

# SPSysML: A meta-model for quantitative evaluation of Simulation-Physical Systems

Wojciech Dudek<sup>1</sup>, Member, IEEE, Narcis Miguel<sup>2</sup>, Member, IEEE, Tomasz Winiarski<sup>1</sup>, Member, IEEE



**Abstract**—Robotic systems are cyber-physical systems (CPS) commonly equipped with multiple sensors and effectors, making them aware of dynamic environments, e.g. surrounding humans. Their complexity requires a multi-component structure and multi-state behaviour specification. This and potential environment harming makes the development of a robotic system challenging. Recently simulation of CPS components has become accurate enough to enable the Digital Twin (DT) concept realisation. However, it is unclear how to employ DT in robotic system development, e.g. in-development testing. During the system development, its parts evolve from simulated mockups to physical parts which run software deployed on the actual hardware. Therefore, a design tool and a flexible development procedure ensuring the integrity of the simulated and physical parts are required.

*Goal:* We aim to maximise the integration between a CPS's simulated and physical parts in various setups. As a result, the CPS has fewer components, requiring less testing. Furthermore, the better integration, the better simulation-based testing coverage of the physical part (hardware and software). Maximising the integration between simulation and the physical system allows using simulation for, e.g. parallel and safer development, faster and safer machine learning, and swift testing of CPS.

*Method:* In this work, we study the relationship between the simulated and physical parts of CPS. We propose a Domain Specification Language (DSL) that we refer to as SPSysML (Simulation-Physical System Modeling Language). It is based on Systems Modeling Language (SysML). SPSysML defines the taxonomy of a Simulation-Physical System (SPSys), being a CPS consisting of at least a physical or simulated part. In particular, the simulated ones can be DTs. We propose a SPSys Development Procedure (SPSysDP) that enables the maximisation of simulation-physical integrity of SPSys by evaluating the proposed factors based on SPSysML.

*Result:* SPSysDP is used to develop a complex robotic system for the INCARE project. In subsequent iterations of SPSysDP, the simulation-physical integrity of the system is maximised. As a result, the system model consists of fewer components and a greater fraction of the system components is shared between various system setups. The implementation and testing are conducted using the popular Robot Operating System (ROS) and Gazebo simulator; therefore, applying SPSysML and SPSysDP to robotic systems is straightforward.

*Conclusion:* SPSysML with SPSysDP enables the design of SPSys (including DT and CPS), multi-setup system development featuring maximised integrity between simulation and physical parts in its setups.

## 1 INTRODUCTION

Simulation is widely used in state-of-the-art development procedures for cyber-physical systems (CPS). Recent papers refer to Model-in-the-loop (MIL) [1], Software-in-the-loop (SIL) [2], Hardware-in-the-loop (HIL) [3] and Rapid Control Prototyping (RCP) [4] techniques. They are used selectively or are composed in sequence as, i.a., verification steps in various development procedures. Each of the techniques requires a simulation of the system parts. There are systems utilising the Digital Twin (DT) [5], [6] concept. They employ an accurate simulation of a system part for, e.g.:

- swapping a malfunctioned system part with the simulated one (DT) that mirrors the part's functionality in the simulated world,
- energy consumption analysis,
- system failure analysis and prediction,
- technology integration,
- real-time monitoring.

Some parts (software and hardware) of such a system interact with the real world and others with the simulated world. The set of the parts used in the real world we call physical embodiment, and the set of the parts used in the simulated world we call simulated embodiment. The Digital Twin is a part of the simulated embodiment mirroring a part of the physical embodiment. DT concept is used in numerous domains and applications [7]. They are systematically analysed by the authors of [8].

Robots comprise multiple devices and their controllers. In particular, autonomous mobile robots require a complex navigation system that features multiple closed control loops, e.g., drive controllers, trajectory controllers, and Simultaneous Localisation and Mapping (SLAM). Furthermore, the navigation system and the rest of the robot's functionality are used to execute the user request (like object transportation). In some cases, only a fraction of such a complex system must have DTs. In other cases, if the robot system has only the simulated embodiment, it is a demonstrator of the future product.

To clear up the taxonomy of systems featuring simulated and physical parts, we introduce the Simulation-Physical System (SPSys) concept (Fig. 1). This kind of system consists of at least one physical/simulated embodiment and a shared controller. If it has both embodiments, a simulated part can be a DT of a physical part. If it has only the physical

<sup>1</sup>Warsaw University of Technology, Institute of Control and Computation Engineering, Poland wojciech.dudek@pw.edu.pl, tomasz.winiarski@pw.edu.pl

<sup>2</sup>PAL Robotics, Barcelona, Spain narcis.miguel@pal-robotics.com

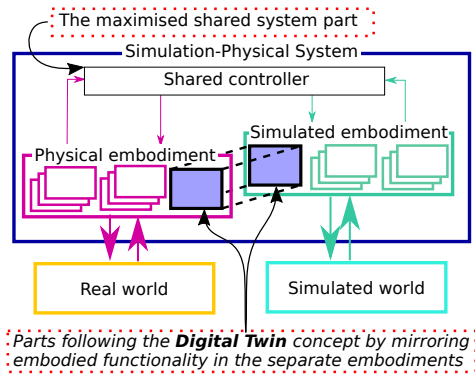


Fig. 1: The idea of Simulation-Physical System

embodiment, it is a CPS, and if it has only the simulated embodiment, it is a simulator.

In the development process, the system parts evolve from simulated mockups to the physical parts and the software deployed on the hardware. The system development procedure must guide the project team through the process. The system design created as a milestone in the procedure must specify all required parts in all development stages and compose them in the testing and deployment setups. Developing reliable systems, especially complex ones like robot systems, requires comprehensive unit and integration testing. Some CPS parts can be tested with simulated hardware only; therefore, additional parts that do not comprise the operational system are required (e.g. human simulator in a social robot case). Comprehensive testing is complex in test case specification and time-consuming in test implementation and execution. From this perspective, software reusability is a key to fast development of complex, reliable systems.

We aim to:

- Specify profiled-based requirements of Simulation-Physical Systems including multiple Digital Twins,
- Re-use of software between DT and its Physical Twin (PT) and between different system setups, improving system's reliability and resulting in more accessible and faster testing,
- The integrity boost between simulation and physical embodiments of SPSys enabling a comprehensive and parallel simulation-based testing of the system parts,
- Reflection of the dynamic environment in simulation to enable Digital Twins to observe exogenous actions.

To reach these goals, we propose a Domain Specification Language (DSL) named Simulation-Physical System Modeling Language (SPSysML). SPSysML models the requirements and the SPSys, in particular, SPSys consisting of simulated parts (used either in the system development or in the operational setup as DT/simulator). Based on SPSysML, we propose a requirement-based SPSys composition method. To measure the inter-embodiment integrity of the system, we propose quantitative optimisation factors for SPSys designs. The factors are used in the proposed SPSys Development Procedure (SPSysDP) (Fig. 2) to maximise the shared controller, minimise the system parts number and

maximise simulation-physical integrity. Our goal originates from the conclusion of [9]: *The digital models are mainly used to examine product performance(...). However, how to optimise the use of those models to enhance the design process and design collaboration is still needed to be investigated.*

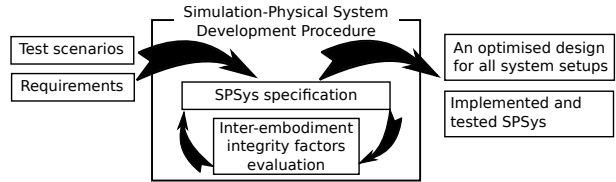


Fig. 2: The concept of DES development procedure

## 2 RELATED WORK

### 2.1 Mixing simulation-physical setups

In [10], the authors note the complexity of robotic systems and that the system components are often developed simultaneously by heterogeneous teams of researchers and developers. The authors suggest a demand for system simulation enabling rapid prototyping and an iterative development process. They propose the so-called agile Robot Development (aRD) concept that facilitates the integration of the hard real-time part and the no real-time part of the system utilising a Matlab/Simulink toolchain. The sim2real problem is known and mentioned in the recent research results, e.g., in the domain of self-driving cars [11]. However, to our knowledge, none of the existing works specifies a flexible robotic system meta-model that defines mixed simulation-physical setups of CPS under development.

Agile development implements the idea of rapid prototyping via, e.g. unit testing [12], [13]. These development strategies are often used in the domain of robotics. However, the tests evaluate just one specified simulation-physical setup of the system. Still, during development, almost every part of the system evolves from mockup (realised in simulation) to deployable software or hardware. Continuous integration is a valuable tool to manage the system implementation [14].

There are reviews on CPS [15] and DT [9] development. The CPS-related works focus on controlling a physical object and its model in a simulation environment with partially or fully simulated hardware. For example, the authors of [16] use the DT approach to run the digital embodiment in a safe virtual machine and confront the physical and digital embodiments to spot anomalies caused by a cyber-attack. Another work, like [17], employs simulation-driven machine learning for robots.

### 2.2 Meta-models for CPS

In the development process, a language is required to plan, conceptualise, and specify the system. Currently, the state-of-the-art approach is a model-based approach utilising a DSL. Unified Modeling Language (UML) is the most known language; however, System Modeling Language (SysML) [18] is proposed by the Object Management Group (OMG) to support the design, analysis, and verification of complex systems comprising software and

hardware components. There are different Model Driving Engineering approaches in the robotics domain [19]. For instance, Embodied Agent-Based cyber-physical control systems modelling Language (EARL) [20] is SysML specialisation for robotics which allows analysis and specification of the robotic system properties. Other DSLs support verification and testing of, e.g. industry 4.0 plants [21], or agent-based computational systems [22]). Besides, the work [23] describes the specification of safety compliance needs for critical computer-based and software-intensive systems. Confrontation of SPSysML with related models and specification methods is presented in Tab. 1. The related models focus on different purposes of Simulation-Physical Systems. Two of the works regard interaction between the Digital and PTs – C2PS [25], [16], three focus on simulation-based development of CPS – Digital Mockup [24], DEVSRT [27] and aRD [10], and the last one— HMLF [17] utilises DT concept in machine learning.

### 3 WORK NOVELTY AND PAPER ORGANISATION

None of the models presented in Sec. 2 specifies SPSys in general, especially including a physical part without a Digital Twin, simulated parts without a PT, or hybrid SPSys including Twins and non-Twin components. The key novel features delivered by SPSysML and SPSysDP are:

- Integrity evaluation with quantitative factors,
- SysML-based structure model and system development procedure,
- System design optimisation considering simulation and physical embodiments of the system,
- Multiple system execution and testing setups specification,
- Forcing simulation-based testing prior to physical embodiment development,
- Specification of Simulation-Physical System including multiple physical and simulated parts and multiple Digital Twins,
- Reflection of the dynamic environment in simulation to enable Digital Twins to observe exogenous actions.

Based on our experience in SPSys development (TIAGo robot [29], Velma robot [30], IRP6 robot<sup>1</sup>) and the literature analysis, we propose the SPSys Development Procedure focused on simulation-based testing coverage and software integrity between the system embodiments maximisation. Complex system structuring is a broad topic with solutions utilising various methods and strategies, e.g. top-down and bottom-up. In [31], we describe a binary communication-focused top-down approach for robotic systems. In this article, we propose a requirement-based top-down method customised for SPSys. SPSysML is a component-based Platform Independent Model (PIM), thus, can be applied to any SPSys. The result of SPSysDP is an implemented and tested Platform Specific Model (PSM) that can be launched in the specified setups. PSM implementation utilises platform-specific tools and software libraries. Therefore, to verify our approach, we design, implement, test and deploy a specific SPSys, including the TIAGo service robot [32] for

the INCARE (Integrated Solution for Innovative Elderly Care) project. The robot features an extended voice interface and additional devices to serve the elderly (e.g., in object transportation task Fig. 3a and fall assistance Fig. 3b). The resulting system utilises component-based frameworks like ROS [33], [34], OROCOS [35], Gazebo [36] (in particular *gazebo\_ros\_control* package [37]) that are a standard in robotics. Thanks to the description of the SPSysDP execution in the robotics domain, we present an example of DT implementation in robotic systems. To make the SPSys development easier for the community, we share the SysML profiles, the SPSysML meta-model and the example INCARE system model<sup>2</sup>.

First, we define SPSys requirements model in Sec. 4.1, the meta-model in Sec. 4.2 and its application procedure in Sec. 4.3. Next, we analyse the proposed evaluation factors of SPSys design and describe features of the systems that evaluation ends with corner values of the factors in Sec. 5. Finally, we verify our method in complex SPSys development for the INCARE project (Sec. 6). This work is concluded in 7.



(a) Transport task (b) Fall assistance task

Fig. 3: TIAGo robot in INCARE tasks execution

## 4 SIMULATION-PHYSICAL SYSTEM

Simulation-Physical System specification is based on SysML and EARL. In the following sections, we describe SPSysML and SPSysDP. SPSysML defines stereotype-based meta-models of requirements and structure. Description of a meta-model requires a notation for multiple blocks or instances of a stereotype (e.g., two instances of «Agent»). For this purpose, we append the stereotype with *s* (e.g., «Agent»*s*).

### 4.1 SPSysML – requirements meta-model for SPSys

We define a stereotype-based model of requirements (Fig. 4). The stereotypes explicate SysML requirement diagrams used for designing SPSys with SPSysML.

SPSysML specifies the requirements model as SysML profile that fosters requirement definition for a system being developed. The requirements can be defined based on various premises. In particular, they can derive from user requirements [38]. The SPSysML profile defines structural,

<sup>1</sup>Real: <https://youtu.be/wJpFcy99Gh0>, Simulation: <https://youtu.be/BjwcbSdouHw>

<sup>2</sup><https://github.com/RCPRG-ros-pkg/spsysml>

TABLE 1: Related works comparison with this work

System development method	EM	EIV	Formal meta-model	SCP	SO	Execution setups	DE	E	QE	MDT	Purpose
Digital Mockup [24]	I	-	-	ES	-	Static structure	✓	-	-	-	RT simulation for HIL testing
C2PS[25]	M	-	Own meta-model	Formal and detailed	BS	Adaptive	✓	-	-	N/D	Digital Twin in the cloud
[16]	M	✓	Dolev-Yao [26]	Formal and detailed	-	Static structure	✓	-	-	✓	Cyber-security, physical-digital twin synchronisation
DEVSRT [27]	I	-	DEVS [28]	General	-	Static structure	-	-	-	-	Simulation to embedded continuous development
aRD[10]	M	-	-	General	-	✓	✓	-	-	-	RT & NonRT parts integration
HMLF [17]	I	-	-	ES	-	Static structure	-	-	-	-	Simulation-driven ML
SPSysML	M	✓	SysML	Formal and detailed	DDO	✓	✓	✓	✓	✓	Structure optimisation for Simulation-Physical Systems

EM – Embodiment modularity, EIV – Embodiment integrity verification, SCP – System creation procedure, SO – Structure optimisation, DE – If the system execution configuration can contain parts of both embodiments, E – If uncontrolled agents of the physical environment can be modelled in simulation, QE – If quantitative evaluation factors for system design are proposed, MDT – If enables multiple Digital Twin design of the system, I – Integrated system, ES – Example system as the method application procedure, M – System decomposable to multiple simulation/physical parts, n/d – No data, BS – The best selection from the previously designed, DDO – Dual-embodied design optimisation, N/D – Possible, however, not defined,

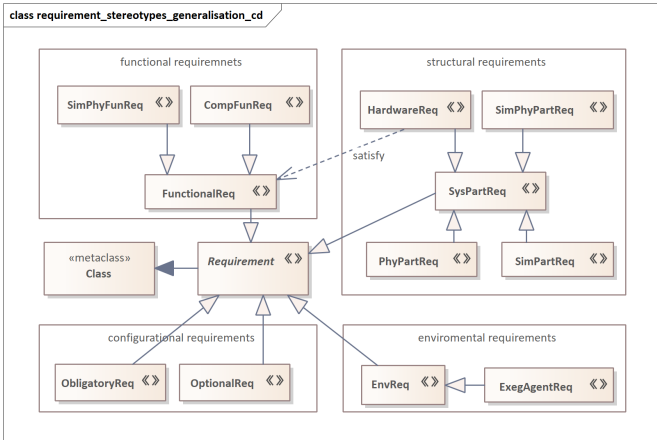


Fig. 4: Model of system requirements defined for SPSys

functional, configurational and environment requirements. Auxiliary sequence diagrams presenting the concept of the system behaviour and use case diagrams are useful for requirements definition. Before SPSys requirements specification, its environment must be analysed. Therefore, in SPSysML we distinguish Environment requirements. They can be of various requirements types. In this article, we focus on «ExegAgentReq» that describe exogenous agents interacting with the world alongside the system. In SPSysML, the system is decomposed into parts based on «SysPartReq». Each «SysPartReq» is classified to elementary «HardwareReq» or more general «PhyPartReq», «SimPartReq» or «SimPhyPartReq». «PhyPartReq» determines a part interacting only with the Physical World (even during the system development, e.g., during parallel development of the connected parts), and its simulated embodiment is not required (or cannot be created) during the system development. «SimPartReq» specifies a part interacting only with the Simulated World (e.g., mock-ups or demonstrators). «SimPhyPartReq»

interacts with both Worlds (e.g. realised with a pair of DT and PT). «HardwareReq»s are in a *satisfy* relationship with «FunctionalReq»s. This means hardware parts specified with a «HardwareReq» must enable realisation of the functionality specified with the given «FunctionalReq». The configurational requirements specify if a system part is obligatory to launch in any of its configurations («ObligatoryReq») or if it is not («OptionalReq»).

## 4.2 SPSysML – structure meta-model for SPSys

SPSysML derives from EARL [20] version 1.3 [39]. We use a Group of Subsystems («GpSubsys») to gather Subsystems with a specific common properties, and a Group of Agents («GpAgents») to organise the Agents (composed of Subsystems) cooperating for a defined aim in the system. In a SPSys we differentiate three specific Agent stereotypes that derive from «Agent» defined in EARL (Fig. 5):

- *Physical Agent* («PhyAgent») – Runs only in the physical embodiment (in particular Agent interacting with or sensing real world),
- *Simulation Agent* («SimAgent») – Runs only in the simulated embodiment (in particular Agent interacting with or sensing simulated world),
- *Hybrid Agent* («HybAgent») – Runs in both embodiments with neither interaction nor perception of any world.

The Groups of Agents called *World Mirror Group of Agents* aggregate «SimAgent»s that are Digital Twins of the Agents in the Physical World that are not under the control of the system, i.e., execute exogenous actions in the Simulated World. Groups of this type exist if the system works in a dynamic environment (e.g., an environment with human inhabitants). For example, a *World Mirror Group of Agents* modifies the Simulated World as humans do in the Physical World.

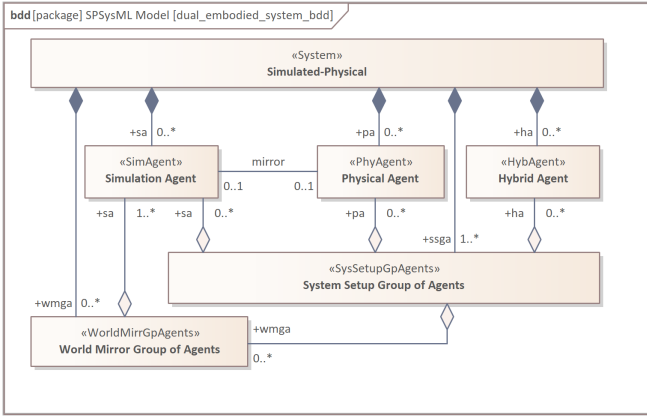


Fig. 5: Simulation-Physical System composition

#### 4.2.1 Digital Twins in SPSys

Integrity between SPSys embodiments is crucial. Therefore, the existence of DT of physical parts is advisable and common. We define the `mirror` relationship between «PhyAgent» and «SimAgent» to model the relationship between DT and its PT. To enable multi-agent DT for a single «PhyAgent» or single-agent DT for multi-agent PT, we introduce «MirrPhyGpAgents» and «MirrSimGpAgents». They aggregate Agents of different stereotypes (Fig. 6).

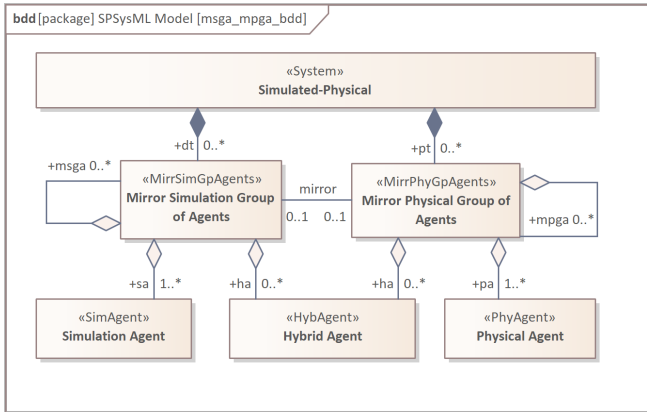


Fig. 6: SPSys composed of 0..\* Digital Twins (+dt) and Physical Twins (+pt) realised with «MirrSimGpAgents» and «MirrPhyGpAgents» accordingly

The definition of the `mirror` relationship is as follows:

**Definition 4.1 (Mirror relationship).** Two Groups of Agents are said to be in a mirror relationship if their input buffers and goals are the same and affect Simulated and Real Worlds congruently. The relationship is an association between Digital and Physical Twins.

It is worth noting that the same stimuli of the mirroring Groups cause corresponding reactions in specific worlds, in the Simulated World for *Mirror Simulation Group of Agents* and in the Physical World for *Mirror Physical Group of Agents*. The particular reaction results from the requirements of the specific system and does not need to be identical within the embodiments. The mirror relationship does not propagate

automatically to all the Agents aggregated in the mirroring Groups, so agents from «MirrSimGpAgents» does not mirror all agents of mirroring «MirrPhyGpAgents» and vice versa.

The amount of mirroring Agents should be maximised to maximise coverage of simulation-based testing and profits of DT. Additionally, the complexity and quantity of mirroring Agent Subsystems should be minimised to boost the modularity and integrity of the system. Otherwise, the system could be composed of just two mirroring agents. If a computational functionality is required in both embodiments, a «HybAgent» should be designed for this purpose.

#### 4.2.2 Subsystem types

The Agents are built with Subsystems of different types based on EARL. In EARL the central computational part of an Agent is the Control Subsystem «ContSubsys». An Agent communicates with other Agents using communication buffers of its «ContSubsys». To percept the environment, an Agent uses Real Receptors («RealRecs»), and to aggregate and preprocess stimuli, it uses Virtual Receptors («VirtRecs»). To affect the environment, an Agent uses Real Effectors («RealEffs»). To preprocess «ContSubsys» commands to signals for «RealEffs» it uses Virtual Effectors («VirtEffs»). SPSys interacts with both the simulated and physical world; thus, SPSysML differentiates between the types of the above Subsystems— Simulated, Physical, and Dual-embodied. To achieve maximum integrity between the simulated and physical embodiments, all «ContSubsys»s should be Dual-embodied («SimPhyContSubsys» stereotype) and constitute the general concept of the shared controller, recall Fig. 1. For iterating design purposes, embodiment-specific Control Subsystems concepts («PhyContSubsys» and «SimContSubsys») may be helpful. The other Subsystems can be Simulated or Physical. SPSysML defines «GpSubsys»s that aggregate receptors and effectors considering their embodiment and the world they interact with (Fig. 7, Fig. 8). As one «SimPhyContSubsys» can communicate with multiple Simulated/Physical Virtual Receptors and Effectors, we define four «GpSubsys»s aggregating Physical/Simulated Virtual/Real Subsystems. The data flow and communication links for «SimPhyContSubsys» interacting with Simulated and Physical Worlds is shown in (Fig. 9).

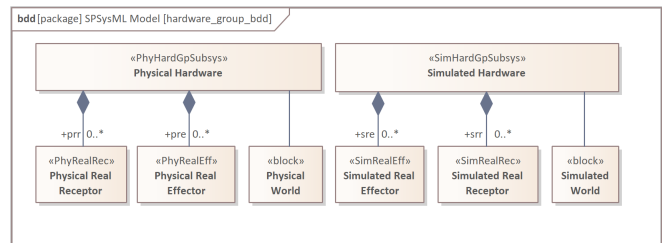


Fig. 7: Embodiments of Real Effectors and Receptors

#### 4.2.3 Realisation of the Subsystem and Agent types

Subsystems in a «PhyHardGpSubsys» expose interface realised with a communication interface controller used by the effector or sensor (e.g., Linux kernel driver for Inter-Integrated Circuit (I<sup>2</sup>C) or network interface card).

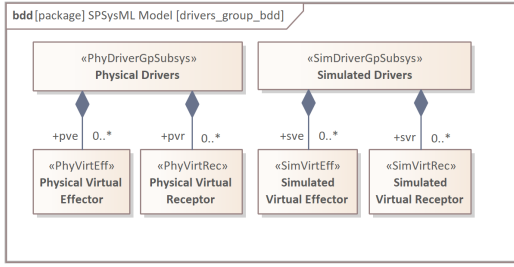
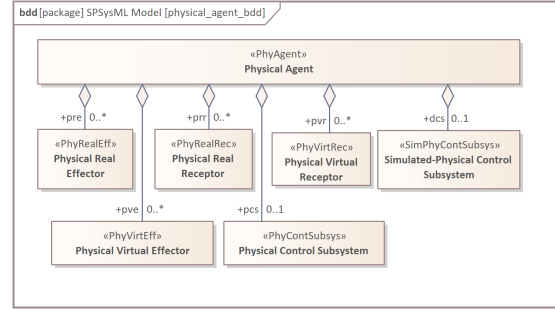


Fig. 8: Embodiments of Virtual Effectors and Receptors



(a) Physical Agent

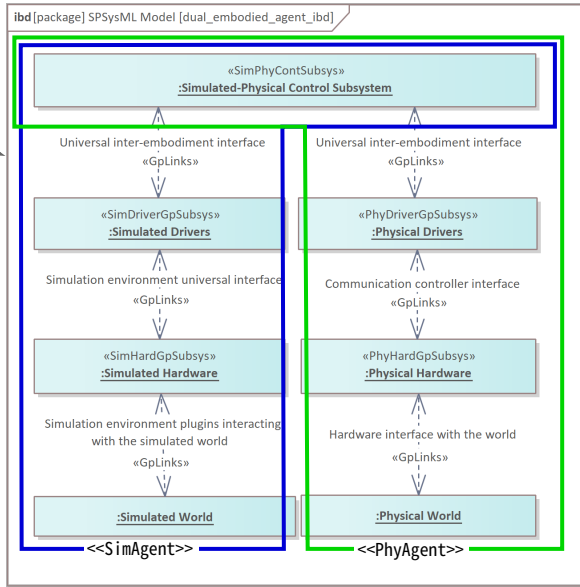


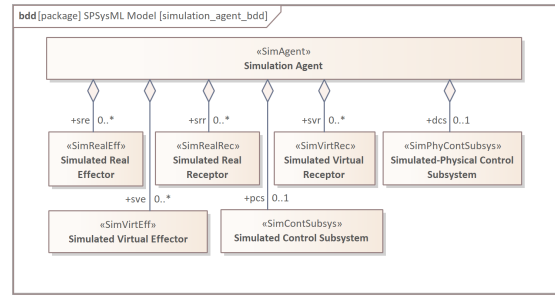
Fig. 9: Interfaces between Subsystem Groups on example, where two agents share «SimPhyContSubsys».

To these interfaces «PhyDriverGpSubsys»s are connected and they expose an embodiment-abstract interface for a «ContSubsys» of the Agent. Subsystems in «SimHardGpSubsys» expose interfaces realised with objects of an interface class defined for the utilised simulation environment (e.g., gazebo::ModelPlugin for «SimRealEff» and gazebo::SensorPlugin for «SimRealRec» in Gazebo<sup>3</sup>). «SimDriverGpSubsys» connects to these interfaces and exposes embodiment-abstract interface for «ContSubsys»s.

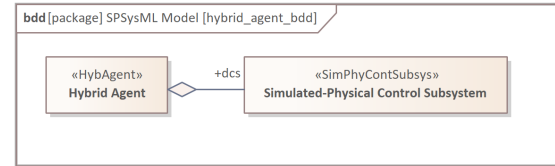
Each Agent type defined in SPSysML aggregates a particular number of Subsystems of a specific type (Fig. 10).

It should be noted that the Basic EARL meta-model defines only one «ContSubsys» in an Agent, and inter-agent communication is handled only by the «ContSubsys». In SPSysML, we differentiate between specific types of «ContSubsys» for the embodiments; still, an Agent can aggregate just one specialisation of «ContSubsys». SPSys can be deployed in various setups (e.g. testing setups). The setups are a Group of Agents fulfilling the system's functionality in the setup. For the system setup definition, we use «SysSetupGpAgents» to specify the Group of Agents working in the setup. The specific «SysSetupGpAgents» is derived from the system's

<sup>3</sup>popular simulation environment for robots



(b) Simulation Agent



(c) Hybrid Agent

Fig. 10: Subsystems aggregated by Agent classes

requirements and test scenarios. This problem is considered in the SPSysDP section (Sec. 4.3).

### 4.3 Simulation-Physical System Development Procedure (SPSysDP)

SPSysML defines system parts and the relations between them to enable SPSys specification. Various agent-based system development procedures can be used for SPSysML-based systems; however, we define one that enables inter-embodiment integrity optimisation for SPSys (Fig. 11). The main activities of the procedure can be executed in different development approaches, traditional, agile, or hybrid. In agile and hybrid approaches, the system is specified and implemented partially in an iterative manner. In the procedure shown in Fig. 11, there are four decision nodes D1, D2, D3, and D4. D1 checks if the requirements of the considered system part are comprehensive and if the system parts and functions, considering the system's test scenarios, are expressed in the requirement diagrams. D2 checks if the structure evaluation is satisfactory. D3 checks whether any detailed analysis or requirements modification is required based on the design iteration, and D4 checks if the required «SysSetupGpAgents»s are implemented and tested successfully. If needed, the procedure enables complete reiteration from the design stage.

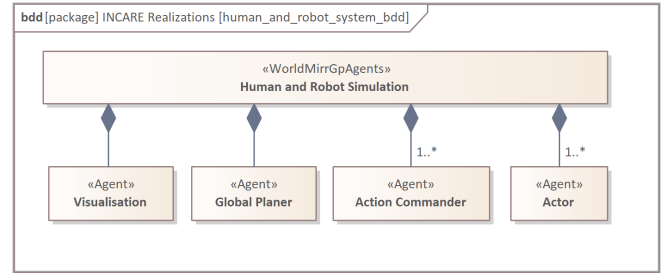
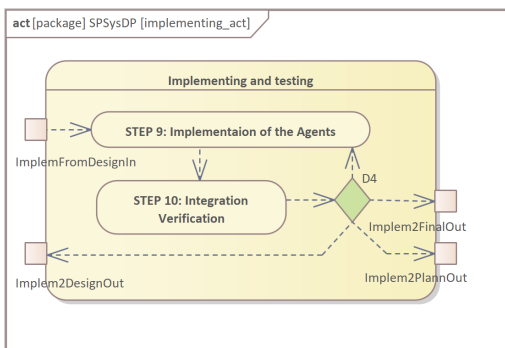
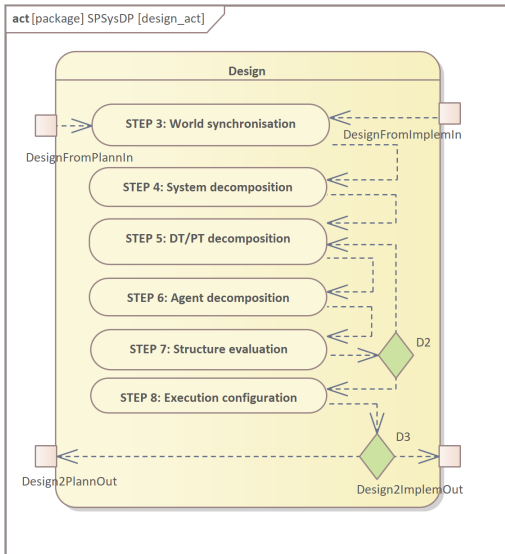
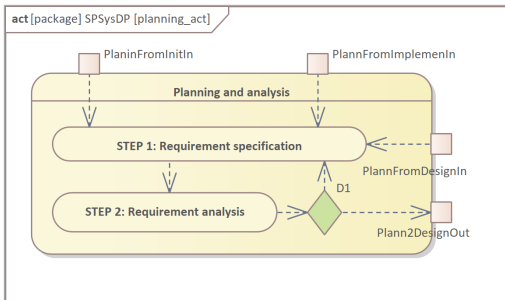
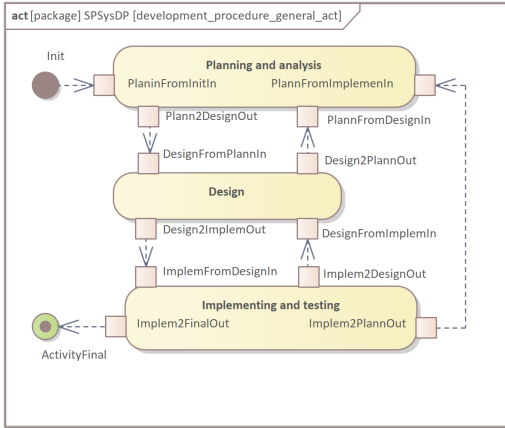


Fig. 12: Agent types of HuBeRo framework [40] used in SPSys as «WorldMirrGpAgents» that mirrors humans in Simulated World.

### System planning and analysis

- **Step 1** (Requirement specification): The requirements are specified following the model defined in Sec. 4.1. It is recommended to define system use cases and typical interactions between system parts on sequence diagrams while specifying the requirements.
- **Step 2** (Requirement analysis): Analysis of the requirements considering the system's test scenarios. In case it is needed, modifications to these requirements are made.

### System design

- **Step 3** (World Synchronization): This step is not applicable if the system works in a static environment (there are no exogenous actions of any non-system agent executed on the system's environment). In other cases, simulation of external agents (e.g. humans) is required, and it is done by *WorldSync* «WorldMirrGpAgents». Each «SimAgent» composed in «WorldMirrGpAgents» manages one «ExegAgentReq» specified in Steps 1-2. The key design aspects for the «SimAgent»s are the actions affecting the Simulated World and their model sensed by the SPSys' receptors. To model humans as «SimAgent», we propose our framework Human Behaviour in Robotics Research (HuBeRo) [40]. It specifies and implements agents mirroring human behaviours and their physical models in simulation (Fig. 12).
- **Step 4** (System decomposition): The system is decomposed into Agents systematically based on the parts in the requirements. For each «PhyPartReq» and «SimPartReq» a part «PhyAgent» or part «SimAgent» are created. Each «SimPhyPartReq» that has only «ComputationalFunReq» becomes part «HybAgent». The other «SimPhyPartReq»s become pairs of DT/PT, thus, are realised with a pair of mirroring part «MirrPhyGpAgents» and part «MirrSimGpAgents» (Fig. 13),
- **Step 5** (DT/PT decomposition – Mirroring Agent Groups specification): Each mirroring part «MirrPhyGpAgents» and part «MirrSimGpAgents» is iteratively decomposed to more groups to finally reach mirroring groups that each can be realised with

Fig. 11: SPSys Development Procedure for Simulation-Physical Systems.

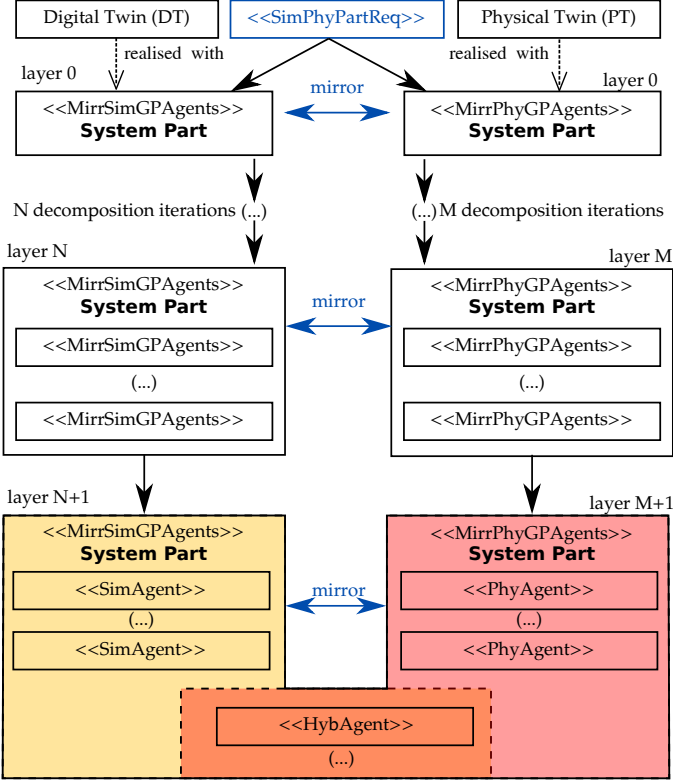


Fig. 13: Requirement-based SPSys decomposition, where *mirror* relationship constituting Physical and Digital Twins is obligatory for *«SimPhyPartReq»*. As a result of the decomposition, DT/PT may share *«HybAgent»*s.

a single *«PhyAgent»* or *«SimAgent»*. Each result of the decomposition iteration represents a layer of the initial Group. The *mirror* relationship is set between *«MirrSimGpAgents»* and *«MirrPhyGpAgents»* and is specified in each layer of the decomposition. The decomposition scheme is shown in Fig. 13.

- **Step 6 (Agent decomposition):** Each *«Agent»* is decomposed to Subsystems. Various approaches can be used depending on the requirements formulation; however, we advise a layered bottom-up one with a definition of each layer interface. First, the hardware assigned to each *«Agent»* is expressed as *«PhyRealRec»*s, *«PhyRealEff»*s, *«SimRealEff»*s and *«SimRealRec»*s (constituting Agent hardware layer). Next, the interfaces of *«PhyVirtRec»*s, *«PhyVirtEff»*s, *«SimVirtEff»*s and *«SimVirtRec»*s are defined based on Sec. 4.2.2 (constituting Agent driver layer). The target is to develop a *«SimPhyContSubsys»* (constituting Agent control layer) that manages mirroring Agents' behaviour, uses an embodiment-common interface and exposes an inter-agent interface. However, during the design process and testing, a temporary *«SimContSubsys»* and *«PhyContSubsys»* are helpful. Mirroring Agents consist of identical control layers; however, the Agents differ in drivers and hardware layers. To enable embodiment abstraction for the control layer in mirroring Agents, the driver layer must fill the gap between the hardware and controller layers in the specific embodiments.

- **Step 7 (Structure evaluation):** In each design iteration, the structure is evaluated. We propose the following evaluation factors:

- **Controller integrity factor** ( $I = \frac{c_{dcs}}{c_{All}}$ ), where  $c_{dcs}$  and  $c_{All}$  are the cardinalities of *«SimPhyContSubsys»*s and all *«ContSubsys»*s accordingly,
- **Mirror integrity factor** ( $M^n = \frac{c_{dcs}^n}{c_{All}^n}$ , where  $n$  is the evaluated pair of mirroring *«MirrPhyGpAgents»* and *«MirrSimGpAgents»* composing one DT,  $c_{dcs}^n$  and  $c_{All}^n$  are the counts of *«SimPhyContSubsys»*s and all *«ContSubsys»*s accordingly in  $n^{th}$  DT,
- **Driver generalisation factor** ( $G = \frac{r_u}{r}$ ), where  $r_u$  is the count of Real Subsystems aggregated in an Agent with a *«SimPhyContSubsys»* and  $r$  is the count of all Real Subsystems.
- **Digital Twin coverage** ( $D = \frac{a_m^P}{a_{All}^P}$ ), where  $a_m^P$  is the count of *«PhyAgent»* aggregated in *«MirrPhyGpAgents»* with a mirror relationship with a *«MirrSimGpAgents»*, and  $a_{All}^P$  is the count of all *«PhyAgent»*.

The factors:

The structure optimisation target is the maximisation of the embodiment-common part of the system ( $I$ ,  $M$ ,  $G$ ,  $D$  factors) to maximise simulation-based testing coverage. The share of physical parts mirrored with Digital Twins ( $D$ ) additionally boosts the system's robustness, as DT can replace malfunctioned hardware. The detailed analysis of the proposed factors is described in Sec. 5. The optimisation target can be defined with the above factors or others that can be defined for a specific system. If the evaluation result is unsatisfactory, the system should be redesigned (starting from **Step 5**) following the factors' maximisation advices.

- **Step 8 (System setups):** Based on the *optional* stereotypes in the requirements, all possible system setups emerge. Each setup (*«SysSetupGpAgents»*) is specified with an Internal Block Diagram. The diagram shows the Agents composing given *«SysSetupGpAgents»*, their communication and interaction with Simulated and Physical Worlds. It should be noted that the system's application setups are just a starting point for *«SysSetupGpAgents»*s specification. For each implementation testing scenario, a *«SysSetupGpAgents»* should be specified. DTs are broadly used in development optimisation and CPS development safety improvement; therefore, in particular for testing *«MirrPhyGpAgents»*, a *«SysSetupGpAgents»* consisting of *«MirrSimGpAgents»* (DT of the *«MirrPhyGpAgents»*) should be defined. The next step of this procedure makes sure that the *«SysSetupGpAgents»* with *«MirrPhyGpAgents»* is implemented and tested prior to *«SysSetupGpAgents»* including *«MirrPhyGpAgents»*.

## System implementation and testing

- **Step 9 (Implementation of the Agents):** Starts with a *«SysSetupGpAgents»*, whose simulation implemen-



tation factor is the highest. The simulation implementation factor for example  $ex1$  «*SysSetupGpAgents*» is  $\text{sif}_{ex1} = \frac{s_{ex1} + h_{ex1}}{a_{all}}$ , where  $s_{ex1}$  and  $h_{ex1}$  are cardinalities of the unimplemented «*SimAgent*»s and «*HybAgent*»s in  $ex1$  «*SysSetupGpAgents*» and  $a_{all}$  is a number of Agents in the system. Before implementation, Subsystems must be translated to a PSM. For robotic systems utilising ROS, we propose MeROS DSL [41]. Implementation should include unit testing of each Subsystem and Agent.

- **Step 10** (Integration verification): Test scenarios execution for all fully implemented «*SysSetupGpAgents*».

## 5 THE DESIGN EVALUATION FACTORS ANALYSIS

This section describes the interpretation of the evaluation factors and which system features they evaluate. We describe characteristics of corner case SPSys that scores maximum or minimum values of the design evaluation factors. This gives a basic intuition on the correlation between the factor values and the SPSys features. The interpretation of the evaluation factors is as follows:

- $I$  – is a share of the software controller common between the embodiments. Virtual Subsystems are not considered, as their count may be related to the Real Subsystem counts in each embodiment. It is maximised by the reduction of «*SimContSubsys*»s and «*PhyContSubsys*»s in favour of an inter-embodiment «*SimPhyContSubsys*»s, The higher  $I$  is, the more software components are shared between the simulation and physical embodiments. At maximum ( $I=1$ ), all hardware abstract parts of the system are common.
  - $I = 0$ : There are no «*SimPhyContSubsys*», only «*PhyContSubsys*» or «*SimContSubsys*». The system's parts in simulated and physical embodiments are disjunctive. Simulation-based testing is not possible. The system's functions in the simulation may be completely different from those in the physical embodiment.
  - $I = 1$ : All Control Subsystems are «*SimPhyContSubsys*», and there are no «*PhyContSubsys*» or «*SimContSubsys*». This means all hardware abstract parts of the system are common between its embodiments, and the coverage of simulation-based testing is maximised and allows integration testing in simulation.
- $M^n$  – is a share of the system parts common between Physical and Digital Twins composing  $n^{th}$  pair twins (managing «*SimPhyPartReq*»). It is similar to  $I$  but within the scope of  $n^{th}$  pair of Physical and Digital Twin. Tips for maximisation of the  $M^n$  factor are:
  - extraction of common functions as «*HybAgent*»s from «*MirrPhyGpAgents*» and «*MirrSimGpAgents*»
  - and/or redesign of interfaces between a «*SimContSubsys*» and «*SimDriverGpSubsys*» and between a «*PhyContSubsys*» and «*PhyDriverGpSubsys*» to emerge a common «*SimPhyContSubsys*» from the «*SimContSubsys*» and «*PhyContSubsys*»,

- $G$  – is a share of Real Subsystems (hardware) controlled by a «*SimPhyContSubsys*» (shared controller). It expresses hardware control integrity between the embodiments.
  - $G = 0$ : All Hardware parts are controlled by embodiment-specific Control Subsystems. The causes of this depend on a specific case:
    - \* For Physical Hardware without a DT, it means the interface to hardware is embodiment-specific; thus, extending the system with DT of the hardware is complicated,
    - \* For Physical hardware with a DT, it means the Hardware Drivers interface of Physical and Digital Twins differ, and the software using the interface differs between the embodiments. This means the system part designed as DT of the Physical hardware is not a proper DT.
  - $G = 1$ : All Hardware parts are controlled by embodiment-abstract Control Subsystems; thus, the Agents managing hardware are interchangeable between the embodiments, or future Digital/Physical Twin integration for Physical/Simulated Hardware is straightforward.
- $D$  – is a share of hardware and its controllers mirrored with a DT. Its increase boosts coverage of simulation-based testing of hardware controllers and system robustness utilising the DT concept. If  $D = 0$ , there are no DTs in the system; if  $D = 1$ , all Physical Hardware parts have DTs.

Based on the factors' values, one can evaluate the system design in terms of the following:

- Safety and hardware independence during software testing – based on  $I$  (for system scope),  $M^n$  (for  $n^{th}$  DT),
- Simulation-based testing and failure examination/prediction of the system parts – based on  $D$
- Inter-embodiment integrity of hardware controllers and readiness for simulation-based hardware testing – based on  $R$ .

Maximisation of the factors is not always required, and the optimisation goal can be set at a different point in the factors' space. The goal depends on the specific system requirements. However, the factors' values inform the designer about the inter-embodiment integrity of the system design, so her/his decision is conscious.

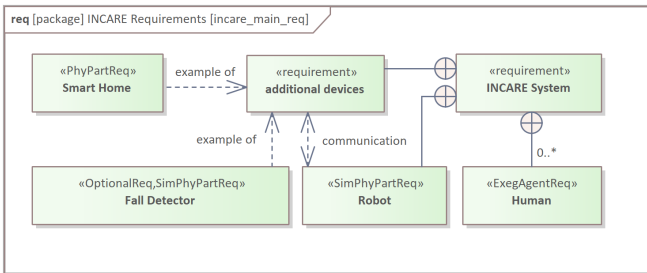
## 6 VERIFICATION

To verify the work result we execute the SPSysDP to design, implement and test a complex SPSys utilising a service robot for the INCARE project. The INCARE system idea and requirements are published in [42]. The framework model developed to manage robot tasks is published in [43], [44]. Developing a complex system requires different implementations of its parts in various development phases. In

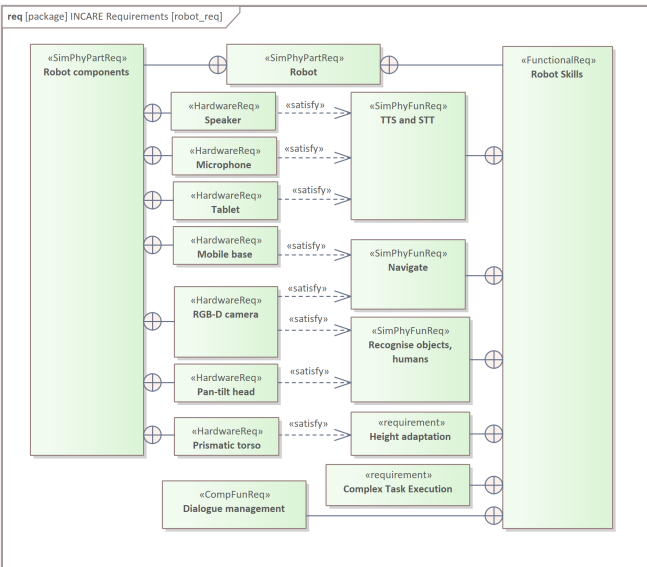
INCARE, we use some commercial products like the TIAGO robot with its control system. We develop and integrate new parts into it (e.g., human fall detector and TIAGO audio interface extension). In such a case, developing one part requires a dummy of another part while being developed simultaneously. Developers can simulate the underdeveloped parts while the physical devices are under construction or development. In the case of INCARE, we use a dummy for the human fall detector while developing a TIAGO robot application helping the elderly that may fall over. In this section, we describe the application of the SPSysML and SPSysDP to the INCARE system which is a Simulation-Physical System. We use and modify the TIAGO robot in this project; thus, we specify its hardware and controller as a part of the INCARE system specification.

The result of each step of SPSysDP is as follows:

**Step 1 and 2:** we specify the structural and functional requirements based on the general requirements of the INCARE project. The general and the robot-specific requirement diagrams are shown in in Fig. 14. The re-



(a) INCARE structural requirements



(b) The Robot requirements, where TTS and STT are text-to-speech and speech-to-text functions

Fig. 14: Example part of the INCARE requirements

quirements were accepted after some Step 1 $\leftrightarrow$ Step 2 iterations. These iterations led, for instance, to the decomposition of the *Communication with humans* «*SimPhyFunReq*» to *TTS and STT* «*SimPhyFunReq*» and *Dialogue management* «*CompFunReq*».

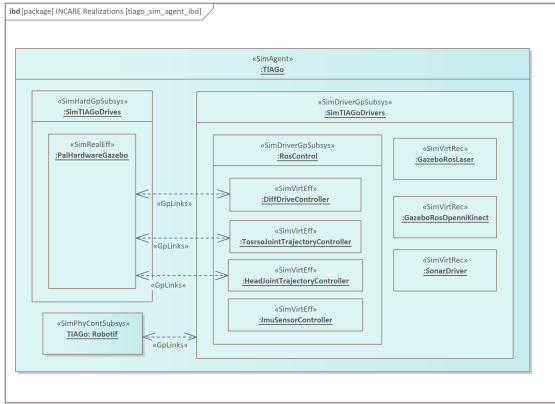
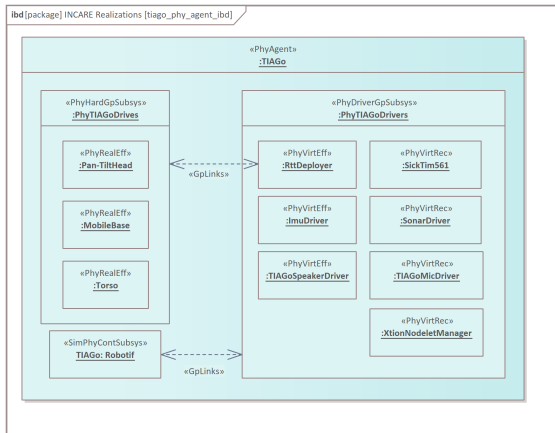
**Step 3:** The robot in the INCARE project coexists with humans; thus, we utilise *WorldSync* «*WorldMirrGpAgents*» to manage humans in the Simulated World. Detailed specification and realisation of *WorldSync* «*WorldMirrGpAgents*» is available in [40].

**Step 4 and 5:** In the final design iteration the system is decomposed to 5 embodiment-specific Agents, one for each system part: *TIAGO* «*SimAgent*» mirroring *TIAGO* «*PhyAgent*», *FallDetector* «*SimAgent*» mirroring *FallDetector* «*PhyAgent*» and *SmartHome* «*PhyAgent*». Additionally, there are 3 Hybrid Agents, one for each computational function of the system: *ComplexTaskExecution* «*HybAgent*», *Talker* «*HybAgent*», *FakeAudio* «*HybAgent*». *TIAGO* «*SimAgent*», *FakeAudio* «*HybAgent*» and *Talker* «*HybAgent*» compose *Robot* «*MirrSimGpAgents*», and *TIAGO* «*PhyAgent*» and *Talker* «*HybAgent*» compose *Robot* «*MirrPhyGpAgents*».

**Step 6:** To provide an example we describe the final decomposition of *TIAGO* «*SimAgent*» and *TIAGO* «*PhyAgent*» Agents only. Each robot hardware component is specified as either «*PhyRealRec*», «*PhyRealEff*», «*SimRealRec*», or «*SimRealEff*». We use Gazebo simulation environment; thus, «*SimRealRec*s» and «*SimRealEff*s» expose gazebo::SensorPlugin and gazebo::ModelPlugin interfaces accordingly. As the physical robot, we use PAL Robotics' TIAGO; thus, «*PhyRealRec*s», «*PhyRealEff*s» interfaces are adequate Linux Kernel drivers managing communication with the devices. The Simulated Drivers connect to the gazebo::SensorPlugin and gazebo::ModelPlugin interfaces and expose the robot state info and typical ROS topics/services (e.g. *JointStateInterface* and *EffortJointInterface* for *MobileBaseController* «*SimVirtEff*»<sup>4</sup> and */scan* ROS topic for *lidar* «*SimVirtRec*»). If Simulated World is the Gazebo environment, «*SimVirtEff*s» and «*SimVirtRec*s» are usually implemented as Gazebo Plugins. «*PhyDriverGpSubsys*» connects to Linux Kernel drivers and, as a whole Group, exposes to «*ContSubsys*» identical interfaces as «*SimDriverGpSubsys*». Fig. 15 shows IBD of both *TIAGO* «*SimAgent*» and *TIAGO* «*PhyAgent*» and their common *RobotIF* «*SimPhyContSubsys*».

**Step 7:** The final structure evaluation resulted with:  $I=1$ ,  $M^{Robot}=1$ ,  $M^{FallDetector}=1$ ,  $G=1$ , and  $D=0.67$ . The result means the structure consists of no «*SimContSubsys*» or «*PhyContSubsys*», and one «*PhyAgent*» is not mirrored by a DT (*SmartHome* «*SimAgent*»). The lack of DT for *SmartHome* «*SimAgent*» results from the requirements (Fig. 14a), where *SmartHome* is not «*SimPhyPartReq*»; thus, this is an informed decision of the designer.  $G=1$  means *SmartHome* «*SimAgent*» has «*SimPhyContSubsys*», therefore, its DT integration to the system in the future is straightforward. One of the previous design iterations resulted with:  $I=\frac{5}{7}=0.71$ ,  $M^{Robot}=1$ ,  $M^{FallDetector}=0$ ,  $G=\frac{20}{22}=0.91$ ,  $D=0.67$ . In this iteration the *FallDetector* «*SimAgent*» and *FallDetector* «*PhyAgent*» use embodiment specific control layer («*SimContSubsys*» and «*PhyContSubsys*»), because *FallDetector* «*SimAgent*» consists «*SimContSubsys*s» only. To increase  $M^{FallDetector}$  and  $G$  the common part of *FallDetector* «*SimContSubsys*» and *FallDetector*

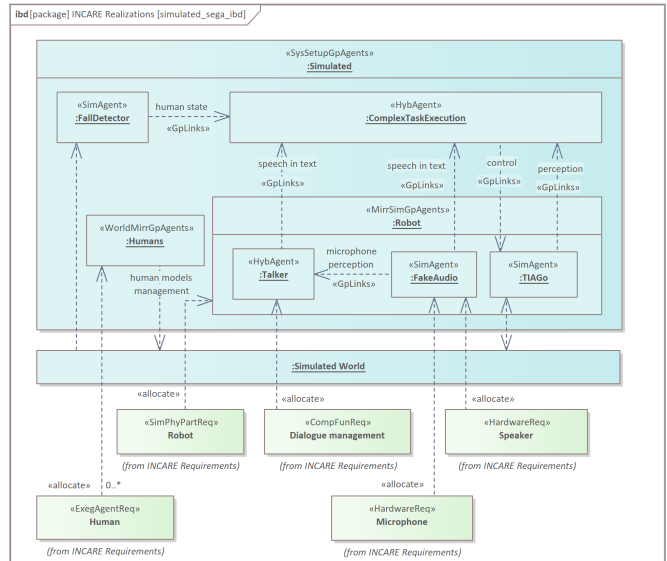
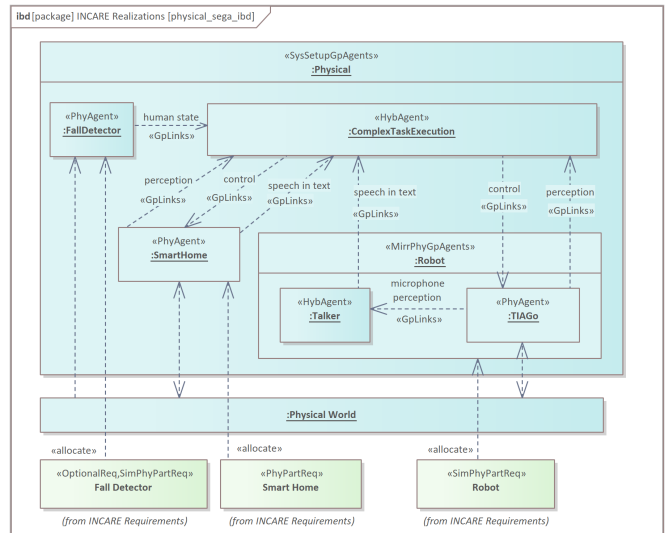
<sup>4</sup>This «*VirtEff*» is based on *gazebo\_ros\_control* package: [https://classic.gazebo.org/tutorials?tut=ros\\_control](https://classic.gazebo.org/tutorials?tut=ros_control)

(a) IBD of *TIAGo* «*SimAgent*»(b) IBD of *TIAGo* «*PhyAgent*»Fig. 15: IBD of mirroring Agents executing *Robot* «*SimPhyPartReq*» in the Physical and Simulated embodiments

«*PhyContSubsys*» was extracted. The common part constitutes *FallDetector* «*SimPhyContSubsys*» in the final design. The *FallDetector* «*SimPhyContSubsys*» is a universal interface between *ComplexTaskExecution* «*HybAgent*» and the driver layer of *FallDetector* «*PhyAgent*». It forced decomposition of *FallDetector* «*SimAgent*» to *FallDetector* «*SimPhyContSubsys*», *FallDetector* «*SimVirtRec*» and *FallDetector* «*SimRealRec*». The latter two simulate fall detector sensing. Thanks to the structure optimisation, *FallDetector* «*SimPhyContSubsys*» is used in both embodiments in the final structure and the translation of the Fall Detector message to the message useful for *ComplexTaskExecution* «*HybAgent*» can be tested in simulation.

**Step 8:** Based on the «*OptionalReq*»s in the requirements we define 12 «*SysSetupGpAgents*»s for (*SmartHome* «*PhyAgent*» existing or not, *FallDetector* «*PhyAgent*» or *FallDetector* «*SimAgent*» or no *FallDetector*, *Robot* «*PhyAgent*» or *Robot* «*SimAgent*»). We show *Simulated* «*SysSetupGpAgents*» and *Physical* «*SysSetupGpAgents*» Internal Block Diagrams (IBDs) (Fig. 16) to exemplify «*SysSetupGpAgents*» specification.

**Step 9 and 10:** In INCARE the *Robot* «*PhyAgent*» and *Robot* «*SimAgent*» are realised with TIAGo robot and its simulation. PAL Robotics mostly implemented these.

(a) *Simulated* «*SysSetupGpAgents*» with example requirement allocations(b) *Physical* «*SysSetupGpAgents*» with example requirement allocationsFig. 16: IBDs for example «*SysSetupGpAgents*»

We mapped the robot hardware and software to our design (as shown in Fig. 15). *RosControl* «*SimDriverGpSubsys*» is a software package<sup>5</sup> implementing various common ROS controllers, and *SimTIAGoDrives* «*SimHardGpSubsys*» is a package<sup>6</sup> implementing Gazebo plugins controlling TIAGo drives. Additionally, we boosted the robot's voice communication ability using the Google Dialogflow service. We equipped *TIAGo* «*PhyAgent*» with additional USB Microphones [45] to improve its audio perception. The other «*Agent*»s were implemented and tested following the simulation implementation factor. In the result, we got *Simulated* «*SysSetupGpAgents*» implemented and tested before *Physical* «*SysSetupGpAgents*», what improved robot system development safety. The world

<sup>5</sup>[http://wiki.ros.org/ros\\_control](http://wiki.ros.org/ros_control)<sup>6</sup>[http://wiki.ros.org/palHardware\\_gazebo](http://wiki.ros.org/palHardware_gazebo)



Fig. 17: A frame from *WorldSync* «*WorldMirrGpAgents*» testing.

synchronisation feature (mirroring humans in the Simulated World) was implemented with HuBeRo framework as *WorldSync* «*WorldMirrGpAgents*» and tested in a crowded hospital environment (Fig. 17).

The INCARE system was deployed in an end-user home, and the videos present its performance in the example tasks—transportation (Fig. 3a)<sup>7</sup> and fall assistance (Fig. 3b)<sup>8</sup>.

## 7 SUMMARY

Numerous Cyber-Physical Systems include simulation parts either as Digital Twins (DT), demonstrators, or mockups utilised during their development. We postulate the Simulation-Physical System (SPSys) concept to describe this kind of system. SPSys includes physical, simulated and hybrid parts. They cooperate to fulfil the system’s aim; however, some are used as DTs to boost the system’s reliability and analysis possibility. Some parts of the system can be used in the development process only as mockups and prototype simulators to increase the simulation-based testing coverage of the system. This, in turn, improves prototype safety and the resulting system robustness.

SPSys application is wide; in particular, it can work in a dynamic environment so that it can observe exogenous actions of the environment’s inhabitants. Such a situation is problematic because DT must perfectly mirror its PT. To answer the above needs, we propose a Domain Specification Language named Simulation-Physical System Modeling Language (SPSysML) that models the SPSys artefact and demonstrates SPSys taxonomy and the relationships between the types of SPSys parts.

One of the crucial aspects of reliable software development is integrity and reusability. Therefore, we propose SPSys Development Procedure (SPSysDP) that, using design evaluation factors, supports quantitative analysis and optimisation targeted to software re-use maximisation between DT and PT and in different system setups. We analyze the evaluation factors and show features of the systems that score corner values of the factors.

Finally, we verify SPSysML and SPSysDP in complex robot system development. We demonstrate step-by-step SPSysDP execution. We point out significant system structure changes resulting from the design evaluation and

the proposed quantitative factor-based guidelines applications. An example is *FallDetector* «*SimAgent*» and *FallDetector* «*PhyAgent*» integration improvement by common part isolation as «*SimPhyContSubsys*». Moreover, the verification shows that 33% of the system’s Physical Agents do not have DTs, and thanks to the design modification, the whole hardware (physical and simulated) is controlled with «*SimPhyContSubsys*» ( $R=1$ ). This has two main advantages. First, all pairs of Digital and Physical Twins share a common control subsystem. Second, if there is a «*PhyAgent*» or «*SimAgent*» without a twin, it can be easily integrated with its Digital or Physical Twin in the future.

The conducted verification shows that the requirement profile for SPSys enables critical analysis and may result in requirement decomposition and explication. An example is the decomposition of the *Communication with humans* «*SimPhyFunReq*» to *TTS* and *STT* «*SimPhyFunReq*» and *Dialogue management* «*CompFunReq*» in the first and second steps of SPSysDP. Thanks to the *Dialogue management* «*CompFunReq*» isolation, the proposed requirement-based procedure for the system decomposition resulted in the implementation of *Talker* «*HybAgent*» managing the requirement. As a consequence of the above, the requirement is managed by a component usable in different setups in cooperation with either the simulated or physical embodiment. Otherwise, the requirement would be managed separately in simulation and physical embodiments of the robot. The presented verification confirms the features distinguishing our method from the others listed in Table 1:

- Integrity evaluation with quantitative factors,
- SysML-based structure model and system development procedure,
- System design optimisation considering simulation and physical embodiments of the system,
- Multiple system execution and testing setups specification,
- Forcing simulation-based testing prior physical embodiment development using simulation implementation factor for sequencing «*SysSetupGpAgents*» implementation and testing,
- Specification of Simulation-Physical System including multiple physical and simulated parts and multiple Digital Twins.
- «*WorldMirrGpAgents*» artefact definition and including its design in SPSysDP to reflect the dynamic environment in simulation, enabling Digital Twins to observe exogenous actions.

In the future, we plan to automate testing SPSysML structure and code generation based on it with known, universal tools like Matlab and Enterprise Architect. Furthermore, this work revealed the need for a taxonomy of CPS structure optimisation indicators. SPSysML and future work leads to optimal structure development automation for complex robot systems.

## ACKNOWLEDGMENT

The research was funded by the Centre for Priority Research Area Artificial Intelligence and Robotics of War-

<sup>7</sup><https://vimeo.com/670252925>

<sup>8</sup><https://vimeo.com/670246589>

saw University of Technology within the Excellence Initiative: Research University (IDUB) programme. The authors also acknowledge TALBOT, from the European Union's Horizon 2020 research and innovation programme under Marie Skłodowska-Curie grant agreement No. 801342 (Tecniospring INDUSTRY) and the Government of Catalonia's Agency for Business Competitiveness (ACCIÓ); and SHAPES, from the European Horizon 2020 research and innovation programme under grant agreement No 857159. The Spanish grant PID2021-125535NB-I00 has also supported the work.

## REFERENCES

- [1] A. R. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 220, no. 3, pp. 183–199, 2006. [Online]. Available: <https://doi.org/10.1243/09596518JSC207>
- [2] S. Demers, P. Gopalakrishnan, and L. Kant, "A generic solution to software-in-the-loop," in *MILCOM 2007 - IEEE Military Communications Conference*, 2007, pp. 1–6.
- [3] J. Millitzer, D. Mayer, C. Henke, T. Jersch, C. Tamm, J. Michael, and C. Ranisch, "Recent developments in hardware-in-the-loop testing," in *Model Validation and Uncertainty Quantification, Volume 3*, R. Barthelemy, Ed. Cham: Springer International Publishing, 2019, pp. 65–73.
- [4] *Rapid Control Prototyping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 295–318. [Online]. Available: [https://doi.org/10.1007/3-540-29525-9\\_7](https://doi.org/10.1007/3-540-29525-9_7)
- [5] G. Lumer-Klabbers, J. O. Hausted, J. L. Kvistgaard, H. D. Macedo, M. Frasher, and P. G. Larsen, "Towards a digital twin framework for autonomous robots," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1254–1259.
- [6] C. Pylianidis, S. Osinga, and I. N. Athanasiadis, "Introducing digital twins to agriculture," *Computers and Electronics in Agriculture*, vol. 184, p. 105942, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0168169920331471>
- [7] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2351978917304067>
- [8] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, and A. Wortmann, "A cross-domain systematic mapping study on software engineering for digital twins," *Journal of Systems and Software*, p. 111361, 2022.
- [9] C. Lo, C. Chen, and R. Y. Zhong, "A review of digital twin in product design and development," *Advanced Engineering Informatics*, vol. 48, p. 101297, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1474034621000513>
- [10] B. Bauml and G. Hirzinger, "Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 3741–3748.
- [11] A. Stocco, B. Pulfer, and P. Tonella, "Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems," *IEEE Transactions on Software Engineering*, pp. 1–13, 2022.
- [12] A. Bihlmaier and H. Wörn, "Robot unit testing," in *Simulation, Modeling, and Programming for Autonomous Robots*, D. Brugali, J. F. Broenink, T. Kroeger, and B. A. MacDonald, Eds. Cham: Springer International Publishing, 2014, p. 255–266.
- [13] A. M. Tiryaki, S. Öztuna, O. Dikenelli, and R. C. Erdur, "Sunit: A unit testing framework for test driven development of multi-agent systems," in *Agent-Oriented Software Engineering VII*, L. Padgham and F. Zambonelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 156–173.
- [14] M. Mossige, A. Gotlieb, and H. Meling, "Testing robot controllers using constraint programming and continuous integration," *Information and Software Technology*, vol. 57, pp. 169–185, 2015.
- [15] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2015.
- [16] C. Gehrmann and M. Gunnarsson, "A digital twin based industrial automation and control system security architecture," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 669–680, 2020.
- [17] M. El-Shamouty, K. Kleeberger, A. Lämmle, and M. Huber, "Simulation-driven machine learning for robotics and automation," *tm - Technisches Messen*, vol. 86, no. 11, pp. 673–684, 2019. [Online]. Available: <https://doi.org/10.1515/teme-2019-0072>
- [18] S. Wolny, A. Mazak, C. Carpella, V. Geist, and M. Wimmer, "Thirteen years of sysml: a systematic mapping study," *Software and Systems Modeling*, vol. 19, pp. 111–169, 2020.
- [19] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, p. 101021, 2021.
- [20] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliński, "EARL – Embodied Agent-Based Robot Control Systems Modelling Language," *Electronics*, vol. 9, no. 2 - 379, 2020.
- [21] T. Glock, B. Sillman, M. Kobold, S. Rebmann, and E. Sax, "Model-based validation and testing of industry 4.0 plants," in *2018 Annual IEEE International Systems Conference (SysCon)*, 2018, pp. 1–8.
- [22] G. Fortino and W. Russo, "Eldameth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems," *Information and Software Technology*, vol. 54, no. 6, pp. 608–624, 2012, special Section: Engineering Complex Software Systems through Multi-Agent Systems and Simulation. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584911001960>
- [23] J. L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R. K. Panesar-Walawege, Ángel López, I. del Río, and T. Kelly, "Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel," *Information and Software Technology*, vol. 72, pp. 16–30, 2016.
- [24] B. Miller, F. VahicK, and T. Givargis, "Application-specific code-sign platform generation for digital mockups in cyber-physical systems," in *2011 Electronic System Level Synthesis Conference (ES-Lsyn)*, 2011, pp. 1–6.
- [25] K. M. Alam and A. El Saddik, "C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.
- [26] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.
- [27] M. Moallemi and G. Wainer, "Modeling and simulation-driven development of embedded real-time systems," *Simulation Modelling Practice and Theory*, vol. 38, pp. 115–131, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1569190X13001196>
- [28] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of modeling and simulation*. Academic press, 2000.
- [29] W. Dudek, "Prudent management of interruptible tasks executed by a service robot," Ph.D. dissertation, Warsaw University of Technology, 2021, access: [https://robotyka.ia.pw.edu.pl/papers/phd\\_thesis\\_wd.pdf](https://robotyka.ia.pw.edu.pl/papers/phd_thesis_wd.pdf).
- [30] T. Winiarski, S. Jarocki, and D. Seredyński, "Grasped object weight compensation in reference to impedance controlled robots," *Energies*, vol. 14, no. 20, 2021. [Online]. Available: <https://www.mdpi.com/1996-1073/14/20/6693>
- [31] 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, 2022. [Online]. Available: <https://www.mdpi.com/1996-1073/15/21/7983>
- [32] J. Pages, L. Marchionni, and F. Ferro, "TIAGo: the modular robot that adapts to different research needs."
- [33] 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*, vol. 3, no. 3.2, 2009.
- [34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>

- [35] H. Bruyninckx, "Open robot control software: The OROCOS project," in *International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2001, pp. 2523–2528.
- [36] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.
- [37] "Documentation of gazebo\_ros\_control package," [https://classic.gazebosim.org/tutorials?tut=ros\\_control&cat=connect\\_ros](https://classic.gazebosim.org/tutorials?tut=ros_control&cat=connect_ros), accessed: 23-05-2022.
- [38] M. dos Santos Soares and J. L. Vrancken, "Model-driven user requirements specification using sysml." *J. Softw.*, vol. 3, no. 6, pp. 57–68, 2008.
- [39] "EARL 1.3 documentation," <https://www.robotyka.ia.pw.edu.pl/projects/earl>, accessed: 2023-03-06.
- [40] J. Karwowski, W. Dudek, M. Węgierek, and T. Winiarski, "Hubero - a framework to simulate human behaviour in robot research," *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 15, no. 1, p. 31–38, Jul. 2021. [Online]. Available: <https://www.jamris.org/index.php/JAMRIS/article/view/664>
- [41] T. Winiarski, "MeROS: Metamodel for ROS-based Systems," 2023.
- [42] N. S. Brenčič, M. Dragoi, I. Mocanu, and T. Winiarski, *Intuitive and Intelligent Solutions for Elderly Care*. Cham: Springer International Publishing, 2020, pp. 101–108.
- [43] W. Dudek, M. Węgierek, J. Karwowski, W. Szykiewicz, and T. Winiarski, "Task harmonisation for a single-task robot controller," in *12th International Workshop on Robot Motion and Control (RoMoCo)*, K. Kozłowski, Ed. IEEE, 2019, pp. 86–91.
- [44] W. Dudek and T. Winiarski, "Scheduling of a Robot's Tasks With the TaskER Framework," *IEEE Access*, vol. 8, pp. 161 449–161 471, 2020.
- [45] T. Winiarski, W. Dudek, M. Stefanczyk, L. Zielinski, D. Gioldowski, and D. Seredynski, "An intent-based approach for creating assistive robots' control systems," *CoRR*, vol. abs/2005.12106, 2020. [Online]. Available: <https://arxiv.org/abs/2005.12106>