

METHODS

MeROS: SysML-Based Metamodel for ROS-Based Systems

TOMASZ WINIARSKI , (Member, IEEE)

Warsaw University of Technology, Institute of Control and Computation Engineering, 00-665 Warsaw, Poland

e-mail: tomasz.winiarski@pw.edu.pl

This work was supported by the Centre for Priority Research Area Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) Programme.

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. Although the use of frameworks is widespread nowadays, for the most popular of them, Robot Operating System (ROS), a contemporary metamodel has been missing so far. This article proposes a new metamodel for ROS called MeROS, 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 example of Rico assistive robot. 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.

INDEX TERMS Systems engineering, robot operating system ROS, ROS 1, ROS 2, robotic framework, SysML, platform specific model PSM.


I. 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 [1], [2], [3]. 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 [4], [5]. Diverse robotics frameworks have been developed so far [6], [7], [8]. 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) [9] is unsuitable for hard RT (Real Time) systems, so one of the solutions in practical applications

(e.g., [10], [11], [12], [13], [14], [15]), is to integrate ROS 1 with Orocos [16], [17]. 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 [18], [19].

ROS 1 has evolved considerably from the initial distributions. Its metamodels created so far are now incomplete and outdated (sec. V). 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 ROS 1 and ROS 2, which this article undertakes by presenting the new metamodel for ROS – MeROS.

The robotic models can be subdivided [20] into Platform Independent Models (PIM), e.g., [21], [22], [23], [24], and

The associate editor coordinating the review of this manuscript and approving it for publication was Yingxiang Liu .

¹<https://discourse.ros.org/t/2022-ros-metrics-report/29594>

Platform Specific Models (PSM). The metamodels of ROS, including MeROS, belong to PSM and should answer to the component nature of ROS [25], [26].

MeROS is founded on SysML (Systems Modeling Language) [27], [28], a profile of UML (Unified Modeling Language). Modelling in languages from the UML family addresses a number of important aspects of systems engineering [29]. 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 [30] for existing systems. Agile-type strategies involve modifying the documentation as the project develops [31].

The following presentation starts with formulating the requirements (sec. II) for the MeROS metamodel. These requirements are allocated to the metamodel that is described in sec. III. The way to present a model of a specific application based on MeROS is presented on a practical example in sec. IV. A comparison of the MeROS with similar metamodels is presented in sec. V. The paper is finalised with conclusions (sec. VI).

II. 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) [24] PIM development and its applications (e.g. [23], [32], [33]). Verification of draft versions of MeROS by its practical applications led to an iterative reformulation of requirements and MeROS itself. The article presents the final version of both MeROS metamodel and the requirements it originates from.

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

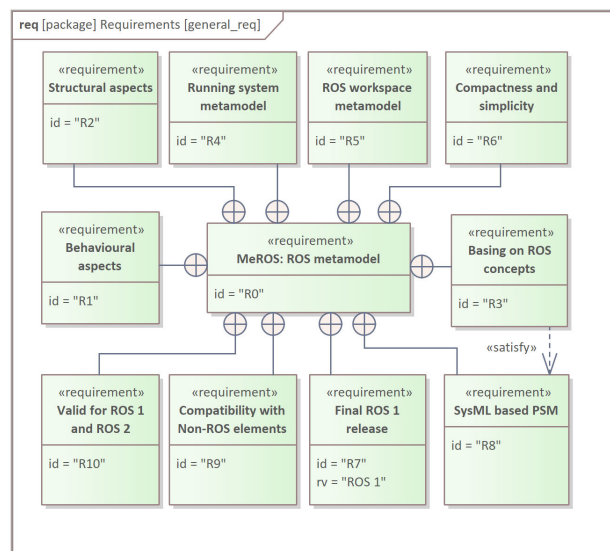


FIGURE 1. General requirements.

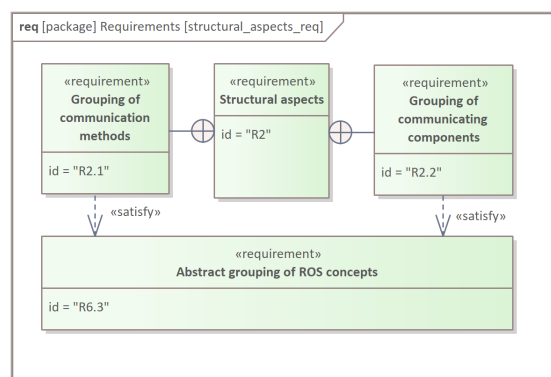


FIGURE 2. Structural aspects requirements.

for. The lack of a tag means the element is general for both ROS versions.

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.

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.

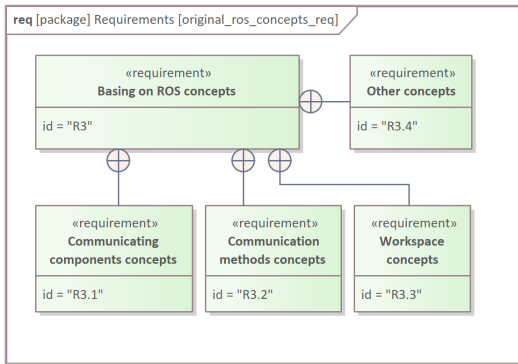


FIGURE 3. ROS concepts requirements.

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].

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] and (ii) Metapackage [R3.3.2] introduced in the latest releases of ROS 1. 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. 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

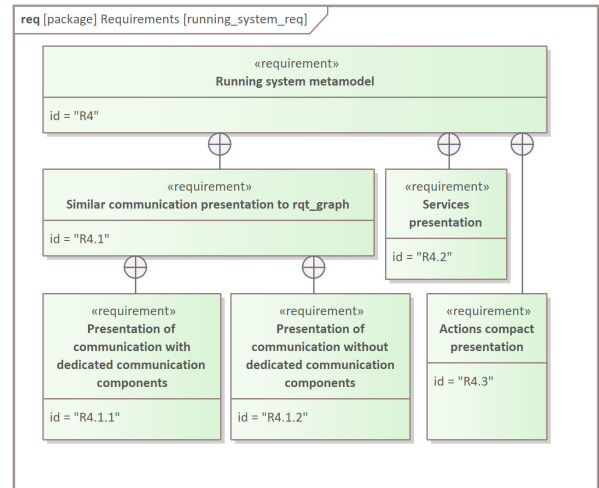


FIGURE 4. Running system requirements.

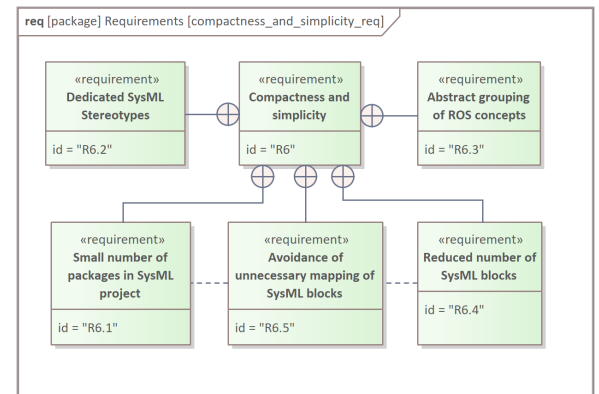


FIGURE 5. Compactness and simplicity requirements.

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.

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.

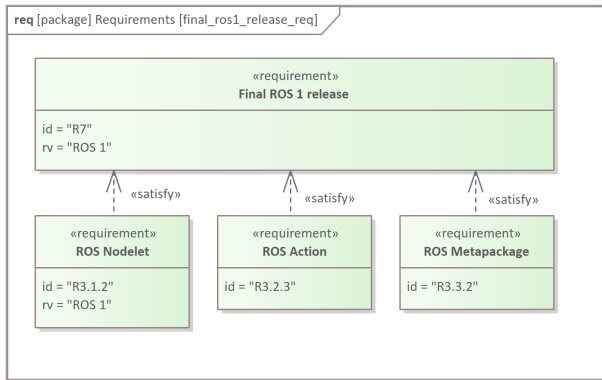


FIGURE 6. Final ROS 1 release requirements.

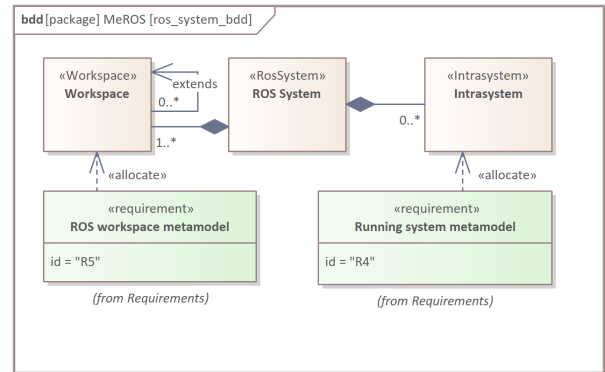


FIGURE 8. ROS system general composition – bdd.

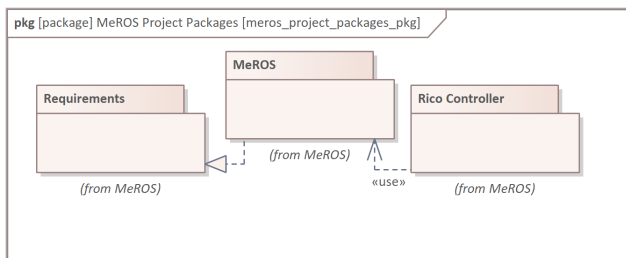


FIGURE 7. MeROS project SysML packages, where rico controller is an exemplary realisation of MeROS metamodel.

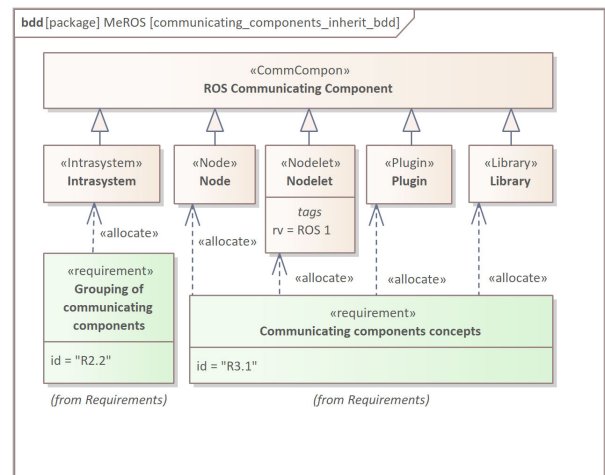


FIGURE 9. Communicating component and specialised blocks – bdd.

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.

III. MeROS METAMODEL

MeROS metamodel is formulated according to the requirements discussed in the previous section. Sec. III-A presents MeROS blocks’ structural composition, and sec. III-B 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. IV-B. The stereotypes are introduced in MeROS metamodel with the dedicated MeROS profile [R6.2].

A. 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, Workspaces and Intrasytems are composed into ROS System (Fig. 8).

Consequently, some concepts (e.g., Node) occur in Workspaces and Intrasytems. 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.

It should be noted that behavioural aspects of a particular model specified in MeROS can be formulated by operation

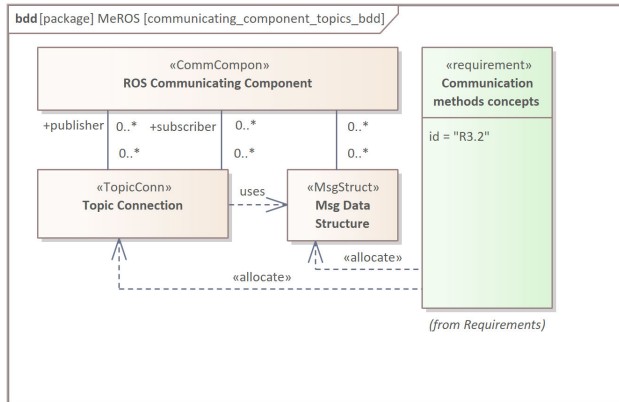


FIGURE 10. Communicating component relations – topics – bdd.

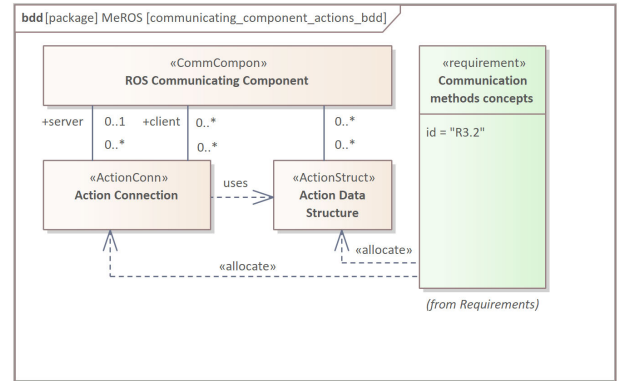


FIGURE 12. Communicating component relations – actions – bdd.

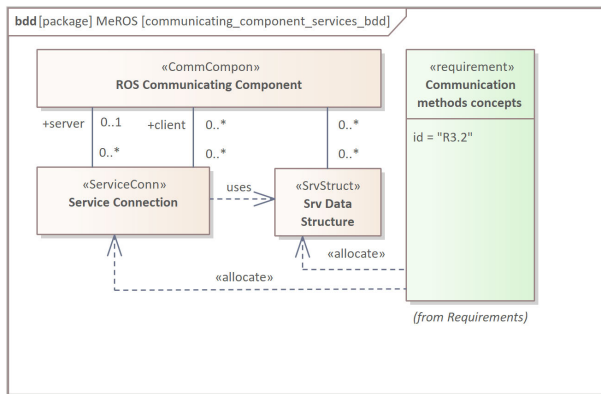


FIGURE 11. Communicating component relations – services – bdd.

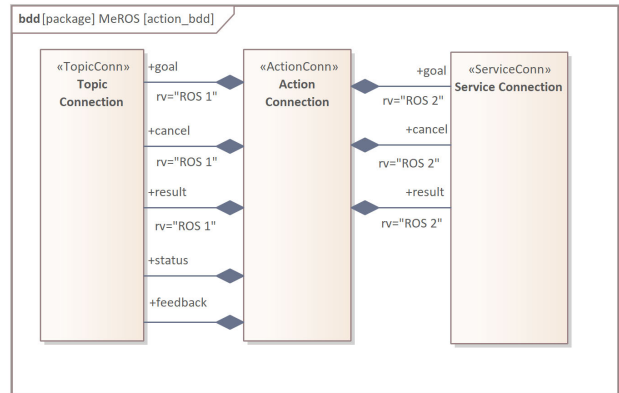


FIGURE 13. Action – bdd.

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.

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

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.

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.

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.

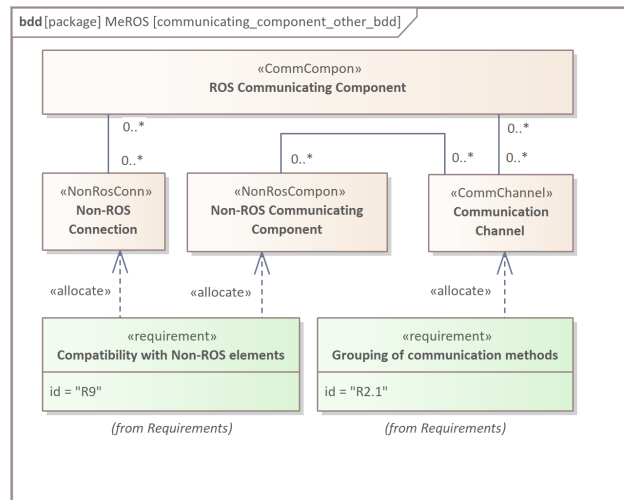


FIGURE 14. Communicating component relations – aggregates, Non-ROS elements – bdd.

The Communication Channel [34] 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 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.

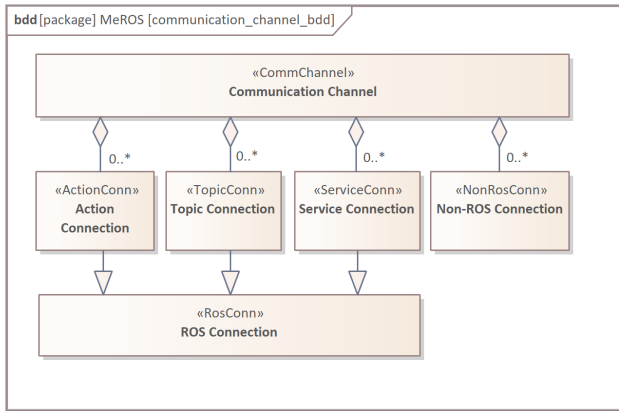


FIGURE 15. Communication channel – bdd.

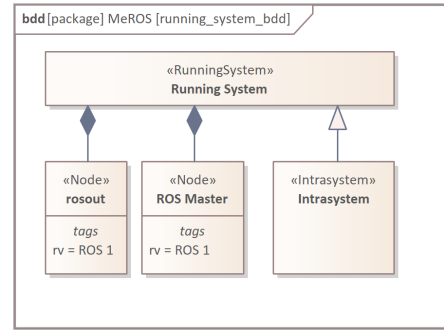


FIGURE 18. Running system compositions – bdd.

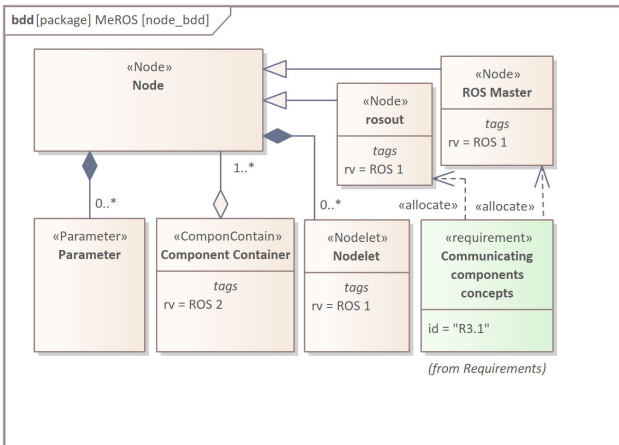


FIGURE 16. Node – bdd.

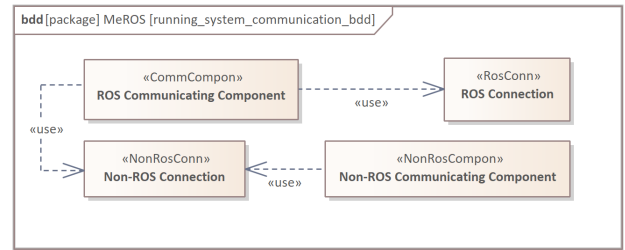


FIGURE 19. Running system communication – bdd.

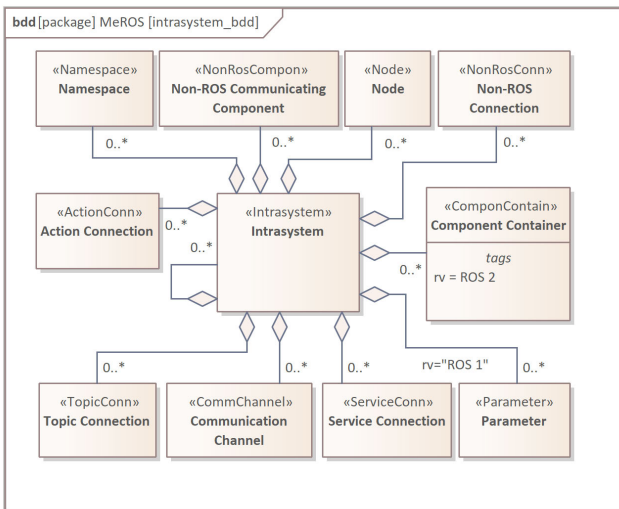


FIGURE 17. Intrasytem compositions – bdd.

The Intrasytem (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 Intrasytem composes the other Intrasytems.

The Running System (Fig. 18) is a specialisation of the Intrasytem 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 Intrasytems corresponds to PIM specification. Then, the detailing of these aggregates corresponds to the transition from PIM to PSM.

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.

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

The Workspace (Fig. 21) contains of Packages that compose 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.

B. 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

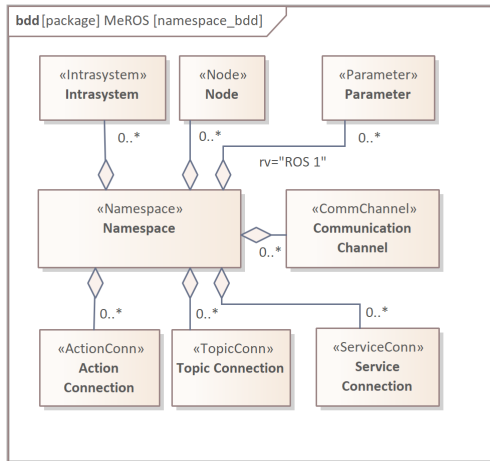


FIGURE 20. Namespace composition – bdd.

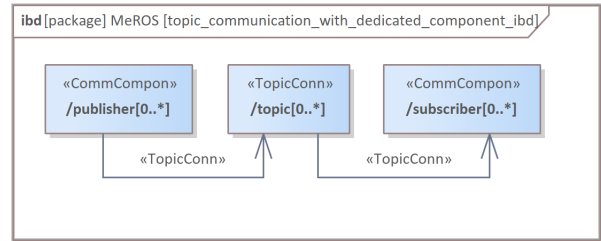


FIGURE 22. Topics with dedicated communication components – all components – ibd.

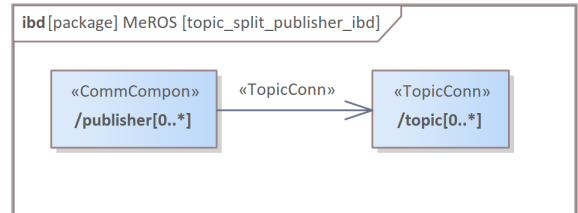


FIGURE 23. Topics with dedicated communication components – publisher – ibd.

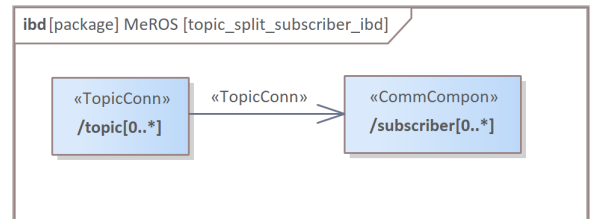


FIGURE 24. Topics with dedicated communication components – subscriber – ibd.

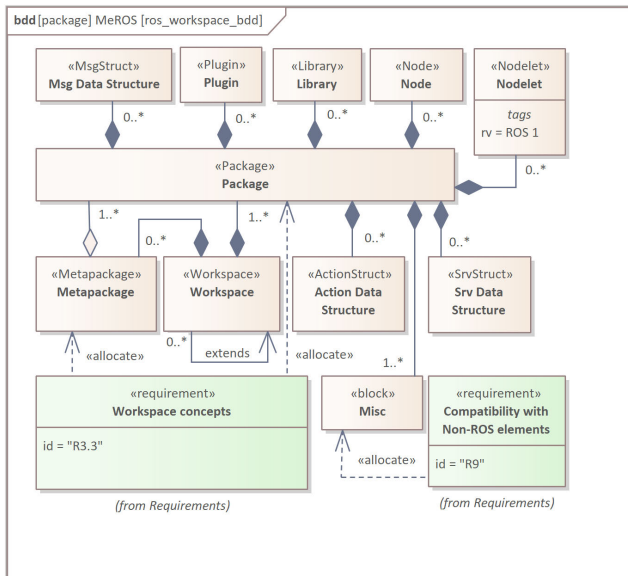


FIGURE 21. ROS workspace composition – bdd.

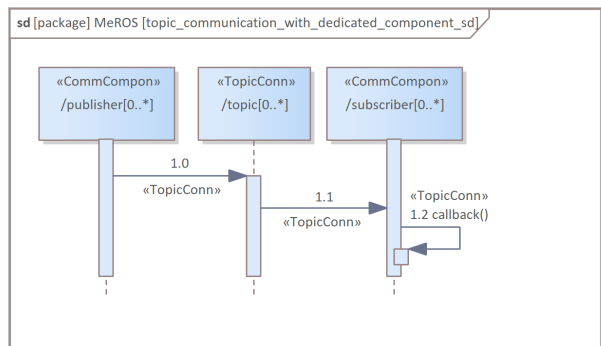


FIGURE 25. Topics with dedicated communication components – sd.

internal block diagrams (ibd) and behavioural diagrams are discussed. The goal is to present three modes of communication: Topic [R4.1] (sec. III-B1), Service [R4.2] (sec. III-B2) and Action [R4.3] (sec. III-B3). 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.

1) TOPIC

Fig. 22 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.

Thanks to a dedicated component to represent communication, the diagram in Fig. 22 can be split into two considering publisher (Fig. 23) and subscriber (Fig. 24) separately, without losing information. It is especially useful when system

fragments are presented after its decomposition that subdivides communication channels.

Fig. 25 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. 26 and Fig. 27 present an alternative approach to depict the system communicating via topics. In this case, no dedicated communication components are used [R4.1.2].

2) SERVICE

For each ROS Service, there is at most one server and a number of clients (Fig. 28 and Fig. 29). Service-type communication is bidirectional and realises RPC (remote procedure call).

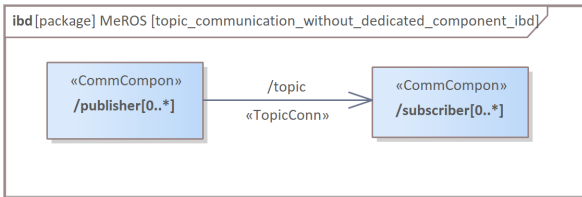


FIGURE 26. Topics without dedicated communication components – ibd.

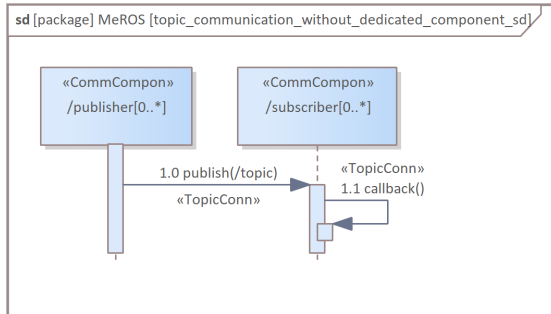


FIGURE 27. Topics without dedicated communication components – sd.

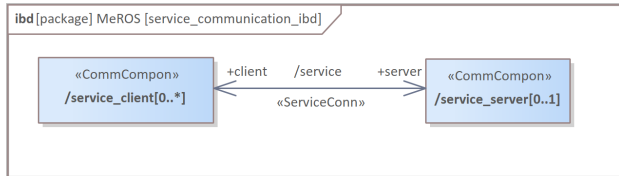


FIGURE 28. Service-based communication – ibd.

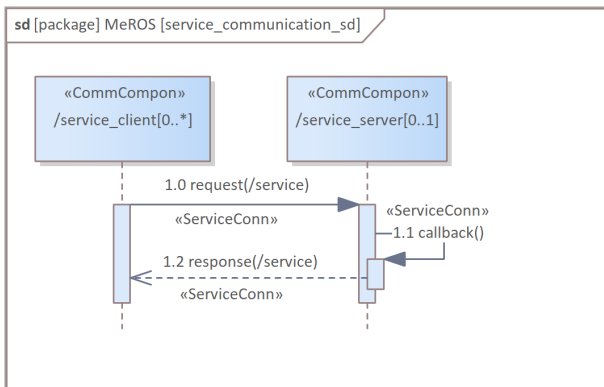


FIGURE 29. Service-based communication – sd.

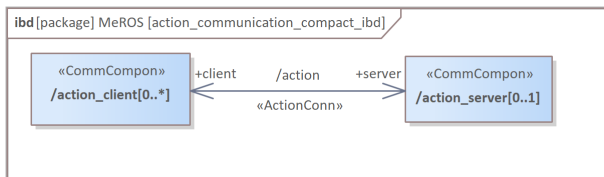


FIGURE 30. Action-based communication – compact representation – ibd.

3) ACTION

ROS Action communication’s general, simplified structure (Fig. 30) is analogous to ROS Service. These type of presentation is universal for ROS 1 and ROS 2.

An Action (Fig. 31) is based on several Topics in ROS 1, while on Topics and Services in ROS 2.

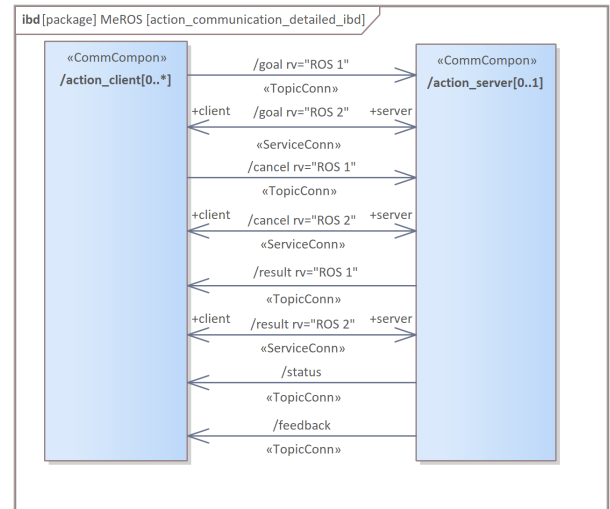


FIGURE 31. Action-based communication – detailed – ibd.

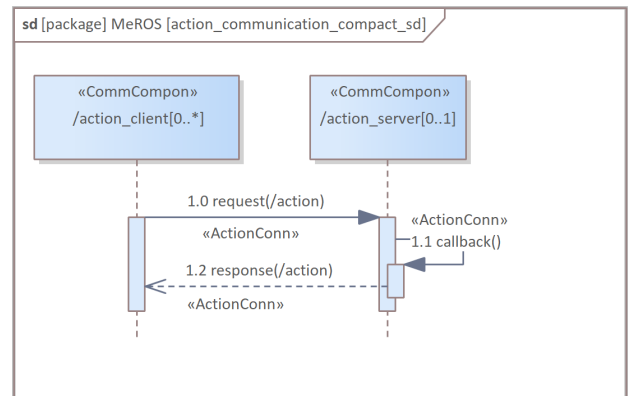


FIGURE 32. Action-based communication sequence – compact presentation – sd.

In practice, to present an action-related communication compactly on sd diagram (Fig. 32) 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.

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. 33 depicts the ROS 1 Action server state machine. Its transitions depend on the new messages sent by the Action client or internal predicates.

The ROS 1 Action client state machine (Fig. 34) depends on the server state provided by the Action server in /status Topic and internal predicates.

²<http://wiki.ros.org/actionlib/DetailedDescription>

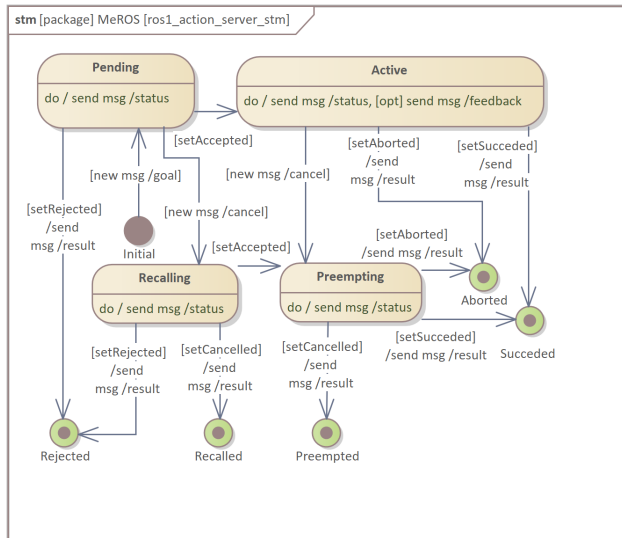


FIGURE 33. ROS 1 Action server – stm.



FIGURE 35. Transportation attendance by Rico robot <https://vimeo.com/670252925>.

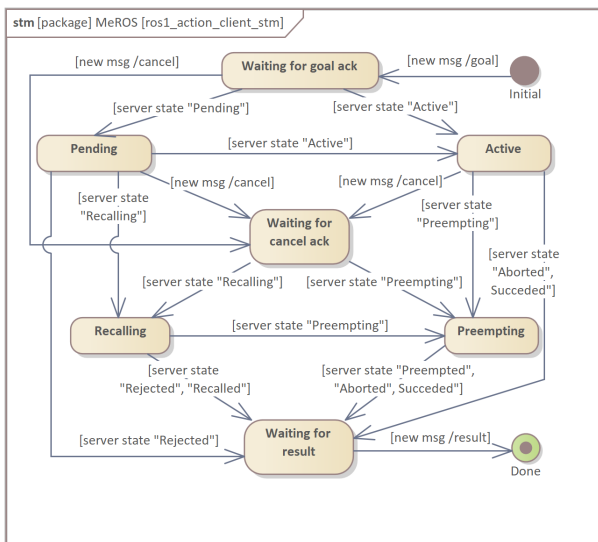


FIGURE 34. ROS 1 Action client – stm.

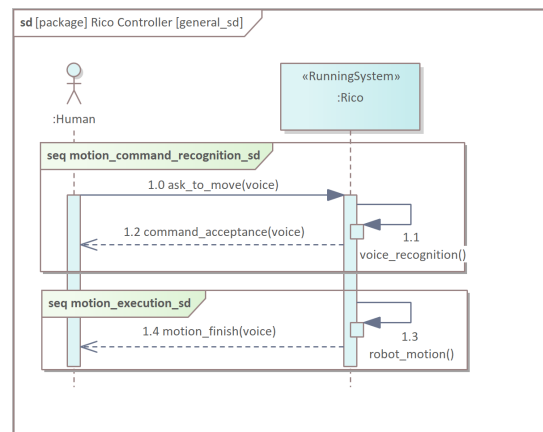


FIGURE 36. Concept scenario – sd.

- 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

IV. MeROS APPLICATION

A. 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.

B. 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. 35).

The purpose of the following description is not to document the entire system but to illustrate, by example, representative aspects of the MeROS application. The part of the application scenario is conceptually presented in Fig. 36.

Here, the system (≪RunningSystem≫ :Rico) and its behaviour are formulated in a general way. An actor

³<http://github.com/twiniars/meros>

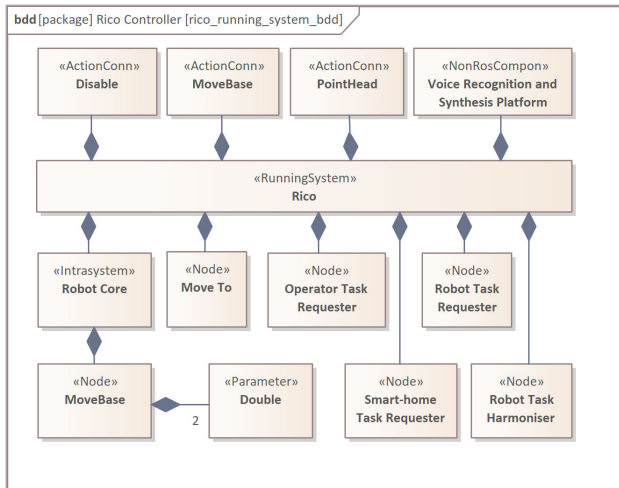


FIGURE 37. Rico «RunningSystem» composition – bdd.

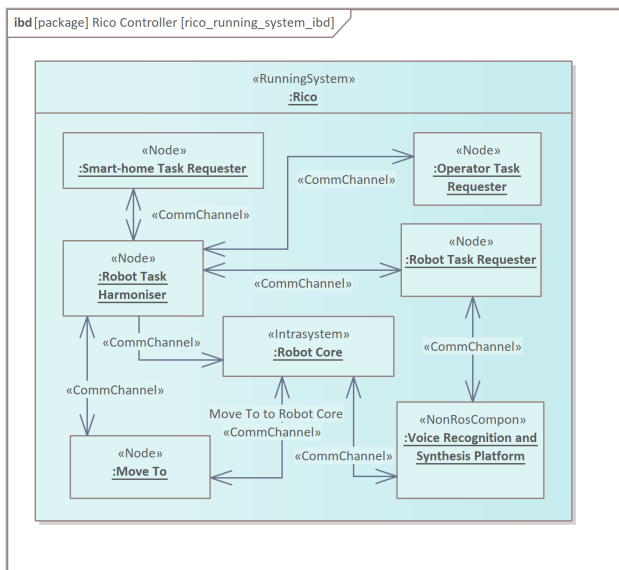


FIGURE 38. Structure of «RunningSystem» :Rico – ibd.

(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. 36 are presented in an explicit way. The block definition diagram in Fig. 37 depicts the composition of «RunningSystem» :Rico.

The «RunningSystem» :Rico structure is depicted in Fig. 38. 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.

The system is based on TaskER framework [23] developed from the RAPP approach to construct systems

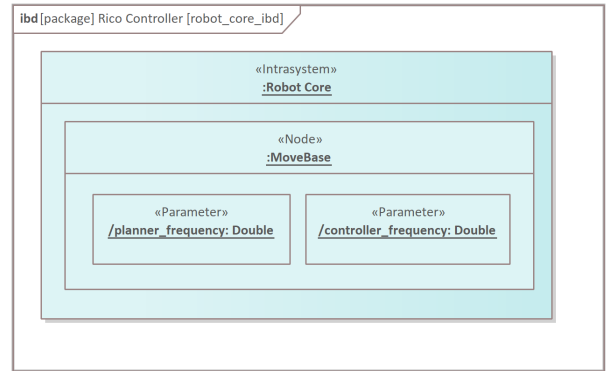


FIGURE 39. Selected elements of «Intrasytem» :Robot Core – ibd.

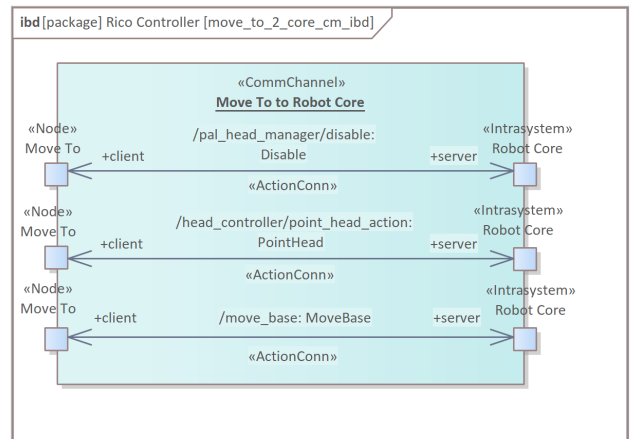


FIGURE 40. Example of «CommChannel» – ibd.

with variable structure [21]. 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 «Intrasytem» :Rico. Fig. 39 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.

«CommChannel» :Move To to Robot Core is depicted in Fig. 40. It comprises three actions.

The part of the scenario generally described in Fig. 36 is depicted in detail in Fig. 41. The presentation remains conceptual from the behavioural point of view, but it considers the particular parts of the «RunningSystem» :Rico.

Finally, the particular communication methods are specified on the most detailed, ROS-specific level (Fig. 42). 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.

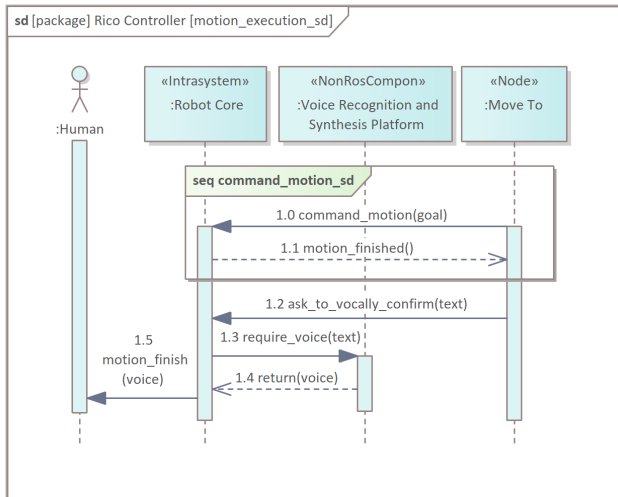


FIGURE 41. Motion execution operation – sd.

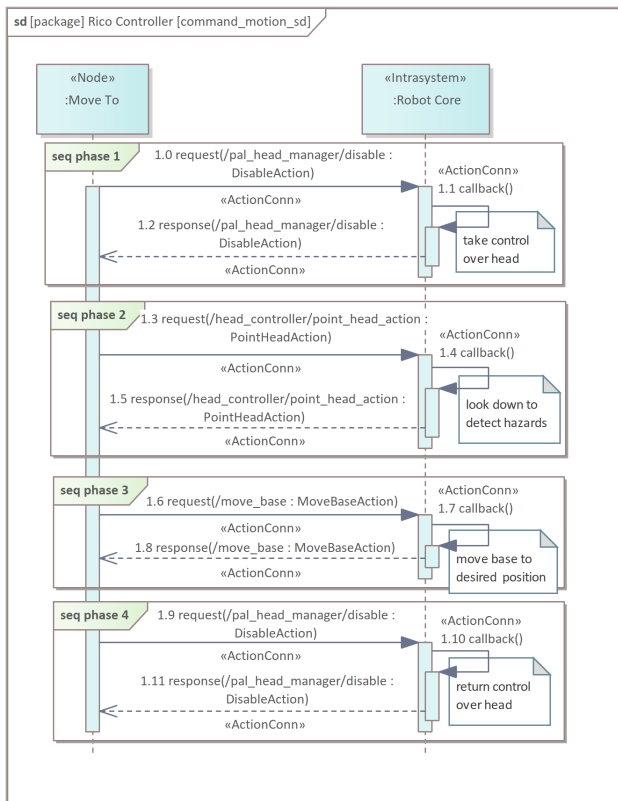


FIGURE 42. Command motion operation with detailed Communication methods presentation – sd.

The part of the «Workspace» :Rico that includes previously mentioned elements is presented in Fig. 43 and Fig. 44.

V. RELATED WORK

Papers in the scope of the literature review are chosen based on an intensive study of the previous scientific work in robotic systems modelling. In particular, the survey [20] is deeply analysed. As the qualification criterion for this review, the ROS metamodel was chosen that must be described using UML language or SysML language. Six papers describing

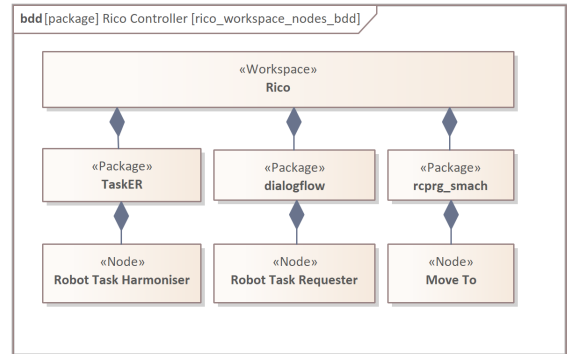


FIGURE 43. Rico «Workspace» composition – Packages with Nodes – bdd.

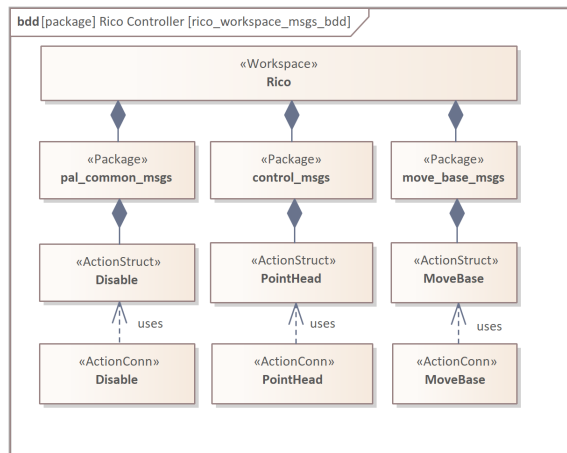


FIGURE 44. Rico «Workspace» composition – Packages with Msgs – bdd.

TABLE 1. MeROS requirements satisfaction in ROS specific metamodels.

Metamodel/req.	R3.1.1 Node	R3.1.2 Nodelet	R3.1.3 ROS plugin	R3.1.4 ROS library	R3.2.1 Topic	R3.2.2 Service	R3.2.3 Action	R3.3.1 Package	R3.3.2 Metapackage	R3.4.2 ROS Parameter	R6.3 Grouping of concepts
Ecore	+	-	-	-	+	+	+	-	-	+	+
RosSystem	-	-	-	-	+	+	+	-	-	-	-
HyperFlex	+	-	-	-	+	+	+	+	-	+	-
RoBMEX	+	-	-	-	+	+	-	+	+	-	+
ROSMOD	+	-	-	+	-	+	-	+	-	-	+
MeROS	+	+	+	+	+	+	+	+	+	+	+

five metamodels met this criterion, and all used UML to describe ROS 1. The metamodels are contrasted with the representative requirements to which MeROS is subjected (Tab. 1) and that clearly differentiate compared metamodels. For clarification, the table refers to aspects of the metamodels, which are visualised in the analysed papers' diagrams.

The authors of [26] present Ecore – the ROS 1 metamodel as the central part of the ReApp workbench created to support the efficiency of software creation for robotic systems. The metamodel is specified in a single, extensive structural diagram, with the ROS node being its central part. The diagram describes the aspects of the running system and comprises all ROS communication methods. The nodes are integrated into an AppNetwork concept [R6.3].

In the paper [35], the authors propose two methods based on the created RosSystem metamodel. It aims at the automated generation of models from manually written artefacts through static code analysis and monitoring the execution of the running system. A large part of the work is concentrated on the toolchain. This ROS metamodel is structurally specified in a UML class diagram, emphasising communication methods.

HyperFlex toolchain [36], [37] includes extensive and comprehensive metamodel addressing ROS 1 and complementary Orocos. Formerly, these two frameworks were used together to take from the RT properties of Orocos and the elasticity of ROS. The presentation of HyperFlex is complex [36], [37], and both the running system and workspace are considered. Concepts such as nodelet or metapackage are missing due to the HyperFlex period of its foundation.

RoBMEX [38] was created as a top-down methodology based on a set of domain-specific languages that enhance the autonomy of ROS-based systems by allowing the creation of missions graphically and then generating automatically executable source codes conforming to the designed missions. Hence, the ROS metamodel was extended by the upper layer with mission/task specification. The metamodel is complex and inspiring and consists of running and workspace parts. Regarding the workspace, the grouping concepts are introduced as subpackages, classified in Tab. 1 as metapackage for generality.

ROSMOD [39] is the Robot Operating System Model-driven development tool suite, an integrated development environment for rapid prototyping component-based software for ROS. Its internal metamodel is complex and comprises a number of standard ROS concepts and additional grouping concepts. Although the description is extensive, the ROSMOD was created in 2016, hence some current concepts are missing, like ROS actions or nodelets.

VI. CONCLUSION AND DISCUSSION

Diagrams are an integral part of the description of component-based robot control systems. ROS comprises `rqt_graph` tool that generates diagrams with the structure of the running system. This capability is readily used by software developers (e.g., [40], [41]) due to its ease of use. Unfortunately, this tool has many limitations despite its numerous advantages and configurability. Hence, in parallel to automatically generated diagrams, others are needed, some of which are based on UML/SysML. The most comprehensive modelling solutions include explicitly defined metamodels. As a novelty regarding previous works, this paper proposes

an up-to-date metamodel for finale release of ROS 1 and ROS 2 supported by profile to support the metamodel application in ROS applications models. In MeROS, the metamodel of original ROS concepts is extended by the abstract grouping concepts. It lets to present part of the system in a PIM-like style instead of a platform specific – PSM.

Although the adoption of UML/SysML-based domain metamodels has many positive implications, it also has its problems and limitations. Although the diagrams can be drawn in general-purpose graphics programs, this is not advisable, especially for complex systems. Modelling software such as Enterprise Architect or Visual Paradigm is highly recommended when creating UML/SysML projects. In particular, creating a set of diagrams outside a SysML project is more time-consuming, and it is easier to introduce errors. In practice, the cost of modelling software is not a major obstacle, and its popularity makes it easier to implement its employment. A problem with SysML development environments is that they are not standardised in many aspects and vary considerably in functionality. This problem makes it difficult to use advanced features such as automatic model analysis, e.g. to check for metamodel compatibility.

There are many methodologies for conducting and documenting projects, and not all are based on languages from the UML family. In some simplification, one could say that some project teams use UML extensively while others do not at all.⁴ In the case of academic robotics projects, the lack of widespread UML use in earlier years may have been partly due to their typically relatively small scope. When projects are extensive and multi-asset, and the consequences of failure are high, the use of UML allows for greater efficiency of operation and reduced risk of project failure. Hence, contemporary complex robotics projects should benefit from appropriate tools to support their guidance and documentation, as has been the case for many large-scale projects, e.g., from the space industry (e.g., [42]) or the medical industry (e.g., [43]).

Many skilful and experienced programmers have not used UML.⁵ This is due to the lack of absolute necessity to use such tools both for the programming and the development of small projects. Hence, the first use of UML in a developers' team can consume a disproportionate amount of time. Another problem is the synchronisation of diagrams with source code. Here, the answer is, among other things, the appropriate level of generality of the diagrams so that unnecessary details are not mapped there. The automatic generation of code from the diagrams or the automatic generation of selected diagrams based on code can also be helpful. Finally, it is worth mentioning that UML diagrams do not constitute a complete system description. In particular, the description by diagrams can be complemented by mathematical expressions. A way of combining these two ways of description

⁴<https://creately.com/guides/advantages-and-disadvantages-of-uml/>

⁵<https://www.techwalla.com/articles/the-disadvantages-of-uml>

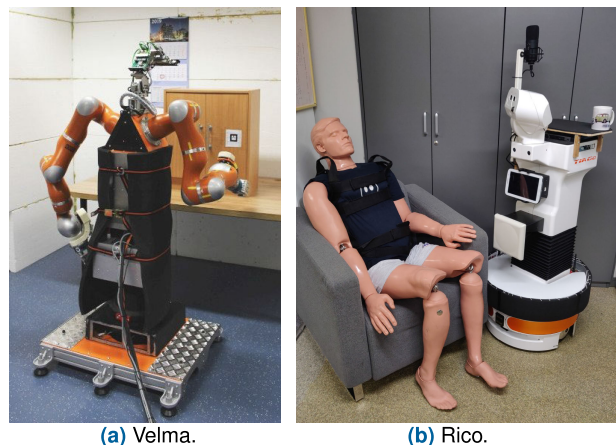


FIGURE 45. Robotic platforms specified with MeROS.

is presented in, e.g. [24]. SysML parametric diagrams also respond to this problem.

System development involves the use of a number of tools organised in toolchains. The degree of tools interaction varies. In software engineering, the aim is to create clear procedures for system development, indicating the dependencies between the successive stages of the development process. In robotics, there have been many works dedicated to toolchains (SmartMDS [44], RobotML [45], [46]), nowadays the ROS is common middleware (e.g. [47], BRIDE [48], HyperFlex [36]). MeROS is part of the toolchain used in the Robot Programming and Machine Perception Team at Warsaw University of Technology (WUT). Currently, on the base of MeROS, the controllers are specified for the following robotic platforms:

- Velma service robot [25], [33]⁶ (Fig. 45(a)) (Custom controller based on ROS 1 and Orocos for hardware control and simulation in Gazebo),
- assistive robot Rico [23], [32]⁷ (Fig.45(b)) (Extended PAL controller based on ROS 1 and Orocos, recent works with ROS 2 in simulation),
- MiniRyś – mobile robot with various modes of locomotion⁸ (Custom controller based on ROS 2 for hardware control and simulation in Gazebo),
- Dobot Magician – portable, 4-DOF robotic manipulator⁹ (Custom controller based on ROS 2 for hardware control).

At the forefront of the toolchain stays the modelling of the system with PIM using the EARL-based [24] SPSysML [49]. EARL is derived from agent theory [14], [21], [22]. Nowadays, in the intermediate stage, MeROS plays the major role as a PSM. Finally, FABRIC [13], as well as alternative approaches [50], [51] are used to support code generation. Current work concerns deepening the integration of MeROS with the rest of the toolchain.

⁶<https://www.robotyka.ia.pw.edu.pl/robots/velma>

⁷<https://www.robotyka.ia.pw.edu.pl/robots/rico>

⁸<https://www.robotyka.ia.pw.edu.pl/robots/miniryś>

⁹<https://www.robotyka.ia.pw.edu.pl/robots/magician>

REFERENCES

- [1] J. Bézivin, “In search of a basic principle for model driven engineering,” *Novatica J., Special Issue*, vol. 5, no. 2, pp. 21–24, 2004.
- [2] D. C. Schmidt, “Model-driven engineering,” *Computer*, vol. 39, no. 2, p. 25, 2006.
- [3] S. Kent, “Model driven engineering,” in *Proc. Int. Conf. Integr. Formal Methods*. Berlin, Germany: Springer, 2002, pp. 286–298.
- [4] E. Mnkandla, “About software engineering frameworks and methodologies,” in *Proc. AFRICON*, 2009, pp. 1–5.
- [5] O. Shehory and A. Sturm, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Berlin, Germany: Springer, 2014.
- [6] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Díaz, “Robotics software frameworks for multi-agent robotic systems development,” *Robot. Auton. Syst.*, vol. 60, no. 6, pp. 803–821, Jun. 2012.
- [7] E. Tzardoulis and P. Mitkas, “Robotic frameworks, architectures and middleware comparison,” 2017, *arXiv:1711.06842*.
- [8] A. Hentout, A. Maoudj, and B. Bouzouia, “A survey of development frameworks for robotics,” in *Proc. 8th Int. Conf. Modelling, Identificat. Control (ICMIC)*, Nov. 2016, pp. 67–72.
- [9] 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 *Proc. ICRA Workshop Open Source Softw.*, vol. 3, no. 3.2, p. 5, 2009.
- [10] W. F. Lages, D. Ioris, and D. C. Santini, “An architecture for controlling the Barrett WAM robot using ROS and OROCOS,” in *Proc. 41st Int. Symp. Robot. (ISR/Robotik)*, Jun. 2014, pp. 1–8.
- [11] 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 *Proc. IROS PR2 Workshop*, San Francisco, CA, USA, vol. 25, 2011, p. 30.
- [12] J. Pages, L. Marchionni, and F. Ferro, “TIAGo: The modular robot that adapts to different research needs,” in *Proc. Int. Workshop Robot Modularity, IROS*, vol. 290, 2016.
- [13] D. Seredyński, T. Winiarski, and C. Zieliski, “FABRIC: Framework for agent-based robot control systems,” in *Proc. 12th Int. Workshop Robot Motion Control (RoMoCo)*, K. Kozowski, Ed., 2019, pp. 215–222.
- [14] T. Kornuta, C. Zieliski, and T. Winiarski, “A universal architectural pattern and specification method for robot control system design,” *Bull. Polish Acad. Sci., Tech. Sci.*, vol. 68, no. 1, pp. 3–29, Feb. 2020.
- [15] M. Cholewinski, M. Janiak, and L. Juszkiwicz, “Software platform for practical verification of control algorithms developed for rescue and exploration mobile platform,” in *Proc. 20th Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2015, pp. 388–393.
- [16] H. Bruyninckx, “Open robot control software: The OROCOS project,” in *Proc. Int. Conf. Robot. Autom. (ICRA)*, vol. 3, 2001, pp. 2523–2528.
- [17] H. Bruyninckx, “OROCOS: Design and implementation of a robot control software framework,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Apr. 2002.
- [18] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ROS2,” in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2016, pp. 1–10.
- [19] J. Park, R. Delgado, and B. W. Choi, “Real-time characteristics of ROS 2.0 in multiagent robot systems: An empirical study,” *IEEE Access*, vol. 8, pp. 154637–154651, 2020.
- [20] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, “A survey of model driven engineering in robotics,” *J. Comput. Lang.*, vol. 62, Feb. 2021, Art. no. 101021.
- [21] C. Zieliski et al., “Variable structure robot control systems: The RAPP approach,” *Robot. Auton. Syst.*, vol. 94, pp. 226–244, Aug. 2017.
- [22] C. Zieliski and T. Winiarski, “Motion generation in the MRROC++ robot programming framework,” *Int. J. Robot. Res.*, vol. 29, no. 4, pp. 386–413, Apr. 2010.
- [23] W. Dudek and T. Winiarski, “Scheduling of a robot’s tasks with the TaskER framework,” *IEEE Access*, vol. 8, pp. 161449–161471, 2020.
- [24] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliski, “EARL—Embodied agent-based robot control systems modelling language,” *Electronics*, vol. 9, no. 2, p. 379, Feb. 2020.
- [25] M. Figat and C. Zieliski, “Parameterised robotic system meta-model expressed by hierarchical Petri nets,” *Robot. Auton. Syst.*, vol. 150, Apr. 2022, Art. no. 103987.

- [26] M. Wenger, W. Eisenmenger, G. Neugschwandtner, B. Schneider, and A. Zoitl, "A model based engineering tool for ROS component composition, configuration and generation of deployment information," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–8.
- [27] *OMG Systems Modeling Language—Version 1.6*, Open Manag. Group, Milford, MA, USA, Dec. 2019. Accessed: Feb. 20, 2020. [Online]. Available: <https://www.omg.org/spec/SysML/1.6/PDF>
- [28] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed. Amsterdam, The Netherlands: Elsevier, 2015.
- [29] M. R. V. Chaudron, W. Heijstek, and A. Nugroho, "How effective is UML modeling: An empirical perspective on costs and benefits," *Softw. Syst. Model.*, vol. 11, no. 4, pp. 571–580, Oct. 2012.
- [30] G. Canfora and M. Di Penta, "New frontiers of reverse engineering," in *Proc. Future Softw. Eng. (FOSE)*, May 2007, pp. 326–341.
- [31] B. Habib and R. Romli, "A systematic mapping study on issues and importance of documentation in agile," in *Proc. IEEE 12th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Aug. 2021, pp. 198–202.
- [32] J. Karwowski, W. Dudek, M. Węgierek, and T. Winiarski, "HuBeRo—A framework to simulate human behaviour in robot research," *J. Autom., Mobile Robot. Intell. Syst.*, vol. 15, no. 1, pp. 31–38, Jul. 2021.
- [33] T. Winiarski, S. Jarocki, and D. Seredyński, "Grasped object weight compensation in reference to impedance controlled robots," *Energies*, vol. 14, no. 20, p. 6693, Oct. 2021.
- [34] P. Pałka, C. Zieliński, W. Dudek, D. Seredyński, and W. Szykiewicz, "Communication-focused top-down design of robotic systems based on binary decomposition," *Energies*, vol. 15, no. 21, p. 7983, Oct. 2022.
- [35] N. H. Garcia, L. Deval, M. Lüdtke, A. Santos, B. Kahl, and M. Bordignon, "Bootstrapping MDE development from ROS manual code—Part 2: Model generation," in *Proc. ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, Sep. 2019, pp. 95–105.
- [36] D. Brugali and L. Gherardi, "HyperFlex: A model driven toolchain for designing and configuring software control systems for autonomous robots," in *Robot Operating System (ROS)*. Cham, Switzerland: Springer, 2016, pp. 509–534.
- [37] L. Gherardi, "Variability modeling and resolution in component-based robotics systems," Ph.D. dissertation, Dept. Eng., Univ. Bergamo, Bergamo, Italy, 2013.
- [38] M. Ladeira, Y. Ouhammou, and E. Grolleau, "RoBMEX: ROS-based modelling framework for end-users and experts," *J. Syst. Archit.*, vol. 117, Aug. 2021, Art. no. 102089.
- [39] P. Kumar, W. Emfinger, G. Karsai, D. Watkins, B. Gasser, and A. Anilkumar, "ROSMOD: A toolsuite for modeling, generating, deploying, and managing distributed real-time component-based software using ROS," *Electronics*, vol. 5, no. 4, p. 53, Sep. 2016.
- [40] S. P. Thale, M. M. Prabhu, P. V. Thakur, and P. Kadam, "ROS based SLAM implementation for autonomous navigation using turtlebot," in *Proc. ITM Web Conf.*, vol. 32, 2020, p. 01011.
- [41] S. Bisi, L. De Luca, B. Shrestha, Z. Yang, and V. Gandhi, "Development of an EMG-controlled mobile robot," *Robotics*, vol. 7, no. 3, p. 36, Jul. 2018.
- [42] S. Friedenthal and C. Oster, *Architecting Spacecraft With SysML: A Model-Based Systems Engineering Approach*. Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2017.
- [43] *Biomedical-Healthcare—Web Page*. Accessed: Aug. 4, 2023. [Online]. Available: <https://www.omgwiki.org/MBSE/doku.php?id=mbse:drug-delivery>
- [44] S. Dennis, L. Alex, L. Matthias, and S. Christian, "The SmartMDS toolchain: An integrated MDS workflow and integrated development environment (IDE) for robotics software," *J. Softw. Eng. Robot.*, vol. 7, no. 1, pp. 3–19, 2016.
- [45] S. Kchir, S. Dhoubib, J. Tatibouet, B. Gradousoff, and M. Da Silva Simoes, "RobotML for industrial robots: Design and simulation of manipulation scenarios," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–8.
- [46] S. Dal Zilio, P.-E. Hladik, F. Ingrand, and A. Mallet, "A formal toolchain for offline and run-time verification of robotic systems," *Robot. Auton. Syst.*, vol. 159, Jan. 2023, Art. no. 104301.
- [47] J. Wienke, A. Nordmann, and S. Wrede, "A meta-model and toolchain for improved interoperability of robotic frameworks," in *Proc. Int. Conf. Simulation, Modeling, Program. Auton. Robots*. Berlin, Germany: Springer, 2012, pp. 323–334.
- [48] A. Bubeck, F. Weisshardt, and A. Verl, "BRIDE—A toolchain for framework-independent development of industrial service robot applications," in *Proc. 41st Int. Symp. Robot. (ISR/Robotik)*, Jun. 2014, pp. 1–6.
- [49] W. Dudek, N. Miguel, and T. Winiarski, "SPSysML: A meta-model for quantitative evaluation of simulation-physical systems," 2023, *arXiv:2303.09565*.
- [50] T. Winiarski and K. Banachowicz, "Automated generation of component system for the calibration of the service robot kinematic parameters," in *Proc. 20th Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2015, pp. 1098–1103.
- [51] M. Figat and C. Zieliński, "Robotic system specification methodology based on hierarchical Petri nets," *IEEE Access*, vol. 8, pp. 71617–71627, 2020.



TOMASZ WINIARSKI (Member, IEEE) received the M.Sc./Eng. and Ph.D. degrees in control and robotics from the Warsaw University of Technology (WUT), in 2002 and 2009, respectively. He was the Head of the WUT Group with the AAL-INCARE Project "Integrated Solution for Innovative Elderly Care." He is an Assistant Professor with WUT. He is a member of the Robotics Group as the Head of the Robotics Laboratory with the Institute of Control and Computation Engineering (ICCE), Faculty of Electronics and Information Technology (FEIT). He is working on the modeling and design of robots and programming methods of robot control systems. The research targets service and social robots as well as didactic robotic platforms. His personal experience concerns the development and modeling of robotic frameworks, manipulator position–force and impedance control, and safety in robotic research. He is an INCOSE member.

• • •