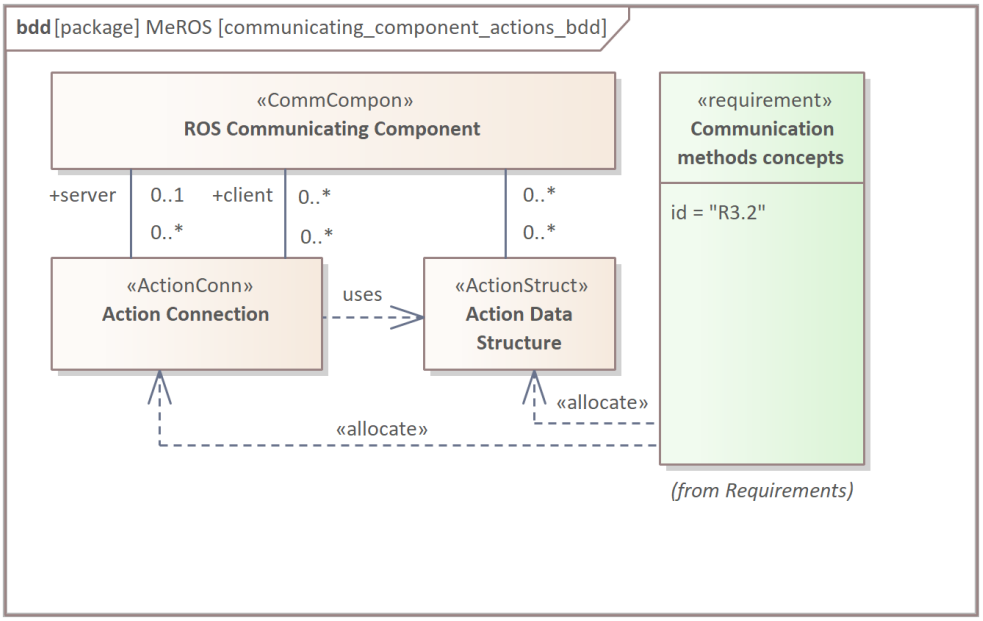

Figure 12: Communicating Component relations – actions – bdd.

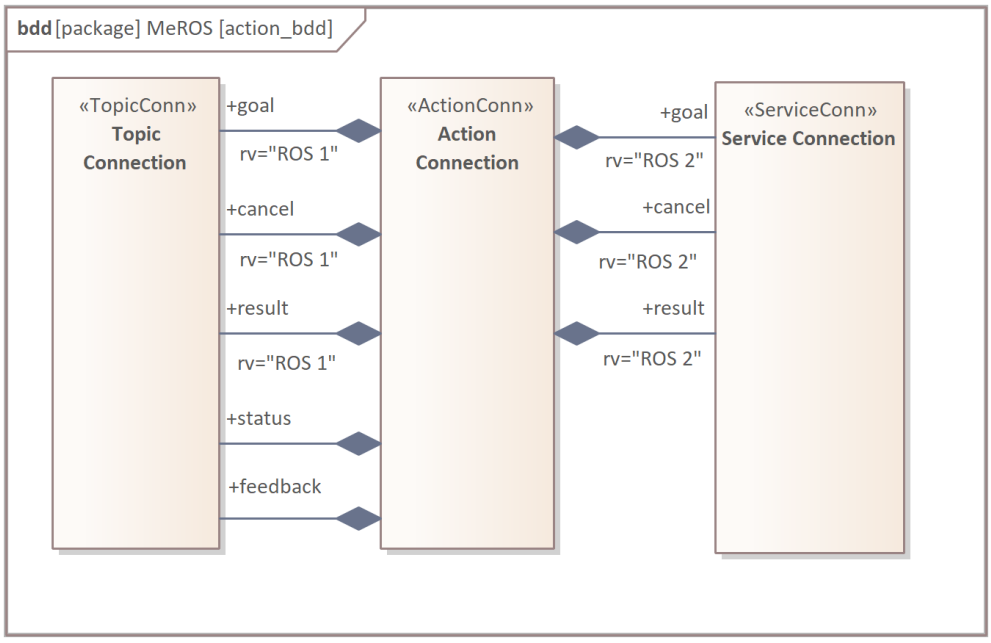

Figure 13: Action – bdd.

Fig. 14 describes how Non-ROS elements are taken into account in relation to communication. Additionally, the figure presents Communication Channel relation to ROS Communicating Component.

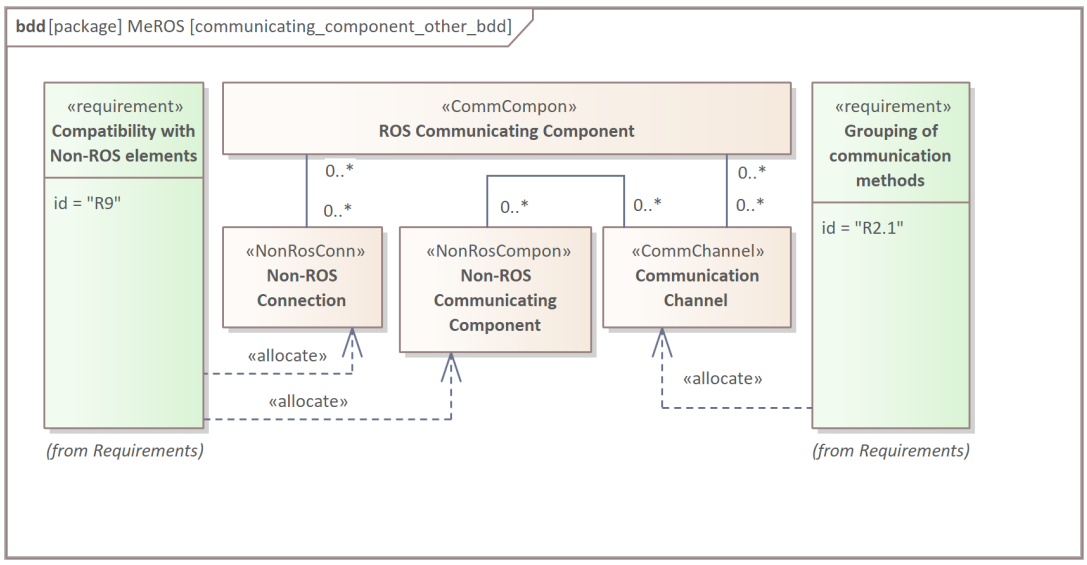

Figure 14: Communicating Component relations – aggregates, Non-ROS elements – bdd.

Besides standard ROS communication methods, the Non-ROS are also included (e.g., http request) to achieve interfaces with Non-ROS parts of the general system. An Action Data Structure comprises data used by three of five Topics composed in Action, i.e., goal, feedback and result. Two remaining Topics, i.e., cancel and status are standardised.

The Communication Channel [24] concept depicted in Fig. 15 is introduced to aggregate specializations of ROS connection (Topic connections, Service connections, and Action connections) as well as Non-ROS Connections. The Communication Channel can also aggregate different Communication Channels.

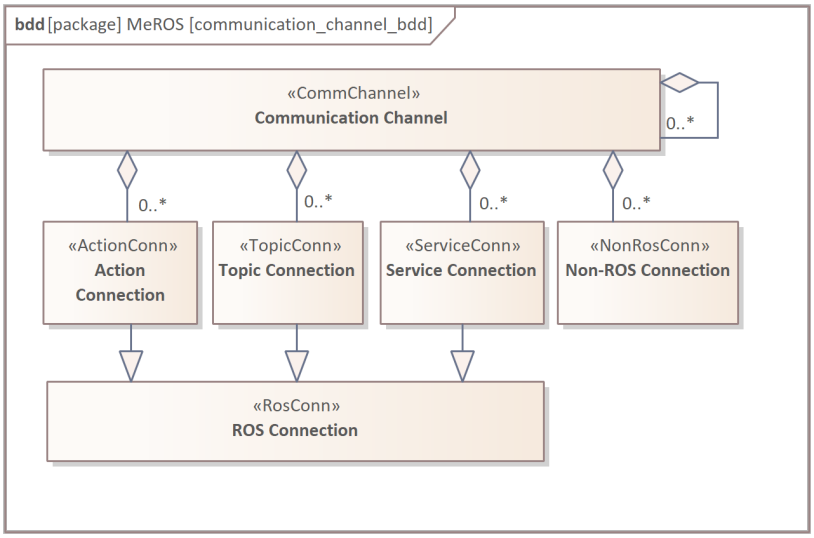

Figure 15: Communication Channel – bdd.

The Node (Fig. 16) composes Parameters and Nodelets (the latter in ROS 1). Two specific Nodes are considered in the metamodel: ROS Master and rosout. In ROS 2, Component Container aggregates Nodes executed in a single process.

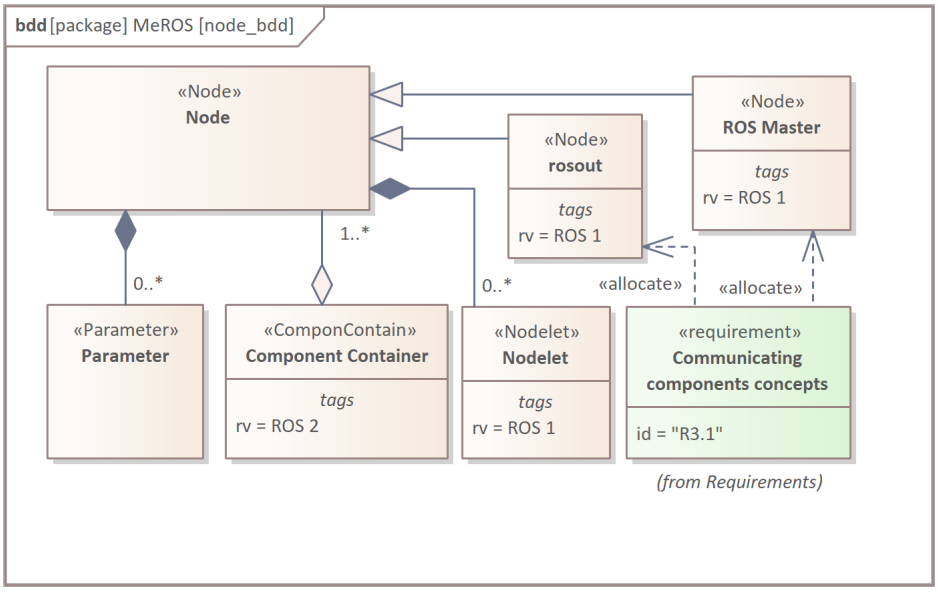

Figure 16: Node – bdd.

The Intrasystem (Fig. 17) composes ROS and Non-ROS Communicating Component specializations as well as Connections between them. A Parameter block is introduced also for ROS 1. In ROS 2, due to safety reasons, Parameter is composed only into Nodes. Optionally Intrasystem composes the other Intrasystems.

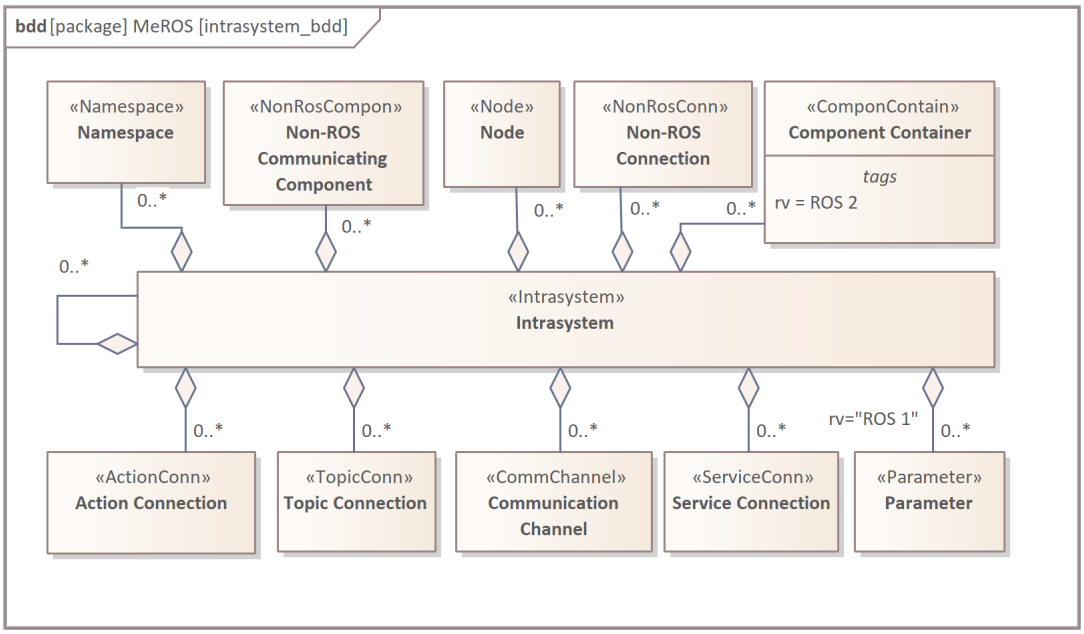

Figure 17: Intrasystem compositions – bdd.

The Running System (Fig. 18) is a specialisation of the Intrasystem that can be executed. Hence, two Nodes are needed for ROS 1: rosout and ROS master. It should be noted that although MeROS could be classified as PSM, the initial, general system description with Communications Channels and Intrasystems corresponds to PIM specification. Then, the detailing of these aggregates corresponds to the transition from PIM to PSM.

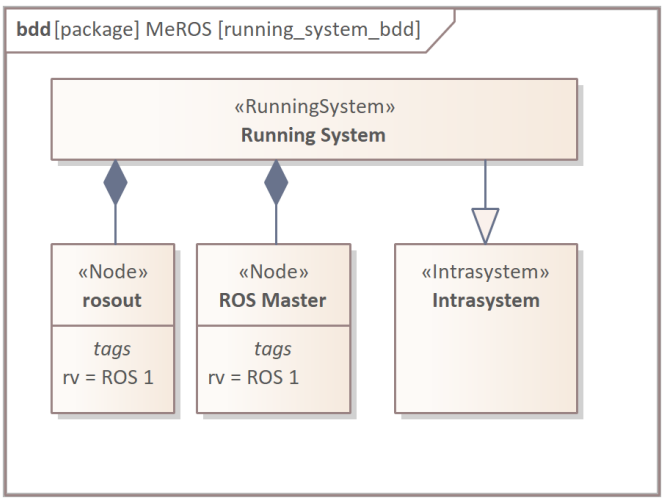

Figure 18: Running System compositions – bdd.

The way Communicating Components use various types of connections is presented in Fig. 19. Both ROS and Non-ROS Communicating Components can communicate via Non-ROS Connections, but only ROS Communicating Components use ROS Connections.

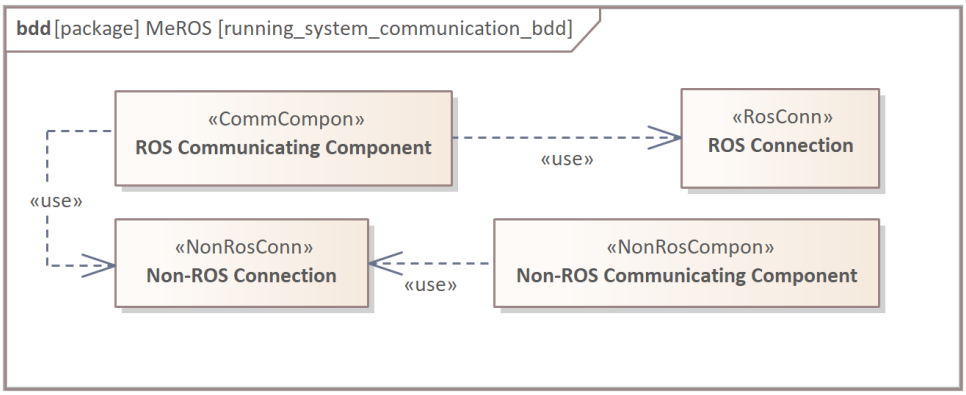

Figure 19: Running System communication – bdd.

The Namespace (Fig. 20) aggregates elements of the Intrasystem, but only ROS related. In opposition to the Intrasystem, the Namespace does not specialise Communicating Component. Hence, it can not act as Communicating Component.

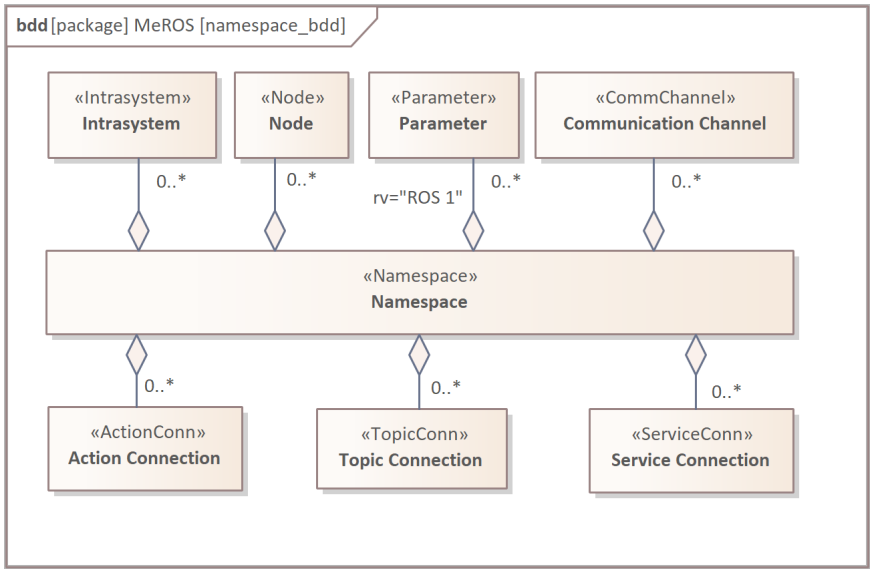

Figure 20: Namespace composition – bdd.

The Package (Fig. 21) composes the files related to general ROS concepts such as Node source codes, communication structures definitions, etc. It should be noted that in case of Actions, specific communication structures definitions are stored in Action Data Structures. The Misc «block» relates to other ROS and Non-ROS files, e.g., roslaunch configuration, obligatory package.xml, CMakeLists.txt. The Metapackage is introduced for conformity with the latest ROS 1 releases [R7] as well as ROS 2.

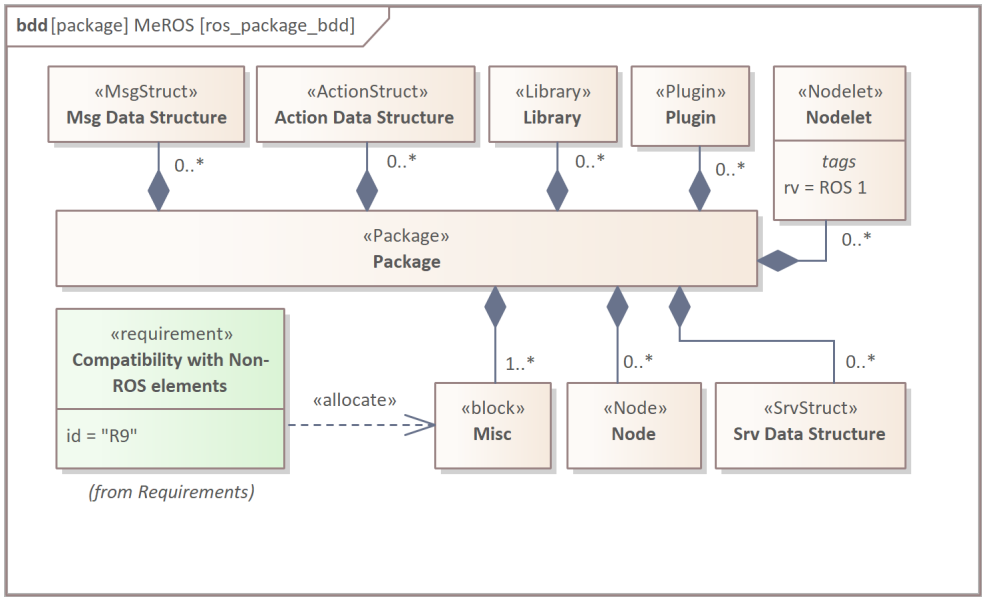

Figure 21: ROS Package composition – bdd.

The Workspace (Fig. 22) contains of Packages that compose the files related to general ROS concepts such as Node source codes, communication structures definitions, etc. As Workspace has a specific file-system nature, Group of Packages were introduced as a more general component. The Repository plays a similar role to the Workspace, but reflects the place where sources are stored.

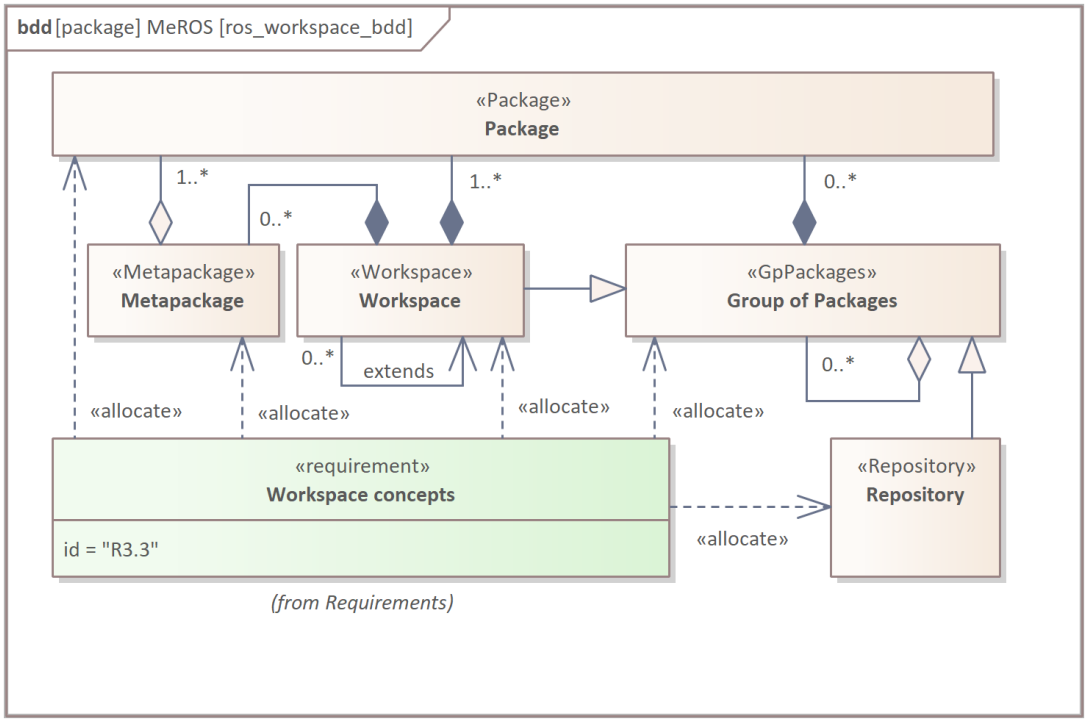

Figure 22: ROS Workspace composition – bdd.

3.2 Communication

This section depicts the behavioural and structural aspects of communication in the system. The previous section considers block definition diagrams (bdd). In the following part, the internal block diagrams (ibd) and behavioural diagrams are discussed. The goal is to present three modes of communication: Topic [R4.1] (sec. 3.2.1), Service [R4.2] (sec. 3.2.2) and Action [R4.3] (sec. 3.2.3). It should be noted that the concept of presentation of communication with and without a dedicated communication component is illustrated on communication with Topics but can also be applied to Services, Actions and Communication Channels.

3.2.1 Topic

Fig. 23 presents the ibd diagram of publishers' and subscribers' communication via topics. This diagram uses a dedicated communication component for each Topic [R4.1.1]. There are no general limits to the number of publishers, subscribers and Topics they communicate with.

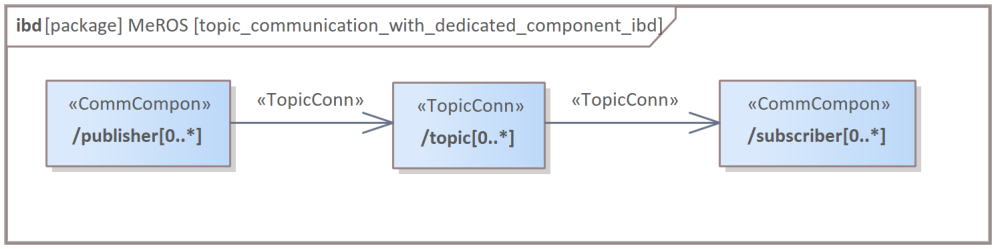

Figure 23: Topics with dedicated communication components – all components – ibd.

Thanks to a dedicated component to represent communication, the diagram in Fig. 23 can be split into two considering publisher (Fig. 24) and subscriber (Fig. 25) separately, without losing information. It is especially useful when system fragments are presented after its decomposition that subdivides communication channels.

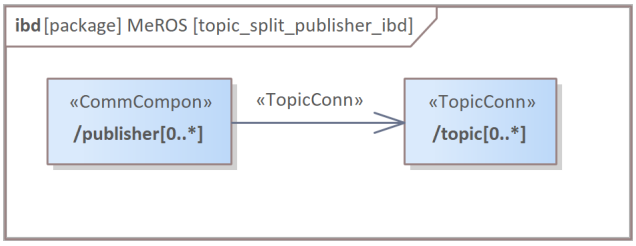

Figure 24: Topics with dedicated communication components – publisher – ibd.

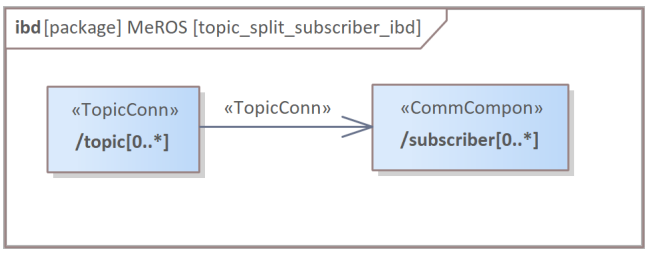

Figure 25: Topics with dedicated communication components – subscriber – ibd.

Fig. 26 depicts the corresponding sequence diagram. Publishers send a message through Topics to the subscribers. The incoming message cause the subscriber to execute the callback function. Fig. 27 and Fig. 28 present an alternative approach to depict the system communicating via topics. In this case, no dedicated communication components are used [R4.1.2].

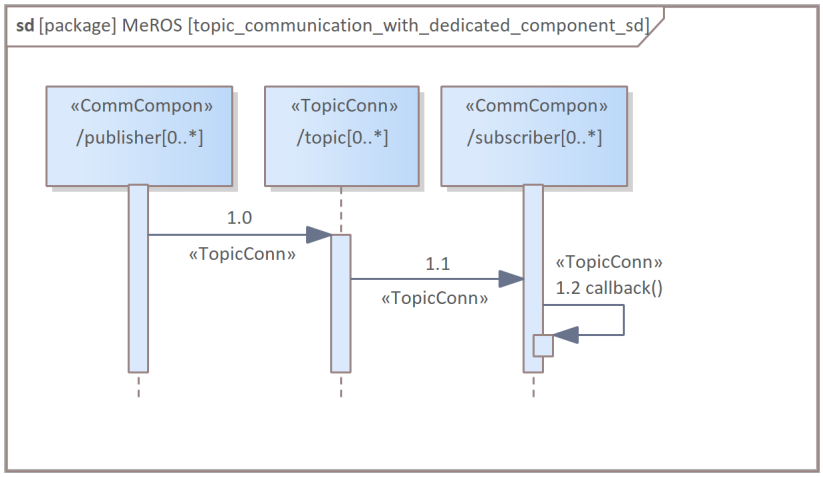

Figure 26: Topics with dedicated communication components – sd.

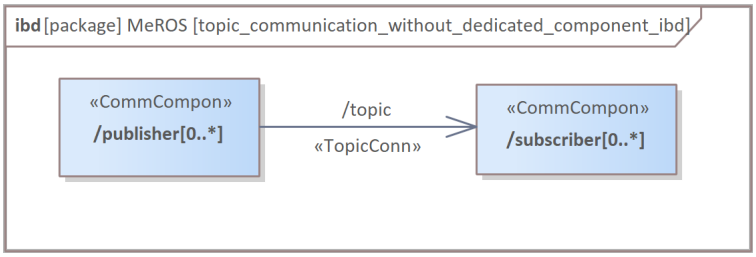

Figure 27: Topics without dedicated communication components – ibd.

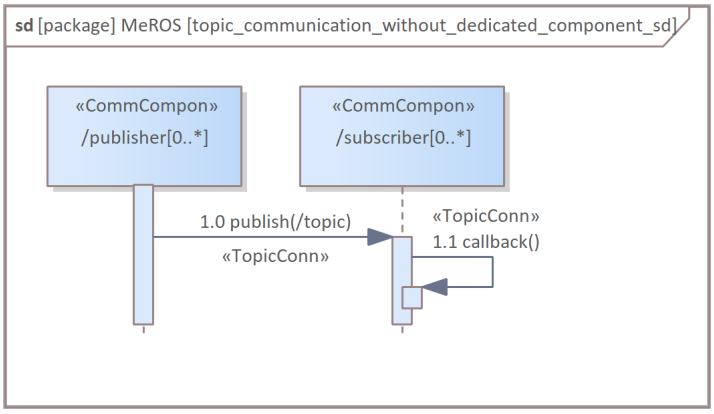

Figure 28: Topics without dedicated communication components – sd.

3.2.2 Service

For each ROS Service, there is at most one server and a number of clients (Fig. 29 and Fig. 30). Service-type communication is bidirectional and realises RPC (remote procedure call).

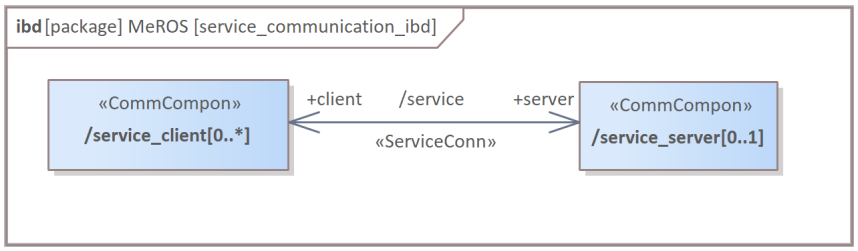

Figure 29: Service-based communication – ibd.

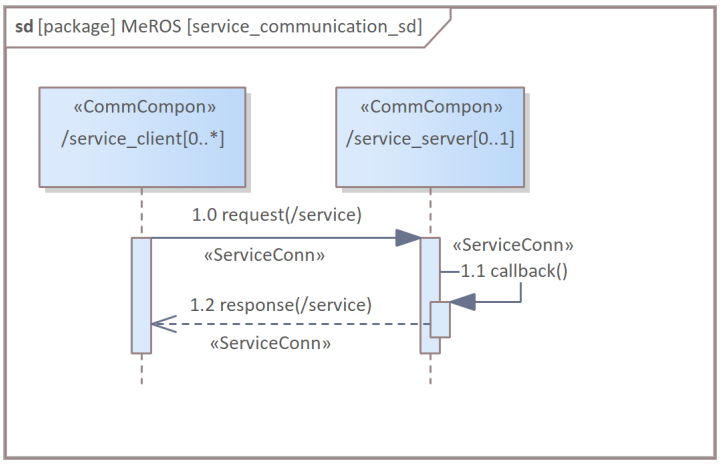

Figure 30: Service-based communication – sd.

3.2.3 Action

ROS Action communication's general, simplified structure (Fig. 31) is analogous to ROS Service. These type of presentation is universal for ROS 1 and ROS 2.

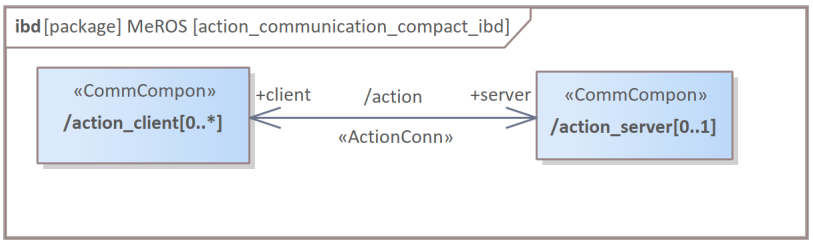

Figure 31: Action-based communication – compact representation – ibd.

An Action (Fig. 32) is based on several Topics in ROS 1, while on Topics and Services in ROS 2.

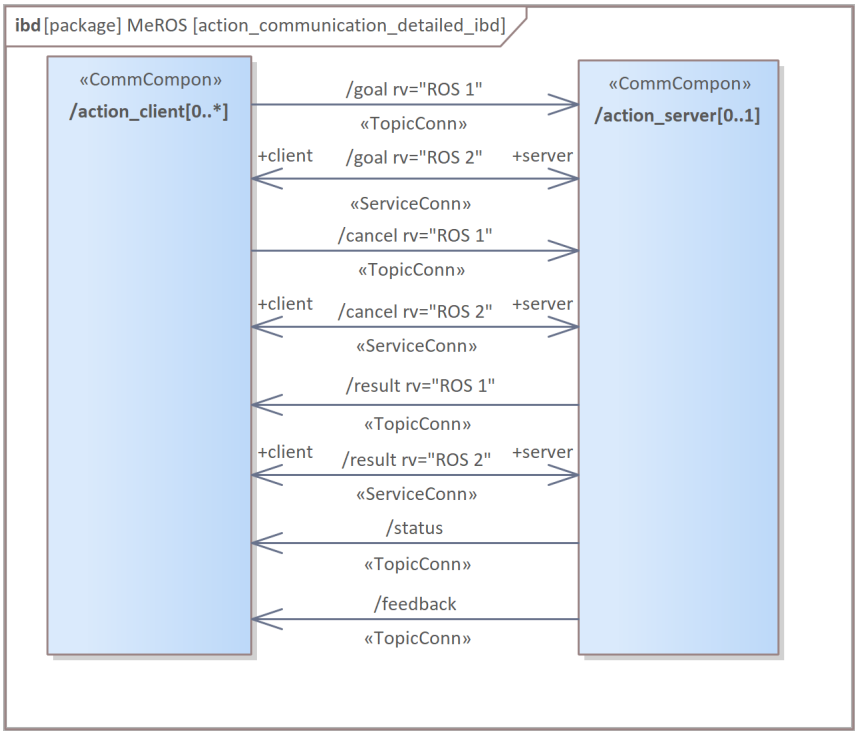

Figure 32: Action-based communication – detailed – ibd.

In practice, to present an action-related communication compactly on sd diagram (Fig. 33) particular Topics and Services can be generalised as a request (for /goal and /cancel) and a response (for /status, /feedback and /result). It should be noted that this diagram presents the Action communication sequence in a simplified way.

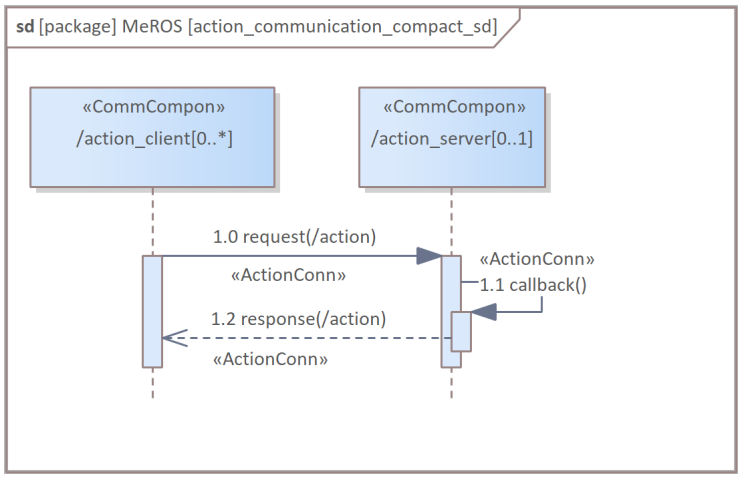

Figure 33: Action-based communication sequence – compact presentation – sd.

The detailed behaviour of the Action server and Action client in ROS 1 is specified by state machines [2]. ROS 2 Action server and Action client behaviour is analogous. Here, these two state machines are depicted in stm diagrams. In the description, in addition to the original ROS wiki presentation, the Topics are directly mentioned both in transitions and states actions. Fig. 34 depicts the ROS 1 Action server state machine. Its transitions depend on the new messages sent by the Action client or internal predicates.

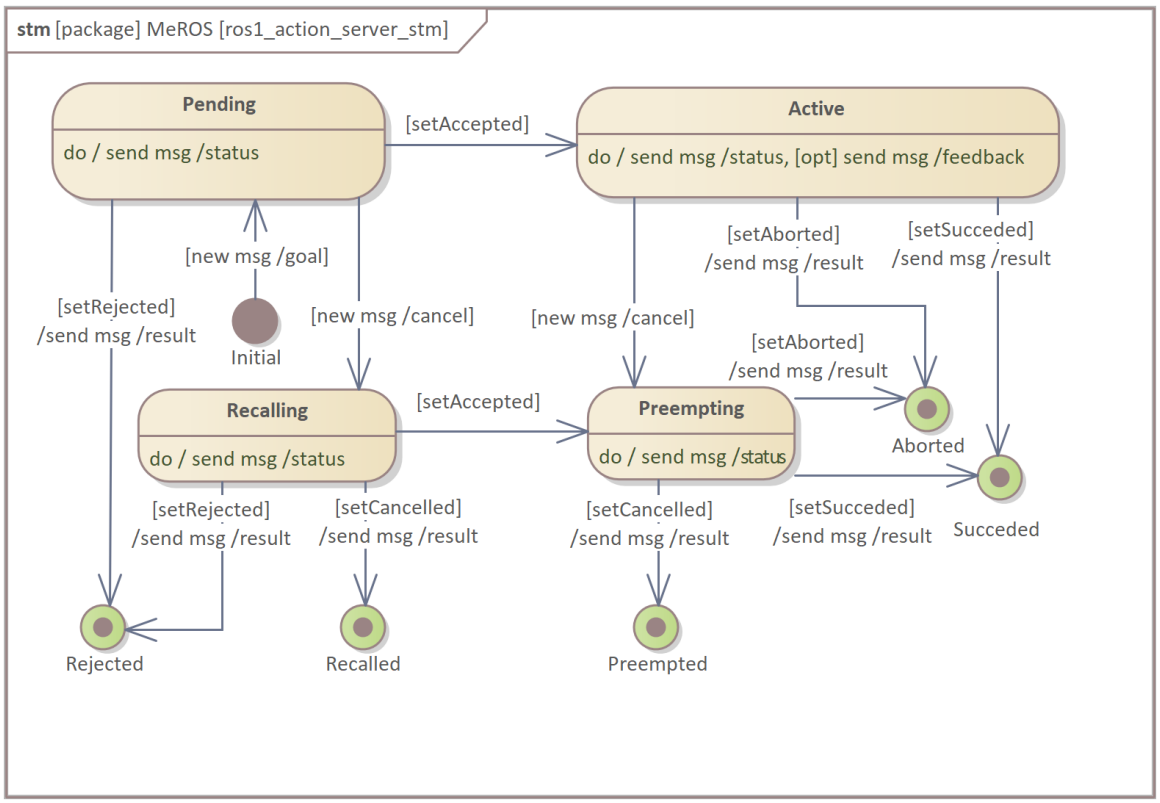

Figure 34: ROS 1 Action server – stm.

[2] http://wiki.ros.org/actionlib/DetailedDescription

The ROS 1 Action client state machine (Fig. 35) depends on the server state provided by the Action server in /status Topic and internal predicates.

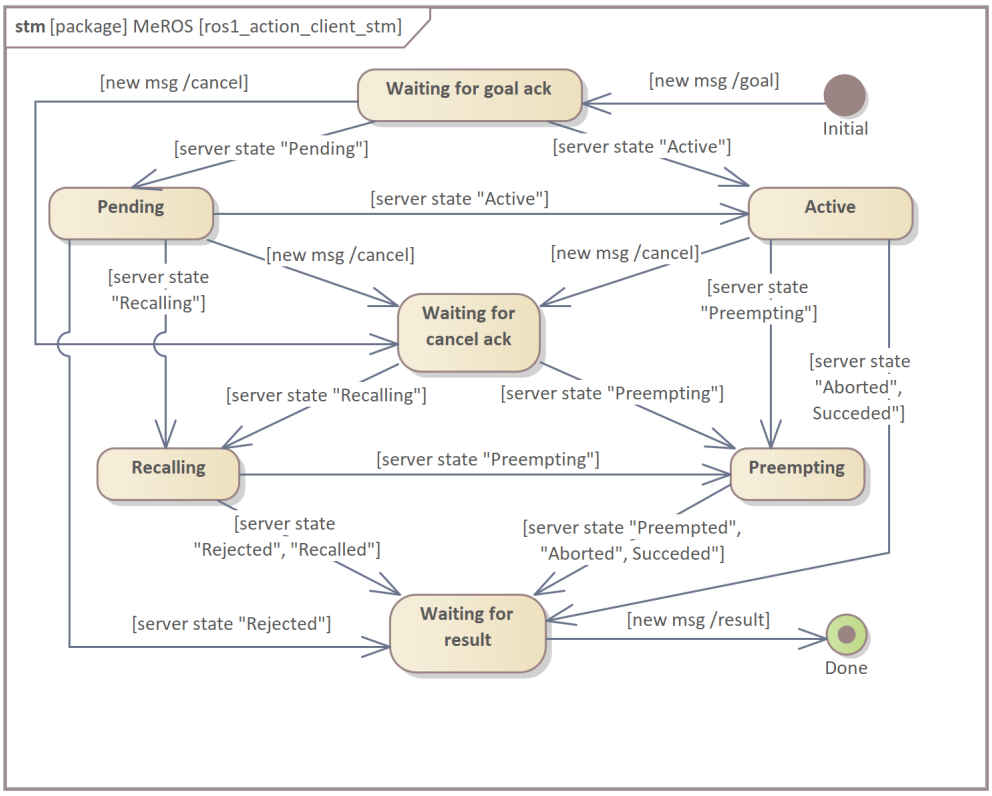

Figure 35: ROS 1 Action client – stm.

4 MeROS application

4.1 Application hints

MeROS metamodel can be employed in various ways in broad context of SE. Although, it is difficult to speak of an indication of the best procedure for its application, it is possible to formulate some practical guidelines for building a particular system model based on MeROS.

- Before defining a SysML Object, one must define the Block of which it is an instance. It is best to place Block definitions on the bdd diagrams as well. Afterwards, the definition of Objects and the formulation of the other diagrams can follow.
- The Object is an instance of the Block, and the Object's classifier corresponds to the Block's name. The Object's name specifies the name of the Block instance. The stereotypes for Block and Object are the same.
- The same Blocks and the same Objects should not be duplicated. A Block or Object is defined once and used in different diagrams (in particular, the same Blocks in both the Running System and Workspace diagrams or Objects in the ibd and sd diagrams).
- In practice, as long as automatic validation of models formulated in MeROS is not planned, there is no need to formulate a complete model in a SysML project.

To help develop user projects, the MeROS UML profile and other materials are accessible from MeROS project page[3].

4.2 Exemplary system

This section presents key aspects of an exemplary system development process incorporating MeROS. The exemplary system was created within the AAL INCARE project to control the Rico assistive robot (modified TIAGo platform with controller based on ROS 1) to execute transportation attendance tasks (Fig. 36).

Figure 36: Transportation attendance by Rico robot https://vimeo.com/670252925

The purpose of the following description is not to document the entire system but to illustrate, by example, representative aspects of the MeROS application.

[3] http://github.com/twiniars/meros

The part of the application scenario is conceptually presented in Fig. 37.

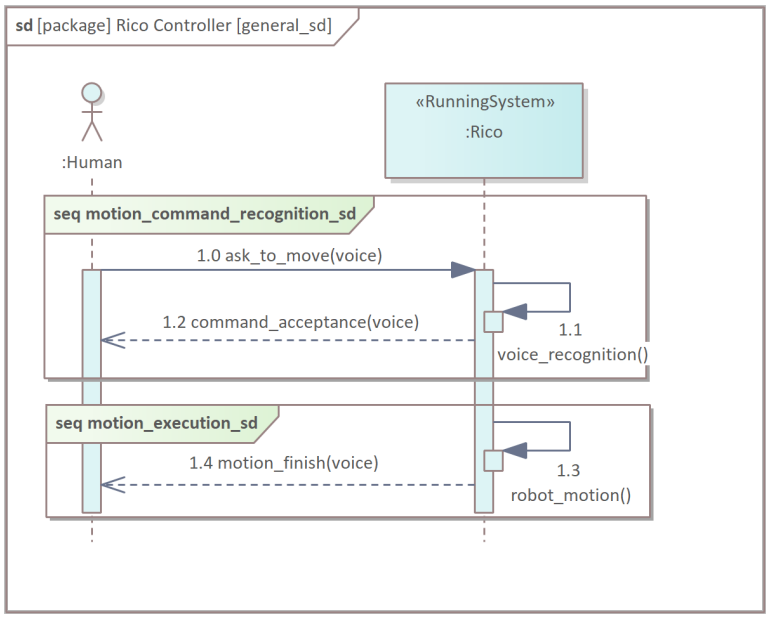

Figure 37: Concept scenario – sd.

Here, the system («RunningSystem» `:Rico`) and its behaviour are formulated in a general way. An actor (e.g. an elderly person) asks the robot to move. Then, the system recognises the voice command and vocally confirms the command's acceptance. Finally, the robot executes the motion and vocally informs that the motion is finished. In the following part of the description, the «RunningSystem» `:Rico` and sequence diagram frame `motion execution` from Fig. 37 are presented in a explicit way. The block definition diagram in Fig. 38 depicts the composition of «RunningSystem» `:Rico`.

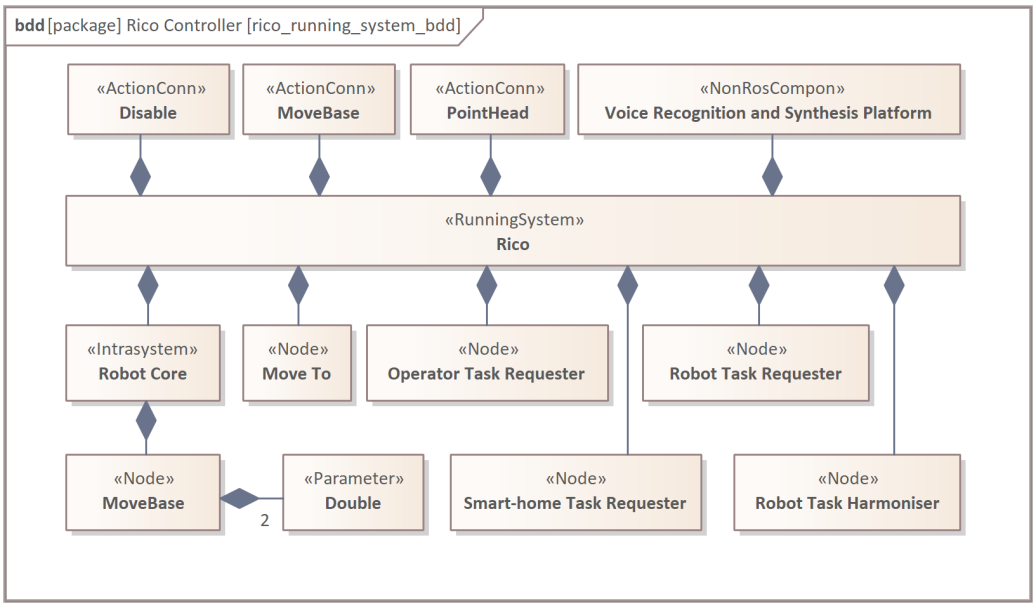

Figure 38: Rico «RunningSystem» composition – bdd.

The «RunningSystem» :Rico structure is depicted in Fig. 39. Here, and in the following diagrams, the rosout and ROS master «Node»s were omitted to make the diagrams more compact. The specific label is needed for «CommChannel», e.g., «CommChannel» :Move To to Robot Core, because this «CommChannel» is described later on.

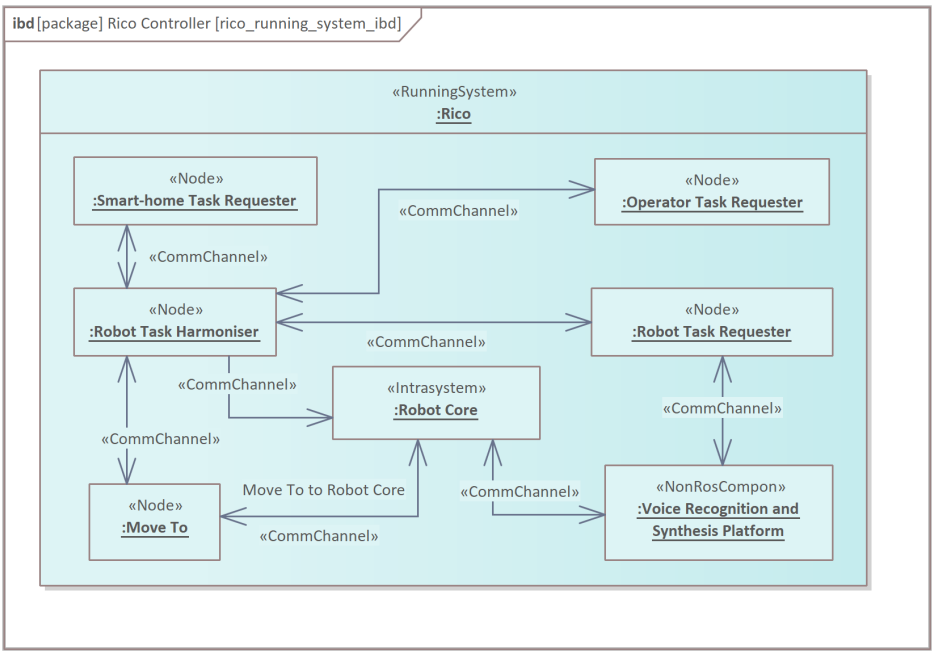

Figure 39: Structure of «RunningSystem» :Rico – ibd.

The system is based on TaskER framework [9] developed from the RAPP approach to construct systems with variable structure [36]. The role of the TaskER is to schedule a robot's tasks. It consists of (i) Task Requesters «Node»s to submit new tasks, (ii) Task Harmoniser «Node» to schedule tasks execution, (iii) dynamic «Node»s (here, «Node» :Move To) to execute a particular task on the robot hardware and (iv) cloud part, here «NonRosCompon» :Voice Recognition and Synthesis Platform. The common part of the controller is located in «Intrasystem» :Rico.

Fig. 40 illustrates how various instances of the same block are depicted in the model. Two «Parameter» Objects of the same classifier :Double are composed into «Node» :MoveBase.

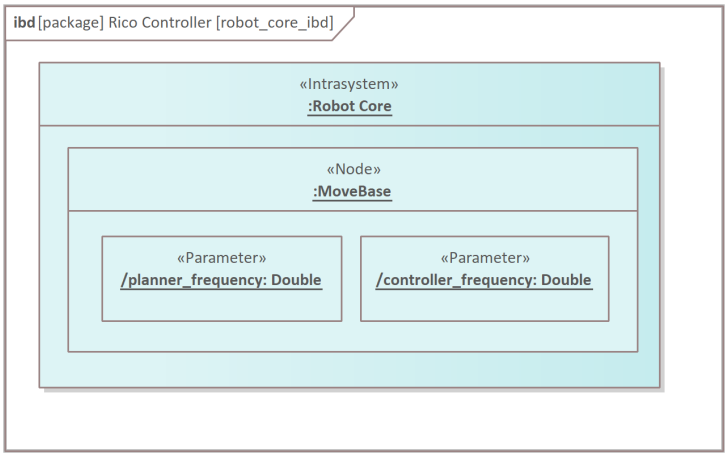

Figure 40: Selected elements of «Intrasystem» :Robot Core – ibd.

«CommChannel» `:Move To to Robot Core` is depicted in Fig. 41. It comprises three actions.

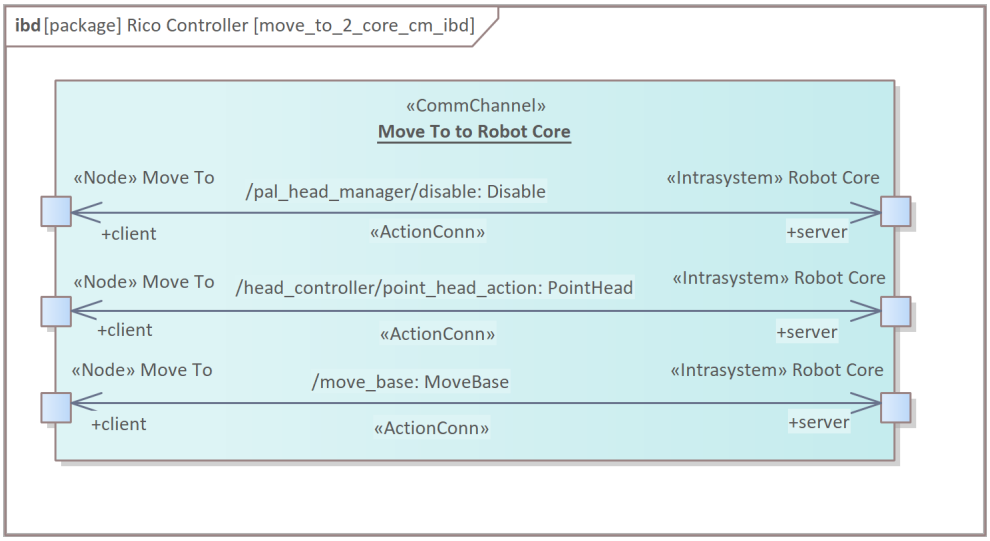

Figure 41: Example of «CommChannel» – ibd.

The part of the scenario generally described in Fig. 37 is depicted in detail in Fig. 42. The presentation remains conceptual from the behavioural point of view, but it considers the particular parts of the «RunningSystem» `:Rico`.

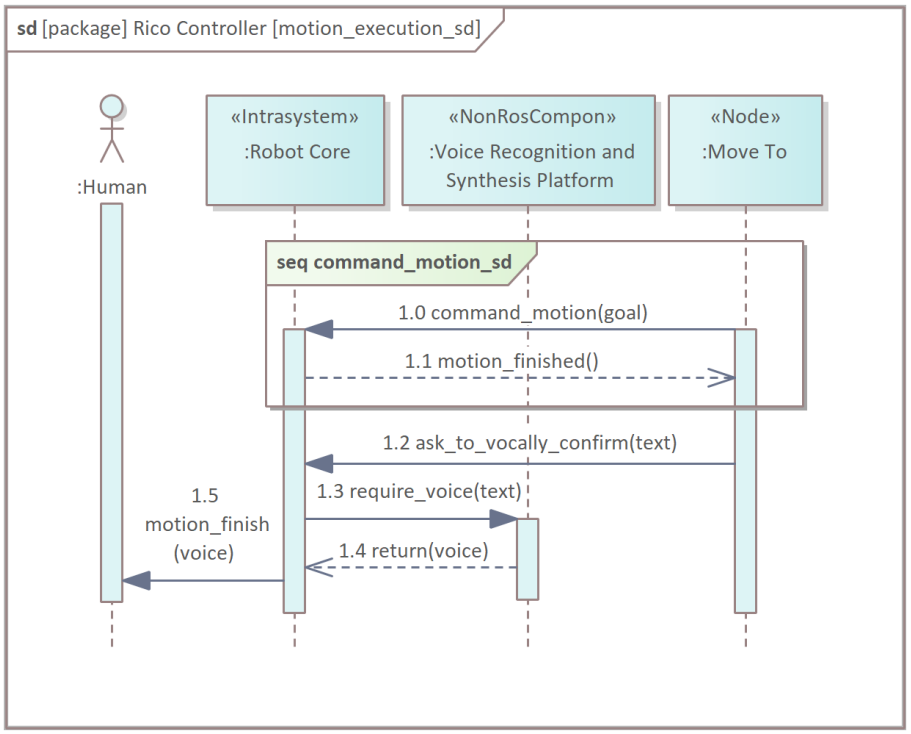

Figure 42: Motion execution operation – sd.

Finally, the particular communication methods are specified on the most detailed, ROS-specific level (Fig. 43). The command_motion operation includes the sequence of four steps of communication. Three Actions realise the communication, one utilised twice. The diagram comprises extra notes that make it easier to interpret.

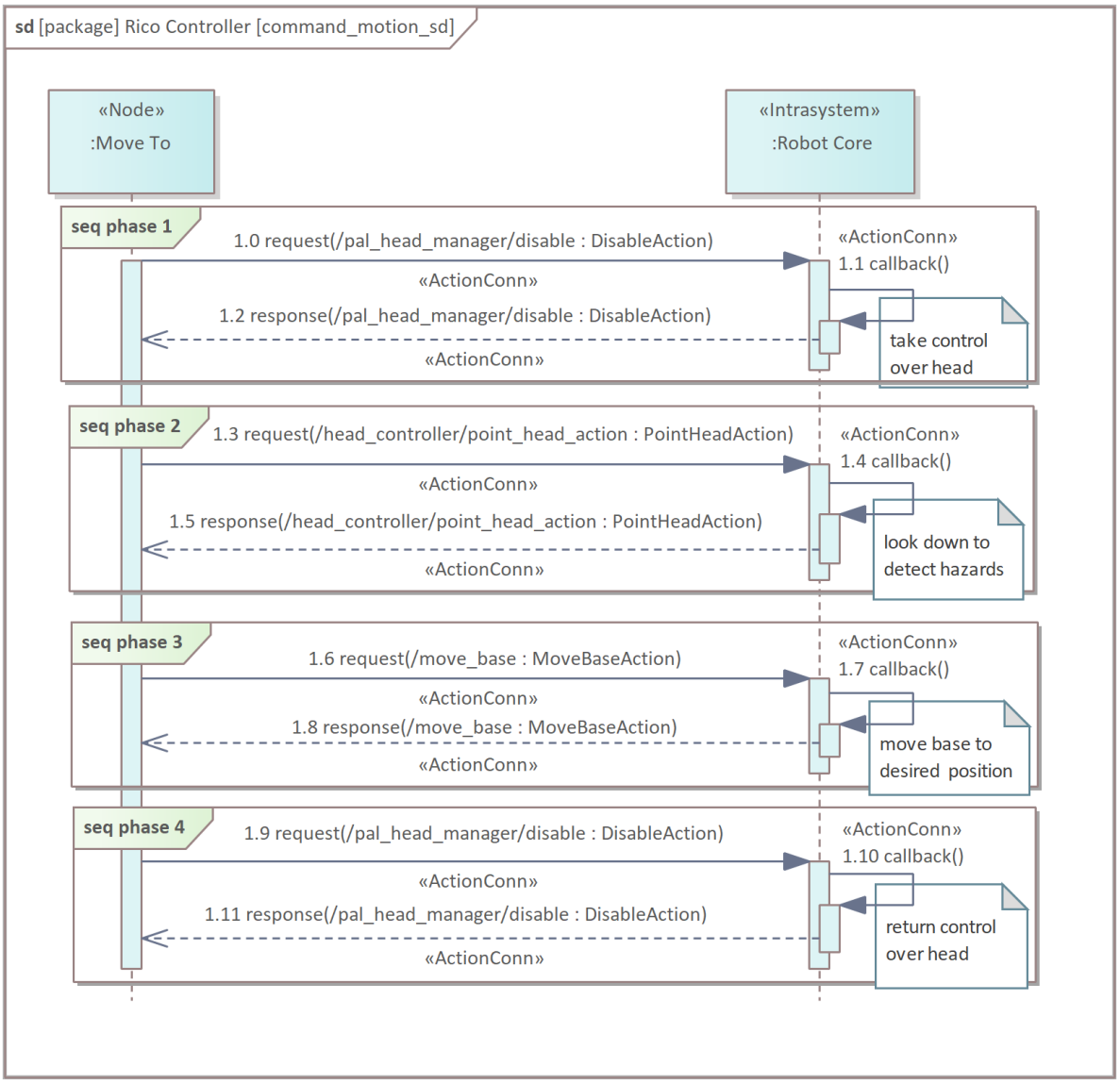

Figure 43: Command motion operation with detailed Communication methods presentation – sd.

The part of the «Workspace» `:Rico` that includes previously mentioned elements is presented in Fig. 44 and Fig. 45.

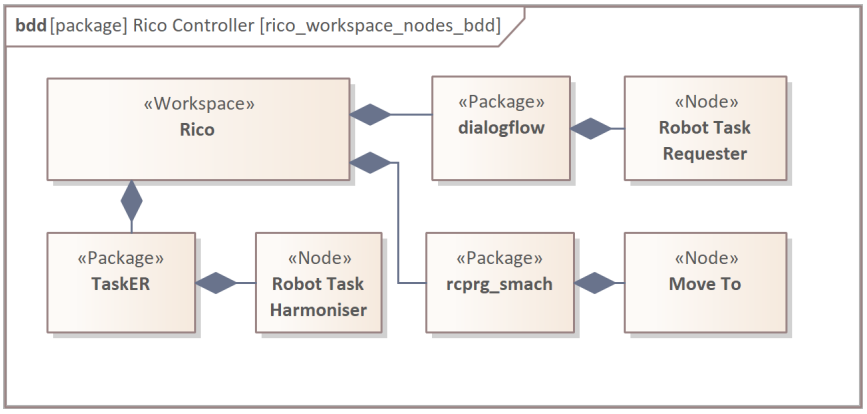

Figure 44: Rico «Workspace» composition – Packages with Nodes – bdd.

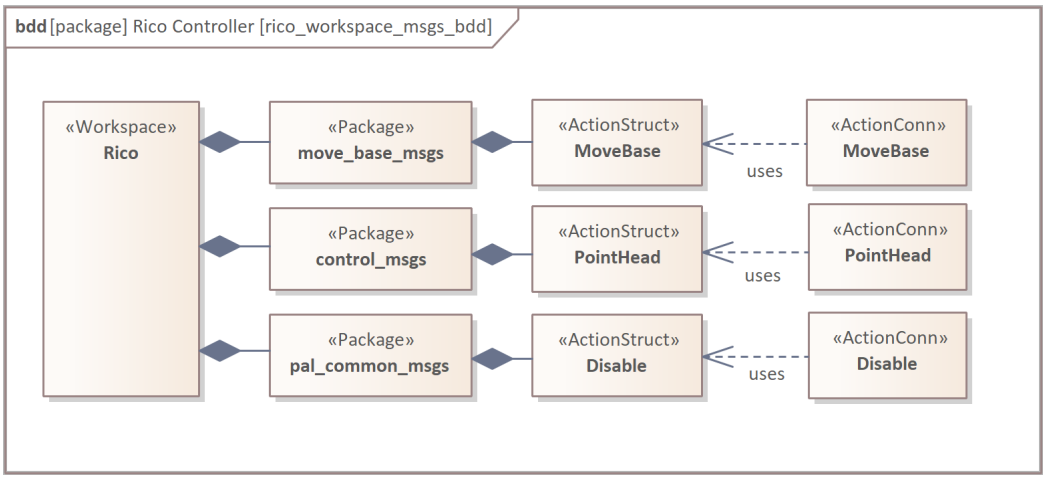

Figure 45: Rico «Workspace» composition – Packages with Msgs – bdd.

References

[1] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto. “A survey of Model Driven Engineering in robotics”. In: *Journal of Computer Languages* (2021), page 101021.

[2] J. Bézivin. “In search of a basic principle for model driven engineering”. In: *Novatica Journal, Special Issue* 5.2 (2004), pages 21–24.

[3] H. Bruyninckx. “Open robot control software: The OROCOS project”. In: *International Conference on Robotics and Automation (ICRA)*. Volume 3. IEEE. 2001, pages 2523–2528.

[4] H. Bruyninckx. “OROCOS: design and implementation of a robot control software framework”. In: *Proceedings of IEEE International Conference on Robotics and Automation*. Citeseer. 2002.

[5] K. Buys, S. Bellens, N. Vanthienen, W. Decre, M. Klotzbücher, T. De Laet, R. Smits, H. Bruyninckx, and J. De Schutter. “Haptic coupling with the PR2 as a demo of the OROCOS-ROS-Blender integration”. In: *IROS PR2 Workshop. San Francisco, California*. Volume 25. 2011, page 30.

[6] G. Canfora and M. Di Penta. “New frontiers of reverse engineering”. In: *Future of Software Engineering (FOSE'07)*. IEEE. 2007, pages 326–341.

[7] M. R. Chaudron, W. Heijstek, and A. Nugroho. “How effective is UML? An empirical perspective on costs and benefits”. In: *Software & Systems Modeling* 11 (2012), pages 571–580.

[8] M. Cholewiński, M. Janiak, and Ł. Juszkiewicz. “Software platform for practical verification of control algorithms developed for rescue and exploration mobile platform”. In: *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. 2015, pages 388–393.

[9] W. Dudek and T. Winiarski. “Scheduling of a Robot's Tasks With the TaskER Framework”. In: *IEEE Access* 8 (2020), pages 161449–161471. DOI: 10.1109/ACCESS.2020.3020265.

[10] M. Figat and C. Zieliński. “Parameterised robotic system meta-model expressed by Hierarchical Petri nets”. In: *Robotics and Autonomous Systems* (2022), page 103987. ISSN: 0921-8890. DOI: 10.1016/j.robot.2021.103987.

[11] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: The systems modeling language. 3rd ed*. Elsevier, Morgan Kaufmann, 2015.

[12] B. Habib and R. Romli. “A systematic mapping study on issues and importance of documentation in agile”. In: *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2021, pages 198–202.

[13] A. Hentout, A. Maoudj, and B. Bouzouia. “A survey of development frameworks for robotics”. In: *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*. IEEE. 2016, pages 67–72.

[14] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Diaz. “Robotics software frameworks for multi-agent robotic systems development”. In: *Robotics and Autonomous Systems* 60.6 (2012), pages 803–821.

[15] J. Karwowski, W. Dudek, M. Węgierek, and T. Winiarski. “HuBeRo-a Framework to Simulate Human Behaviour in Robot Research”. In: *Journal of Automation, Mobile Robotics and Intelligent Systems* 15.1 (2021), pages 31–38. DOI: 10.14313/JAMRIS/1-2021/4.

[16] S. Kent. “Model driven engineering”. In: *International conference on integrated formal methods*. Springer. 2002, pages 286–298.

[17] T. Kornuta, C. Zieliński, and T. Winiarski. “A universal architectural pattern and specification method for robot control system design”. In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* 68.No. 1 February (2020), pages 3–29. DOI: 10.24425/bpasts.2020.131827.

[18] W. F. Lages, D. Ioris, and D. C. Santini. “An architecture for controlling the barrett wam robot using ROS and Orocos”. In: *ISR/Robotik 2014; 41st International Symposium on Robotics*. VDE. 2014, pages 1–8.

[19] Y. Maruyama, S. Kato, and T. Azumi. “Exploring the performance of ROS2”. In: *Proceedings of the 13th International Conference on Embedded Software*. 2016, pages 1–10.

[20] *MeROS project – web page*. https://github.com/twiniars/MeROS.

[21] E. Mnkandla. “About software engineering frameworks and methodologies”. In: *AFRICON 2009*. IEEE. 2009, pages 1–5.

[22] *OMG Systems Modeling Language - Version 1.6*. Available online: https://www.omg.org/spec/SysML/1.6/PDF (accessed on 20 February 2020). Open Management Group. Dec. 2019. URL: https://www.omg.org/spec/SysML/1.6/.

[23] J. Pages, L. Marchionni, and F. Ferro. “Tiago: the modular robot that adapts to different research needs”. In: *International workshop on robot modularity, IROS*. Volume 290. 2016.

[24] P. Pałka, C. Zieliński, W. Dudek, D. Seredyński, and W. Szynkiewicz. “Communication-Focused Top-Down Design of Robotic Systems Based on Binary Decomposition”. In: *Energies* 15.21 (2022), page 7983.

[25] J. Park, R. Delgado, and B. W. Choi. “Real–time characteristics of ROS 2.0 in multiagent robot systems: an empirical study”. In: *IEEE Access* 8 (2020), pages 154637–154651.

[26] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Volume 3. 3.2. 2009.

[27] D. C. Schmidt. “Model-driven engineering”. In: *Computer-IEEE Computer Society*- 39.2 (2006), page 25.

[28] D. Seredyński, T. Winiarski, and C. Zieliński. “FABRIC: Framework for Agent-Based Robot Control Systems”. In: *12th International Workshop on Robot Motion and Control (RoMoCo)*. Edited by K. Kozłowski. IEEE. 2019, pages 215–222. DOI: 10.1109/RoMoCo.2019.8787370.

[29] O. Shehory and A. Sturm. *Agent-oriented software engineering: reflections on architectures, methodologies, languages, and frameworks*. Springer, 2014.

[30] E. Tsardoulias and P. Mitkas. “Robotic frameworks, architectures and middleware comparison”. In: *arXiv preprint arXiv:1711.06842* (2017).

[31] M. Wenger, W. Eisenmenger, G. Neugschwandtner, B. Schneider, and A. Zoitl. “A model based engineering tool for ROS component compositioning, configuration and generation of deployment information”. In: *2016 IEEE 21st international conference on emerging technologies and factory automation (ETFA)*. IEEE. 2016, pages 1–8.

[32] T. Winiarski. “MeROS: SysML-based Metamodel for ROS-based Systems”. In: *IEEE Access* 11 (2023), pages 82802–82815. DOI: 10.1109/ACCESS.2023.3301727.

[33] T. Winiarski, S. Jarocki, and D. Seredyński. “Grasped Object Weight Compensation in Reference to Impedance Controlled Robots”. In: *Energies* 14.20 (2021). ISSN: 1996-1073. DOI: 10.3390/en14206693.

[34] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliński. “EARL – Embodied Agent-Based Robot Control Systems Modelling Language”. In: *Electronics* 9.2 - 379 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9020379.

[35] C. Zieliński and T. Winiarski. “Motion Generation in the MRROC++ Robot Programming Framework”. In: *International Journal of Robotics Research* 29.4 (2010), pages 386–413. DOI: 10.1177/0278364909348761.

[36] C. Zieliński et al. “Variable structure robot control systems: The RAPP approach”. In: *Robotics and Autonomous Systems* 94 (2017), pages 226–244. ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.05.002.