
SESAME
SECURE AND SAFE MULTI-ROBOT SYSTEMS

**Project Number 101017258**

# D3.1 MRS Design Concept and Methodology

**Version 1.0**
**30 September 2021**
**Final**

**Public Distribution**

**Hochschule Bonn-Rhein-Sieg**

**Project Partners:** **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

# Project Partner Contact Information


| | |
|---|---|
| **Aero41**<br>Frédéric Hemmeler<br>Chemin de Mornex 3<br>1003 Lausanne<br>Switzerland<br>E-mail: frederic.hemmeler@aero41.ch | **ATB**<br>Sebastian Scholze<br>Wiener Strasse 1<br>28359 Bremen<br>Germany<br>E-mail: scholze@atb-bremen.de |
| **AVL**<br>Martin Weinzerl<br>Hans-List-Platz 1<br>8020 Graz<br>Austria<br>E-mail: martin.weinzerl@avl.com | **Bonn-Rhein-Sieg University**<br>Nico Hochgeschwender<br>Grantham-Allee 20<br>53757 Sankt Augustin<br>Germany<br>E-mail: nico.hochgeschwender@h-brs.de |
| **Cyprus Civil Defence**<br>Eftychia Stokkou<br>Cyprus Ministry of Interior<br>1453 Lefkosia<br>Cyprus<br>E-mail: estokkou@cd.moi.gov.cy | **Domaine Kox**<br>Corinne Kox<br>6 Rue des Prés<br>5561 Remich<br>Luxembourg<br>E-mail: corinne@domainekox.lu |
| **FORTH**<br>Sotiris Ioannidis<br>N Plastira Str 100<br>70013 Heraklion<br>Greece<br>E-mail: sotiris@ics.forth.gr | **Fraunhofer IESE**<br>Daniel Schneider<br>Fraunhofer-Platz 1<br>67663 Kaiserslautern<br>Germany<br>E-mail: daniel.schneider@iese.fraunhofer.de |
| **KIOS**<br>Panayiotis Kolios<br>1 Panepistimiou Avenue<br>2109 Aglatzia, Nicosia<br>Cyprus<br>E-mail: kolios.panayiotis@ucy.ac.cy | **KUKA Assembly & Test**<br>Michael Laackmann<br>Uhthoffstrasse 1<br>28757 Bremen<br>Germany<br>E-mail: michael.laackmann@kuka.com |
| **Locomotec**<br>Sebastian Blumenthal<br>Bergiusstrasse 15<br>86199 Augsburg<br>Germany<br>E-mail: blumenthal@locomotec.com | **Luxsense**<br>Gilles Rock<br>85-87 Parc d'Activités<br>8303 Luxembourg<br>Luxembourg<br>E-mail: gilles.rock@luxsense.lu |
| **The Open Group**<br>Scott Hansen<br>Rond Point Schuman 6, 5$^{th}$ Floor<br>1040 Brussels<br>Belgium<br>E-mail: s.hansen@opengroup.org | **Technology Transfer Systems**<br>Paolo Pedrazzoli<br>Via Francesco d'Ovidio, 3<br>20131 Milano<br>Italy<br>E-mail: pedrazzoli@ttsnetwork.com |
| **University of Hull**<br>Yiannis Papadopoulos<br>Cottingham Road<br>Hull HU6 7TQ<br>United Kingdom<br>E-mail: y.i.papadopoulos@hull.ac.uk | **University of Luxembourg**<br>Miguel Olivares Mendez<br>2 Avenue de l'Universite<br>4365 Esch-sur-Alzette<br>Luxembourg<br>E-mail: miguel.olivaresmendez@uni.lu |
| **University of York**<br>Simos Gerasimou & Nicholas Matragkas<br>Deramore Lane<br>York YO10 5GH<br>United Kingdom<br>E-mail: simos.gerasimou@york.ac.uk<br>nicholas.matragkas@york.ac.uk | |

# Document Control

| Version | Status | Date |
|---|---|---|
| 0.1 | Document outline | 23 July 2021 |
| 0.2 | First draft | 27 August 2021 |
| 0.6 | Version for partner reviews | 23 September 2021 |
| 0.7 | Feedback from Fraunhofer IESE resolved | 30 September 2021 |
| 0.8 | Feedback from Locomotec resolved | 30 September 2021 |
| 1.0 | Final QA version for EC submission | 30 September 2021 |

# Table of Contents

# Executive Summary

This deliverable reports about the research performed in Task 3.1 of the SESAME project. The main purpose of the deliverable is to motivate and explain to experts inside and outside the consortium the scientific foundations, concepts and principles of the proposed Executable Scenario (ExSce) methodology for engineering dependable MRS. In a nutshell, Executable Scenarios are *model-based* narrative descriptions of robotic missions which guide the engineering of MRS applications. We argue that the systematic engineering of dependable MRS calls for involving and integrating heterogeneous stakeholders with different robotics expertise throughout the life-cycle of MRS. In order to support this integration and to enable stakeholders to *specify*, *execute* and *manage* scenarios of MRS missions, the ExSce methodology is based on a model-based paradigm, which facilitates the formalization of reusable scenario concepts. Supported by a thorough empirical domain analysis of engineering practices for MRS, we identify and describe several core scenario concepts and their constraints on each other as well as their inherent variability originating from, to name a few, different stakeholder needs and robot systems. In particular, we discuss some scenario concepts involved in exploiting scenarios as a way *(i)* to express MRS requirements and associated criteria of acceptance, and *(ii)* to execute and test scenarios in simulation environments.

# 1 Scientific Foundations, Prior and Related Work

The objective of this chapter is twofold. First, to provide a brief introduction into the scientific foundations of the Executable Scenario concept and methodology which will be introduced in Chapter 2.To this end, we employ a system engineering perspective to answer the question *which* and *how* stakeholders are involved in engineering multi-robot systems (cf., Section 1.1). Second, to introduce and discuss prior and related work in the broad field of robot system and software engineering, which is crucial to understand our motivation to introduce the Executable Scenario concept and methodology. To this end, we use the stakeholders introduced in Section 1.1 to discuss how model-based approaches in robotics support stakeholders to perform their tasks. We also discuss how existing robot software frameworks and process models are tailored toward robotic needs and how stakeholders play a role in them (cf., Section 1.2).

## 1.1 Stakeholders in Robotic Applications

Engineering multi-robot systems as those considered in SESAME is an activity which is performed by various stakeholders. Generally, engineering processes often refer to *stakeholders* as the source of requirements that the developed system or software need to fulfil. Identifying parties who may hold stakes in the system-of-interest (SoI) is challenging, as relevant stakeholders can vary depending on the particular system being developed, as well as the specific stage the SoI is in over its life cycle. For consistency, we adopt the definition of stakeholders from the ISO standard on system life cycle processes [2]:

> **Stakeholder** Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations.

From an organizational perspective, partners in the SESAME project can be broadly categorized into two groups: *industrial partners* who provide use cases inspired by their commercial application, and *research and development (R&D) partners* who drive the innovations behind technologies developed in the project.

- Having to consider tangible financial risks, industrial partners share a more conservative view on innovations and are generally resistant to any major modifications of their existing processes or systems. Furthermore, they tend to focus on problems and requirements tightly coupled to their use case and not necessary applicable anywhere else. Perhaps by design, SESAME's industrial representatives cover a wide variety of both MRS scenarios (e.g., from smart manufacturing to drone inspection) and maturity (e.g., from emerging startup to international corporation), which can allow for different levels of tolerance towards more generalized and potentially disruptive innovations.
- Compared to their industrial counterparts, R&D partners are either research institutes or universities and, as a result, are less pressured by financial concerns. They can explore a wide variety of approaches in order to produce innovative solutions for more generic problems, which may also include ones found in industry. However, research-oriented groups tend to focus on very specific aspects of such problems, skimming over assumptions that may require significant engineering effort to resolve in industrial settings.

In addition to the organizational perspective, stakeholders may also be identified by the activities they perform in the project. The Horizon 2020 project RobMoSys[1] (Composable Models and Software Robotic Systems) lists several roles that may influence the development of robotics applications [98]. The definitions of task, skill, service, and other RobMoSys relevant terms used for the following description of the stakeholders can be found on the project's wiki page[2]:

[1] https://robmosys.eu/
[2] https://robmosys.eu/wiki/glossary

*Behaviour developers* develop high-level tasks and task-plots, which can be performed using the robot's available functionalities (i.e., skills), to realize the desired robotic application.

*Service designers* provide the service definitions which consist of communication semantics, data structure and other properties required to describe components' functions and their interfaces. These definitions are independent on the components' concrete realization and can be used for designing the system's architecture without having to consider concerns specific to a particular use case or platform.

*Function developers* develop functions to be used by *component suppliers* to create components.

*Component suppliers* use functions provided by *function developers* to create components based on service definitions from *service designers*.

*System architects* designs a system architecture using existing service definitions provided by *service designers*. The resulting architecture is independent on specific realization of components.

*System builders* assemble the system by selecting and composing appropriate components from the ones provided by *component suppliers* to realize the desired services specified in the architecture designed by *system architects*.

*Safety engineer* deals with safety-related concerns of the system. It is important to emphasize, that in SESAME we argue that this role should be extended to also cover security, i.e. *safety and security engineer*, as both can be critical to a MRS system's "correct operations". Requirements provided by this role may determine some of the instrumentation needs for verification and validation (V&V), which *system builders* and *component suppliers* need to take into consideration during their respective activities.

*Performance designer* should be extended to cover other non-functional requirements (NFR) [1] beyond performance, such as reliability, maintainability, and so on. Constraints specified by this role may also influence the decisions of *component suppliers* and *system builders* during component creation and selection. Actors in this role may need to balance the priorities of NFR that often conflict with each other: reliability, for example, may imply inserting extra code that can impair performance.

In SESAME, such roles may be represented by individuals from both industrial or R&D partners. For example, a R&D partner may wish to reuse existing components that need to interact with ones from an industrial partner. This may require coordination of *service designers* and *system architects* from both organizations to agree up on a final system architecture. Furthermore, and especially for teams of smaller size, a single individual may assume responsibilities for several roles. A *system architect*, for example, often also selects the appropriate components to realize their architecture, i.e. performs a *system builder*'s activity. Assuming responsibilities for multiple roles is particularly common in smaller organizations, evident as shown and discussed in Section 2.3.

Other relevant activities may encompass several of the aforementioned roles. Testing, for example, can be done at the functional, component or system level. At the same time, tests can be used to verify performance and safety/security requirements or validate expected behaviours. Testing as an activity, therefore, can be performed by every role listed above. Another example is hardware design and/or selection, which in SESAME is likely to involve only *component suppliers* and *system builders*. Other projects, however, maybe interested in developing hardware components from scratch, which then would demand activities from the *function developer* role.

The RobMoSys roles cover *intrinsic* views on designing and building systems. External factors like customers or regulatory bodies may also influence this process. For example, each use case described and developed within SESAME needs to conform to general and domain-specific standards imposed by authorities before it can be deployed in the field.

## 1.2 Robot System and Software Engineering

### 1.2.1 Robot Software Frameworks

Robot software frameworks provide developers with a set of tools, libraries, and communication mechanisms for building, programming, and controlling robots. Most notable examples are ROS (Robot Operating System) [91], Orocos (Open Robot Control Software) [21][103] and YARP (Yet Another Robot Platform) [40]. ROS is a widely used robot software framework and provides a modular architecture and libraries for common robot tasks. With the advent of ROS 2 several peculiarities regarding communication and node management of ROS 1 are mitigated. Orocos follows as ROS a component-based paradigm, and enables the development of reusable software components that can be combined to create complex robotic systems. YARP is another flexible and modular robot software framework.

### 1.2.2 Model-Driven Engineering in Robotics

The Executable Scenario concept and methodology is based on the model-driven engineering paradigm [27] which promotes the idea to consolidate, harmonise and capture domain knowledge both conceptually and technically in the form of meta-models. Those meta-models, to mention a few benefits, pave the way to create domain models with the help of textual and graphical domain-specific languages (DSLs) which are programming or specification languages having abstractions from a particular problem domain [111]. Exploiting domain models as core artefacts throughout the complete development cycle ranging from design, specification, implementation to testing and even deployment is nowadays well-established in several application fields such as avionics, automotive and telecommunications where general-purpose modelling languages and their domain-specific extensions (e.g., UML MARTE [8], SysML [85], AADL [38]) are used to model and analyse various system aspects and properties. In the control engineering field, modelling approaches and associated tools such as Matlab/Simulink [72], 20-sim[3], LabView[4], OpenModelica[5], and Dymola[6] are popular to rapidly develop, for example, control policies and to automatically generate optimised source code for various embedded devices and platforms. In [17] and [79] these modelling tools are used together with software architectural modelling tools to construct robot control architectures.

In robotics, DSLs are also not a new concept and early work dates back to 1984 where Henderson *et al.* introduced the Logical Sensor Specification Language (LSS) [52] to support robot engineers to specify both architectural and algorithmic concerns of sensing and state estimation processing chains. Nowadays, DSLs are frequently used in robotics to specify knowledge from different problem domains such as kinematics [43][55], dynamics [42], motion control [70][105] and perception [54][16]. Generally speaking, for any model-based approach to be successful it is crucial to understand when, how and by whom the DSL and associated infrastructure is used and for which kind of task [111] and Schneider *et al.* [100] investigated the origin of design decisions and underlying development process that resulted in a robotics DSL for specifying robot grasping problems. In the context of the SESAME project we investigate in the following paragraphs the rich Body of Knowledge in robotic DSLs from the perspective of the RobMoSys stakeholders introduced in Sec. 1.1. To this end, we classify existing MDE approaches in robotics along stakeholders and their activities.

*Behaviour developers* can rely on several DSLs to specify and analyse coordination activities both on a programming language level [13][14][97][28] and on an architectural level [60][66][108][109][95], where different variants of hierarchical state machine languages are used to specify behaviours. Notable examples are formalisms based on behaviour trees [45] and approaches inspired by behaviour-based

[3] https://www.20sim.com/ Last accessed: 14th of March 2023
[4] https://www.ni.com/labview Last accessed: 14th of March 2023
[5] https://openmodelica.org/ Last accessed: 14th of March 2023
[6] http://www.3ds.com/products-services/catia/products/dymola Last accessed: 14th of March 2023

robotics [110]. For orchestrating several robots existing DSLs and associated frameworks enable behaviour developers to specify MRS tasks and missions [67][73] as well as managing execution-related aspects such as resources [68][81]. Most of the mission specification languages include domain-specific aspects, for example, the work of [24] allows the definition of permitted flying zones as part of the drone mission specification.

*Service designers* are concerned with specifying re-usable interfaces and data structures of robot capabilities and components. The general idea of DSLs supporting this role is to provide abstractions to model both the problem space and solution space [92][44][93] of robotic systems in the form of architectural patterns [9][99], behaviour templates [105] or specifications of high-level robot capabilities which are independent of concrete realizations (e.g., feature model approaches [48][18]).

*Function developers* are supported by a wide range of DSLs to specify different robot functions ranging from kinematic and dynamic algorithms [42], motion control architectures [83][84] and force control algorithms [59] to perception [52][53]. The majority of these DSLs also support *function developers* in tasks which are in one way or another either associated to *service designers* or *system architects*. For example, in the work of Klotzbücher *et al.* also enables function developers to create re-usable templates of different types of hybrid force/velocity motions and the motion control languages introduced by Nordmann [84] enables developers also to model architectural concerns of motion control systems.

*Component suppliers* package developed functions into components and existing DSLs support this task for dedicated type of components (e.g., supporting packaging of sensor components [10]) or more generally by providing dedicated views in model-based tools (e.g., see [86], [33]). Another approach is proposed by Mallet *et al.* where functions are integrated directly in components conforming to a particular robot software framework (e.g., [69]).

*System architects* are supported by various DSLs to assemble components into an overall architecture. DSLs range from supporting the specification of domain-specific architectures and sub-systems (e.g., motion control [83] or perception [54]) and general-purpose approaches [9] to languages for modelling architectures of decisional autonomy [35] and the analysis of execution (e.g., scheduling, WCET etc.) properties of robot control architectures [64].

*System builders* are similarly supported with DSLs as system architects, and the majority of existing approaches do not make a concrete distinction of these two roles. However, some approaches provide dedicated views in their model-based tools to support the system builder [33].

*Performance designer* are supported by DSLs to specify non-functional requirements and properties on various levels of abstraction ranging from components and sub-systems [56][80][94] to quality-of-service properties of robot software middlewares [87]

*Safety engineers* need to specify and verify safety related requirements and existing approaches employ either domain-specific abstractions to model safety-related aspects (e.g., [4]) or employ well-established formal languages and approaches such as LTL [39] and other formalisms [3][31][30] to specify requirements and to synthesize systems and safe robot behaviours.

### 1.2.3 Development Processes in Robotics

While there are several reports about successful deployments of robotic systems in the real-world (e.g., deployment of robots in long-term domestic environments [50] or healthcare robots [57]), little is known about the underlying and employed development processes used to engineer these and other applications. This is partly due to the fact that in the robotics community it is more accepted and common to report about relevant and successful core robotic algorithms (e.g., perception, navigation, planning and control) than to report about actual development practices. A notable exception is the work of Garcia *et al.*[46] where the authors assessed in an empirical manner the impact of variability challenges to robotics software engineering. Another example is

the BRICS project, where researchers assessed different robot application development processes and derived a holistic process model, namely the BRICS Robot Application Process. The BRICS RAP [51] is a holistic process model and combines several ideas and process elements known from traditional software engineering (e.g., unit testing, agile development, V-Model etc.). In essence, the BRICS RAP is a multi-stage, iterative and tailorable development process model composed of, to name a few, activities related to *scenario building*, *platform building capability building* and *maintenance* supporting different stakeholders. Even though, the BRICS RAP paved the way to systematically work out challenges and desirables of robotics software engineering, it remains a conceptual model. While some activities such as *capability building* are in the meanwhile well-supported with tools and modelling approaches (see Section 1.2.2) some activities are not supported at all. In particular, activities related to scenario building, scenario verification and so forth are crucial, yet overlooked activities of robotics software engineering which greatly motivates the proposed ExSce concept and methodology described in this deliverable.

# 2 Concept and Methodology of the Executable Scenario

In this chapter we provide a definition of the main terms associated with the ExSce, namely the ExSce Concept, the ExSce Methodology and the ExSce Workbench. Then we exemplify those definitions and derive models as well as tools that can help SESAME stakeholders to exploit the ExSce.

**Definition: ExSce Concept**

Executable Scenarios (ExSce) are model-based narrative descriptions of robotic missions guiding the engineering of MRS applications. An ExSce supports the definition of scenarios, composed of *mission-relevant* and *mission-plausible* information. On the one hand, by mission-relevant information we refer to, among others, the environment and its dynamics, time and events, objects (e.g., inspected building) and subjects (e.g., human operators) and their potential behaviour. On the other hand, the mission-plausible information describes acceptance criteria that enable the verification and validation of MRS requirements. Both, mission-plausible and mission-relevant information are computer-interpretable models and hence enable the transformation into artefacts which can be executed in different contexts such as in simulators or on real robots.

**Definition: ExSce Methodology**

To put the ExSce concept into action, the ExSce methodology enables the structured development of MRS from stakeholders' scenarios. Hence, it comprises both a tailorable *process model* and associated *tools* that guide and support the stakeholders in *(i)* specifying scenarios; *(ii)* transforming them to executable artefacts; *(iii)* executing those artefacts; *(iv)* assuring the quality of the MRS in those scenarios; and *(v)* generalizing those scenarios.

**Definition: ExSce Workbench**

The ExSce Workbench is a collection of tools that supports stakeholders in carrying out one or more activities of the ExSce Methodology.

## 2.1 Exemplifying the ExSce Concept

To represent the narrative of the ExSce Concept, we follow the established practice of user stories [25] as commonly found in computer science. A user story facilitates the communication with end-users or stakeholders of a software system. They are a natural-language specification that is easy to understand by a particular stakeholder and allows them and developers to agree on software requirements. In the ExSce Concept, we extend that definition to the whole MRS, including hardware and software, as exemplified in Figure 1 that shows a potential, yet simple, user story of a mobile robot navigating in a hospital.

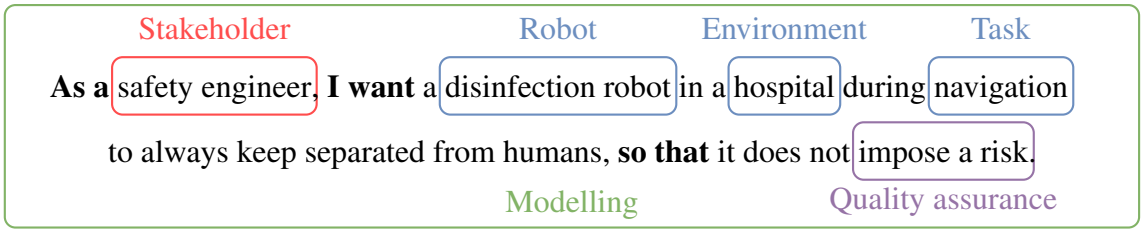



Figure 1: Example user story for a disinfection robot

We follow the common pattern of user stories [104] to structure the narrative via "As a ... I want to ..., so that ...". Highlighted by the red box, we see the stakeholder or role from whose perspective this user story is written. Each stakeholder introduces a different view on the scenario and its constituents, as we have already

discussed in Section 1.1. The following three blue boxes represent the mission-relevant information from the ExSce Concept, i.e. the robot itself, the environment in which the robot operates and the task that the robot performs. The purple box shows the mission-plausible information, i.e. the acceptance criteria that are an integral part of a user story that facilitate the quality assurance of a robotic system.

The user story already follows a semiformal structure, as indicated by the terms in bold font that represent the keywords of the user story language. Yet, the highlighted terms remain in natural language and hence inaccessible to computer-based interpretation. The objective of ExSce Concept is to provide means to formalize those terms and capture their semantics as well as their links to the user story language via explicit models as indicated via the green box.

While being a valuable approach in requirements engineering, a user story is still too abstract to execute because it lacks more detailed information about the narrative or concretely measurable quantities. To this end, computer scientist have introduced scenario specifications to express software behaviour (e.g., intended reaction to events) in a natural way and to exploit those specifications for the actual implementation. Well-known representatives of this approach are Use Case diagrams of the Unified Modelling Language (UML) or more formal approaches such as Live Sequence Charts [29] and others [65]. However, for the ExSce Concept, we employ Behaviour-Driven Development (BDD) that builds on the success stories of test-driven development (TDD) [12]. While TDD is a methodology for testing implementations, BDD supports acceptance testing. To this end, it employs scenarios to bridge the gap between software customers and software developers. The success of BDD can also be tied to the availability of the (informal) Gherkin language[7] and the Cucumber software tooling [112] that supports developers in interpreting Gherkin user stories. Gherkin extends user stories by the description of a narrative as a collection of scenarios. Each scenario description consists of a `given-when-then` cascade that define the initial situation, an event or action and an acceptance criterion, respectively. Figure 2 exemplifies such a scenario for the previous user story.

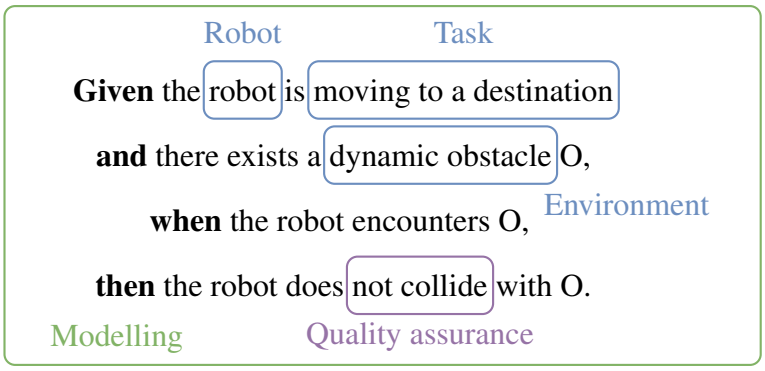



Figure 2: Gherkin-like scenario specification to extend the user story from Figure 1.

## 2.2 Exemplifying the ExSce Methodology

As a next step, we focus on the ExSce Methodology that exploits the scenario models that we have discussed above. To this end, we present the methodology's process and tools that support the activities in that process. Figure 3 shows a graphical representation of the ExSce process, which is composed of the five activities. We visit those activities in the context of the example scenario in Figure 2 and, to keep it concrete for a wide robotics audience, assume here a ROS-based robot [62].

Each activity can be supported by tools to various degrees. Most robotic developers follow this process in one way or another, as we will also discuss in Section 4. However, they tend to skip a clear requirements specification and transition to code quickly. As a result, many software solutions become *implicitly* tailored to specific robots, environments or tasks and, hence, difficult to reuse in a different context.

[7] https://cucumber.io/docs/gherkin/

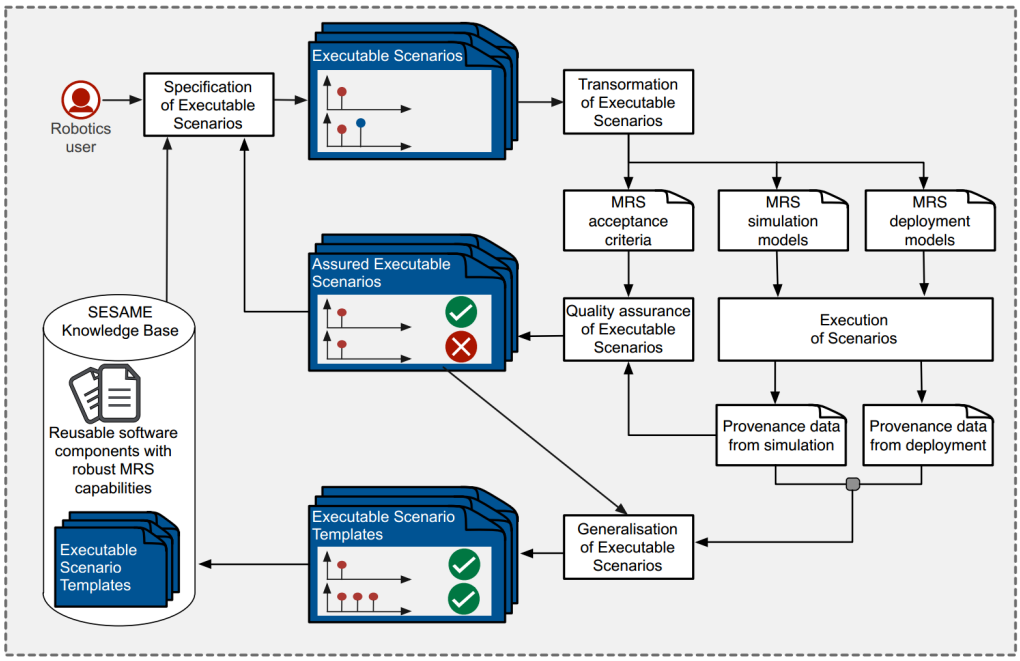



Figure 3: Process view of the ExSce Methodology

**Specification of Executable Scenarios** This activity addresses the model building for the scenario. Here, function developers would, for example, provide URDF [23] files for specific kinematic chains, component suppliers would contribute, for instance, launch files[8] or initial configuration files for ROS nodes, whereas system builders create launch files for an overall robotic system. Those models are part of the *robot* in the scenario specification. Performance engineers or safety engineers would usually reuse such existing models.

The stakeholder also must provide the robot with the *task* to be performed. Often the task specification remains hidden in code. Yet, in various domains some formal representations have emerged such as the Planning Domain Definition Language (PDDL) [47] in task planning (example in Listing 2), the Task Frame Formalism (TFF) [71, 20] in motion control or behaviour trees[9] in navigation stacks. In the example of Figure 2, the task could be as simple as a navigation goal that is eventually sent to the robot's navigation stack during the execution, as shown in the more extensive Listing 1 (cf. `navigation` type).

```
- type: "navigation"
  goal: [3.82, -2.5, -2.14]
- type: "lamp_control"
  switch_on: True
- type: "standstill"
  duration: 10.0
- type: "wall_following"
  checkpoints:
  - [ 3.16, -3.34, -0.56]
  - ...
  polygon:
  - [4.11, -0.67]
  - ...
```

Listing 1: Specification of a disinfection task [88].

```
(define (problem disinfect-hospital)
  (:domain disinfection-domain)
  (:objects robot-1 - Robot
            room-1 room-2 - Room)
  (:init (robot-location robot-1 room-1)
         (connected room-1 room-2)
         (infected room-2))
  (:goal (disinfected room-2)))
```

Listing 2: Simple mission specification in PDDL.

[8] http://wiki.ros.org/roslaunch/XML
[9] https://navigation.ros.org/behavior_trees/index.html

```
...
events:
  - time: 0.65
    variable: intIn
    value: 2
  - time: 0.8
    variable: intOut
    value: 5
```

Listing 3: Example scenario in libcosim.

```
wiggle = (-10 deg, 10 deg)
ego = Car with roadDeviation wiggle
c = Car visible, \
        with roadDeviation resample(wiggle)
leftRight = Uniform(1.0, -1.0) * (1.25, 2.75)
Car beyond c by leftRight @ (4, 10), \
    with roadDeviation resample(wiggle)
```

Listing 4: Example scenario in Scenic.

Additionally, the stakeholder must define the *environment* in which the robot should execute the task. This includes, for instance, obstacles, their shape, appearance or surface materials but also agents in that environment or "events" occurring in the environment. Listing 3 exemplifies the latter of a libcosim scenario[10] where at defined points in time certain values are injected into a simulated system. On the contrary, Listing 4 is an example that contains the description of an active agent $c$ in the Scenic [41] specification. However, the example specification in Figure 2 only defines the presence of a "dynamic obstacle" which could simply be realized in an ad-hoc manner by a person randomly walking in front of the real robot or in a more controlled manner, for example, by another robot that crosses in front of the robot under test. The degree of formality and control tends to depend on the strictness of regulations.

Finally, the stakeholder specifies the mission-plausible information or acceptance criteria, here concretely, that the robot should *not collide* with an obstacle. At this point, the specification may or may not link to a concrete way of validating that criterion. For example, an easy "implementation" could be realized by the safety engineer observing the robot and just noting down what they consider a collision. Instead, a more objective analysis could also be based on the robot's internal data or derived from external measurements such as a motion capturing system.

To support the stakeholders in this activity, we foresee various candidates for models and tools. First, handling the BDD specification as such, as we will discuss in Section 3. Second, another set of candidate models comprises the task definition (e.g. a more formal representation of "moving to a destination") and the environment definition (e.g. how and when should the dynamic obstacle move).

**Transformation of Executable Scenarios** Here, the scenarios are transformed to bring them into a shape that is ready for execution either in simulation or on a real robot. For many artefacts, especially when they are only described in natural language, a manual implementation is required and a human must make sure that this implementation conforms to the requirements. However, as also discussed in Section 1 there already exist a wide range of languages and related tools to automatically transform some of the robot-centered models into code, for example, to derive a controller implementation from Matlab/Simulink models or to turn a declarative PDDL mission specification into concrete plan via automated planning.

Additionally, in this activity, the acceptance criteria need to be extracted and potentially refined from the ExSce specifications. In the running example, the choice on how to evaluate the "no collide" acceptance criterion must be made at the latest at this stage. Similarly, the dynamics of the environment must be grounded in code or configuration, for instance, by implementing a Gazebo plugin that lets an agent follow a predefined trajectory[11] or randomly move[12] in the environment.

The more commonalities there are between concepts, the more beneficial formal models and automated transformations are. For instance, both, the robot's task and the dynamic obstacle's behaviour could be realized by a trajectory or waypoint following controller and hence could share the same model representation and tools to process those models. Further, candidates for tools comprise various code, configuration or deployment script generators for various framework-related artefacts including monitors to evaluate acceptance criteria. Finally,

[10] https://open-simulation-platform.github.io/libcosim/scenario
[11] https://classic.gazebosim.org/tutorials?tut=animated_box&cat=build_robot
[12] https://classic.gazebosim.org/tutorials?tut=plugins_random_velocity

the formal representation of all those models provides the opportunity to systematically and automatically introduce controlled variations into (parts of) the models.

**Execution of Scenarios** In this activity, the stakeholder deploys all required software on the robotic system, initializes the system as required and tries to solve the task. Tool-wise, the execution environments are already well covered by the various established Robot Software Frameworks such as ROS or OROCOS. We review the simulation side in more detail in Section 5. To obtain data to be used in the following activities, the robot and the execution environment must be instrumented accordingly. In the ROS context, the former can be done to a certain extent using rosbag[13] *if* the appropriate data is exposed via topics. Otherwise, additional implementation and tooling effort is required. For the latter, i.e. the instrumentation of the execution environment, this could mean to interface with a simulator or, in the real world, employing external measurement devices such as motion capturing systems. In any case, for traceability the recorded data should be connected to *(i)* the artefacts produced by the scenario specification and transformation activities; as well as *(ii)* the run-time system (for example the ROS node from which the data originated) via provenance meta-data [78].

**Quality assurance of Executable Scenarios** *Quality assurance of Executable Scenarios* consumes the execution traces and provenance data. By applying the acceptance criteria to the data contained in those artefacts, the execution can be evaluated and associated with scenarios. Section 4.1 demonstrates *one* possible realization of the quality assurance activity that is employed in the hospital disinfection use case.

**Generalization of Executable Scenarios** The final activity, *Generalization of Executable Scenarios* distils the outcomes of the previous phases to narrow down test cases, open up tolerances, add or remove conditions to scenario specifications which results in generalized scenario templates. Given the insights from the execution traces, the respective stakeholder can also contribute new scenarios.

In the running example of Figure 2, the safety engineer may find out that the initial scenario was incomplete. For instance, *(i)* during the execution the robot was colliding with other obstacles such as walls that were not explicitly listed; *(ii)* the robot was not colliding with the obstacle but approaching it too closely so that it would result in uncomfortable encounters with humans; or *(iii)* the environment was not representative of the desired deployment site due to a lack of constraints in terms of rooms or corridors.

From a tooling perspective, stakeholders could be supported by comparing multiple ExSce specifications to identify commonalities or differences between such specifications. Such an analysis could then guide them in generating ExSce templates to be reused across different robots or environments. Such templates could also be valuable inputs for design space exploration tools that could, for instance, sample open parameters to test a system for robustness.

## 2.3 Instantiating the ExSce for further Use Cases and their Stakeholders

Every stakeholder has a different point of view of the system, hence they also have a unique view on the executable scenarios. Each stakeholder can benefit in different ways through the models and artefacts of the ExSce tooling. In this section, the advantages of the tooling are explored for various stakeholders, through user stories highlighting their objectives and benefits, and scenario descriptions that illustrate how they would use scenarios for achieving their goals. In the following, we provide concrete, *representative* examples across SESAME use cases.

[13] http://wiki.ros.org/rosbag

### 2.3.1 Use case: Disinfecting Hospital Environments using Robotic Teams

**User Story 2.1. As a** *safety engineer* for a hospital disinfection robot, **I want** the robot to detect persons in its surrounding area **so that** the robot can safely operate its UVC light.

**Scenario 2.1.1. Given** a person is <occluded>, **when** the robot encounters the person, **then** the robot detects the person **and** the UVC lights are turned off after at most <time limit>.

**Scenario 2.1.2. Given** robot is outside the room **and** the room is lit with <lighting condition> **and** a person is inside the room, **when** the robot enters the room, **then** the robot detects the person **and** the UVC lights are turned off after at most <time limit>.

The tasks of a safety engineer should be supported by the descriptions of the scenario, by enabling the modelling of hazardous situations and the appropriate response of the system. Variation points can be integrated into the models to generalise the scenarios. In the scenarios above, these are introduced with terms surrounded by angle brackets (<...>). Specifically, in the first scenario, the person (as seen by the robot) is occluded to varying degrees; for instance, a part of their body or face is covered. In the second scenario, the room can be lit with varying lighting conditions, such as sunlight or luminescent light. Given a model of those variation points, a tool to support the transformation activity in the ExSce Workbench could sample from the specified variations to generate inputs to a simulator. With support of a tool, different values for the variation points can be selected, and appropriate artefacts can be generated, so that the system can be exposed to new situations that might uncover edge cases.

The person detection component is mission-critical to safely perform the disinfection task. This component could be powered by a machine learning model, which performance can be negatively affected when trained over data that is not representative of all the possible situations. As it is a critical component, it is important to generalise the training and verification process. This could be done by changing the posture of a person: standing in the centre of the room, standing next to a wall, seated behind a desk; and changing the orientation of the person: facing directly to the robot, turned 90 degrees, or facing away completely.

**User Story 2.2. As a** *component supplier* for a hospital disinfection robot, **I want** the robot to navigate in long, featureless corridors **so that** it successfully reaches its goal.

**Scenario 2.2.1. Given** a long corridor without distinct features **and** the robot has the correct map of the environment, **when** the robot is sent a navigation goal on the other side of the corridor, **then** the robot navigates to the goal **and** the robot is localised for the entire task.

**Scenario 2.2.2. Given** a long corridor with carts **and** the robot has no knowledge about the carts, **when** the robot is sent a navigation goal on the other side of the corridor, **then** the robot navigates to the goal **and** the robot is localised for the entire task.

Component suppliers should be able to verify their components and assess their performance in simulation. However, this can be cumbersome to implement as this requires a 3D model of the test environment. The creation of one 3D model of an environment can consume too much time and effort with traditional manual methods, such as modelling with a 3D editor. Furthermore, the effort needed for simulation-based testing increases when variations of the environment are desired. The ExSce Workbench can reduce this effort by providing models to specify environments and their variations (specification activity). Then during the transformation activity a tool can automatically generate, among others, a wide variety of 3D simulator models. When the component supplier detects a failure case of the component, he would introduce further, more targeted variations of the scenario during the generalization activity. On the one hand, this helps to narrow down the origins of the problem, on the other hand those additional scenarios also provides regression tests to detect the failure early if it occurs again in the future.

### 2.3.2 Use case: Dependable Multi-Robot Systems in Battery Innovation Centre

**User Story 2.3. As a** *behaviour developer* for a robot in the battery innovation centre, **I want** to specify how the robot moves to pick up a battery, **so that** I can adapt the robots' motion to different locations of the battery.

**Scenario 2.3.1. Given** a battery is located <location> in front of one robot manipulator, **when** the start signal is triggered, **then** the manipulator moves to lift the battery, **and** the moving manipulator remains within the tolerances of the <motion specification>.

In the battery innovation centre use case, the robot arms may need to manipulate batteries from different surfaces, e.g. from an elevated surface, the floor, or transported by another robot. A *behaviour developer* may wish to choose specific motion specifications that are most suitable for the particular location where the battery is placed, as shown in Scenario 2.3.1. In this context, the ExSce workbench may provide tools to specify different motions for robot manipulators.

**User Story 2.4. As a** *function developer* for a battery innovation centre, **I want** to use two robot arms to lift a battery safely, **so that** heavier batteries can be manipulated.

**Scenario 2.4.1. Given** a battery is placed between two robot manipulators on a flat surface, **when** the start signal is triggered, **then** the manipulators hold the battery at <height> for <duration> **and** the end-effectors do not exert more than <force limit> on the battery.

In this use case, there also exist heavier batteries whose weight may exceed the specified limit of some robot manipulators. The use of two manipulators to handle such cases is of interest to *function developers*. A criterion to verify the success of the dual-arm lift can be the battery being held at some height for some predefined duration. A criterion to ensure that the robot arms will not damage the battery can be a limit on how much force the end-effectors may exert on the battery. In this context, the ExSce workbench may provide tools to link the instrumentation needs as specified in acceptance criteria, here in the form of BDD scenarios, to the corresponding monitors that can provide the relevant information available in the system.

### 2.3.3 Use case: Autonomous Pest Management in Viticulture

**User Story 2.5. As a** *function developer* of the drone's trajectory planner, **I want** the planner to minimize the distance between the robot's current position and the next waypoint **so that** the robot passes through all the specified waypoints.

**Scenario 2.5.1. Given** a trajectory with several waypoints, **and** all waypoints are in free space, **when** the plan is generated, **then** the trajectory passes through all the waypoints.

**Scenario 2.5.2. Given** a trajectory with several waypoints, **and** obstacles are located near those waypoints, **when** the trajectory is generated, **then** the trajectory passes through all waypoints, **and** the drone maintains a distance to the obstacles of more than 50cm.

**Scenario 2.5.3. Given** a trajectory with several waypoints, **and** a dynamic obstacle, **when** the drone executes the trajectory, **then** the drone passes through all waypoints, **and** the drone keeps a minimal distance to the obstacle of 50cm.

The function developer for the trajectory planner of the drones will want to verify and validate that the functionality they designed behaves as expected. As this is a single component running in a robot within an MRS, there are different levels at which this can be done, for example, verifying the correctness of the optimization algorithm, verifying and validating its behaviour in an integrated system using simulation, and validating it in the real world against application requirements where its online behaviour to external phenomena can be

tested. This also implies that throughout the development cycle, the function developer will communicate with other stakeholders about the planner's role in the system.

In this user story, developers will require minimally a task specification (e.g. the list of waypoints to be used to generate commands) and the chosen optimization parameters of the planner itself. Depending on the level at which the planner is being evaluated, developers need to evaluate the acceptance criteria given the task and parameters being used. The execution of the scenarios can be done offline, verifying algorithmically that the planner produces the correct output, or online using existing ROS-based tools to simulate or perform tests in the real world. An analysis of historical execution traces can help identify the boundaries at which the functionality degrades or fails, particularly when the planner is tested as part of the system. Generating scenarios with variations across deployment sites (e.g. a different vineyard on a steep hill) and obstacles (e.g., multiple dynamic obstacles of different sizes flying at different, potentially varying, speeds) provides new, unseen input data for these tests (e.g. waypoints that require sharp turns or tasks that also specify the orientation for each waypoint), and requires a corresponding acceptance criteria to be specified (e.g., tolerances on the computed trajectory).

**User Story 2.6. As a** *performance designer* of an autonomous pest management drone, **I want** the drone to spray only infected vines **so that** fungicide is used efficiently.

**Scenario 2.6.1. Given** a row with vines, **and** an infected vine V, **and** adjacent vines U and W, **when** the robot is spraying vine V, **then** its altitude is 2.5 m above the canopy of vine V, **and** the localization accuracy is not below 15 cm, **and** no fungicide is sprayed on vine U and W.

**Scenario 2.6.2. Given** a row with vines, **and** an infected vine V, **and** adjacent vines U and W, **and** the robot is spraying vine V, **when** wind increases, **then** its altitude is 2.5 m above the canopy of vine V, **and** the localization accuracy is not below 25 cm, **and** no fungicide is sprayed on vine U and W.

**Scenario 2.6.3. Given** a row with vines, **and** an infected vine V, **and** adjacent vines U and W, **and** the robot is spraying vine V, **when** connection is lost to the Global Navigation Satellite System (GNSS), **then** its altitude is 2.5 m above the canopy of vine V, **and** the localization accuracy does not fall below 25 cm, **and** no fungicide is sprayed on vine U and W.

A performance designer looking at the efficiency of the spraying drone will examine its nominal behaviour to ensure the NFR for spraying are met. To evaluate this, the scenarios must define new or reuse existing robot and environment models, the task specification (e.g. the waypoints of the vines to be sprayed) and the drone's configuration parameters. As acceptance criteria, the drone's position should be monitored and compared against the ground truth in simulation and the vines it has sprayed, and whether healthy vines were sprayed. In the scenarios defined above, the performance designer of the spraying drone has set the lower threshold to 15 cm for a nominal case ("good" performance) and 25 cm for scenarios with disturbances or failures ("sufficient"). Note that "good" and "sufficient" localization accuracy thresholds are much different for the data collection drone in this same use case (where the sufficient values are between 2 and 3 cm). The execution can take place in simulation or the real world, and would require the ability to introduce environmental disturbances (e.g. wind) and failures to the system (e.g. loss of communication with the GNSS). The acceptance criteria might depend on the specific target application, so a comprehensive set of scenarios would allow the designers to optimize configuration parameters to achieve a desired performance and specify the acceptable thresholds for e.g. the localization accuracy.

**User Story 2.7. As a** *safety engineer* of an autonomous pest management drone, **I want** the robot to react to high speed winds **so that** it does not damage the vines, other drones or people.

**Scenario 2.7.1. Given** a drone that is spraying a vine, **when** the wind speed exceeds the allowed limit, **then** the spraying component is stopped.

**Scenario 2.7.2. Given** a drone following a route, **when** the wind speed exceeds the allowed limit, **then** the drone does not fly at an altitude below 2.5 m from the canopy.

| Modelling |
|---|
| Metametamodel |
| Metamodel |
| Model |
| Realization |

| Quality assurance |
|---|
| Verification |
| Validation |
| Instrumentation |
| Acceptance criteria |

| Execution context |
|---|
| Real world |
| Hardware in the loop |
| Simulation |

| Robot |
|---|
| Fleet |
| Flock |
| Platform |
| Device |
| Component |

| Environment |
|---|
| Crowd |
| Traffic |
| Co-worker |
| Obstacle |
| Disturbance |

| Task |
|---|
| Skill |
| Motion |

Table 1: Various design dimensions associated with the ExSce (partially based on [19]). Blue-coloured tables represent core robotics domains that are present in the scenario specification.

**Scenario 2.7.3. Given** a drone A following a route **and** a remote sensing drone B following drone A, **when** the wind speed exceeds the allowed limit, **then** drone B will maintain a minimum distance of X meters.

A safety engineer will be interested in scenarios that describe potential hazards and dangerous situations. This user story must be verified and validated both in simulation and the real world. The former allows engineers to put the drones in dangerous situations without the risks of damaging the system, its environment or living beings. The latter is required to validate that the available mitigations work in practice and to reveal corner cases that had not occurred to engineers. In this user story, in addition to the usual elements of the scenario specification, engineers also want to specify the behaviour and parameters of the resulting disturbances from high wind speeds. The acceptance criteria corresponds to the application's safety requirements (e.g. no spraying during high winds, no flying below the safety altitude, safety distance to other drones). To simulate the wind disturbances, the required code must be developed or generated (e.g. as a Gazebo plugin). For safety engineers, the generation of new scenarios helps discover corner cases; for automated generation, the corresponding acceptance criteria could be automatically generated.

## 2.4 Design Dimensions, Concepts and Aspects of the Executable Scenario

While a pure scenario specification could be built from the robotics domains, this is insufficient for the *executable* scenario that also must consider *(i)* the realization of the robot's behaviour; and *(ii)* the execution context that the robot is embedded in. Additionally, the ExSce must be able to evaluate the robot's performance. Table 1 summarizes some recurring design dimensions of the ExSce with their major constituents.

### 2.4.1 Design dimensions and their concepts as focused views on the Executable Scenario

**Modelling.** The lowest level of abstraction in modelling is the system, which is also known as a realization. One level of abstraction higher are the models, which are specific abstractions that represent the systems. Each view or abstractions of the same system may introduce a new type of model, or the other way around, many models can represent the same system. The core relation in modelling is that of *conformance* or *conforms-to* [22] between models. A model conforms to its meta-model if it satisfies all the constraints imposed by the meta-model . Hence, a meta-model is one level of abstraction above models and defines the language in which models can be specified: a so-called domain-specific language. The mission specification in Listing 2 provides one example that conforms to the syntactic meta-model of PDDL and the semantic meta-meta-model of topology — here, the connection between rooms.

As we discussed in Section 1, a rich zoo of DSLs exists in robotics. An equivalently rich ecosystem of tools is available to create meta-models or DSLs and support features such as creating, modifying, checking, linking, transforming, storing or querying concrete models. Such tools can be realized within general-purpose programming languages, using narrowly-focused language workbenches or on top of graph databases. Even if the representation of models varies between tools, the meta-meta-models must remain stable, thus allowing the definition of model-to-model transformations between not only tools but also different types of models.

**Robotic domains.** In the above example, only the robotics domains referred more-or-less directly to robot-specific concepts, although at a very abstract level. The challenge in making a scenario executable and then evaluating it, is that all those concepts require a more detailed representation. Many of those (meta)models, fortunately, share a wide range of domain-independent meta-meta-models that have already been scrutinized and formalized, often even in "outside" fields of research. Some examples include meta-meta-models of *(i)* geometry which introduces concepts such as points, lines, frames, position or orientation; *(ii)* kinematics with concepts like velocity or acceleration; *(iii)* classical mechanics for momentum, force, or various "types" of energy; *(iv)* dynamics as the mappings between geometry and kinematics on the one side and classical mechanics on the other side; *(v)* control to make systems move and estimation for perception; *(vi)* algorithms for the composition of computations; *(vii)* block-port-connector for hierarchical system composition [101] as can be found in finite state machines, kinematic chains, dynamic Bayesian networks, component-based software systems or bond graphs; or *(viii)* directed-acyclic graphs to represent partial or total order. For a more elaborate treatment, we refer the interested reader to [19].

**Quality assurance.** The quality assurance design dimension in Table 1 includes elements which are necessary to assure that the system being developed satisfies the requirements from various stakeholders, e.g. user stories in Section 2.3. This typically involves defining criteria for verifying and validating that the listed requirements are met, e.g. the BDD scenarios listed in Section 2.3. This in turn requires instrumentation to collect the necessary information about the system and environment while the task is carried out. Such data can be information about the execution, such as state transitions to verify that the UVC lights are turned off when people are detected or execution time to verify that the lights are turned off quickly enough, in the hospital disinfection use case, e.g. in scenarios for User Story 2.1. Relevant information may also include estimations of internal states of the robots or external states of the environments. Examples from the battery innovation centre use case include the applied forces at the robot arms' end-effectors to verify that the batteries will not be damaged by the end-effector, and the height of the battery above a surface to verify that the batteries are lifted successfully (c.f. User Story 2.4).

While acceptance criteria can hint at the type of data necessary for verification, how such data can be acquired depends on the capabilities of the specific MRS. Verifying the forces applied by the arms' end-effectors on the battery, for example, is dependent on whether appropriate sensors are available on the robots. If joint torque sensors are available, the end-effector forces can be computed via forward dynamics, which demands both such capability to be available during execution and the appropriate data to be exposed for collection and monitoring. Furthermore, acquisition of the same information may differ vastly depending on whether execution is carried out in simulation or the real world. Verifying the localization of a mobile robot, as in the hospital disinfection or vineyard use cases, implies that ground truth position of the robot is available, which can be directly acquired in simulation but is not trivial in the real world. Finally, instrumentation needs may change over time, e.g. force estimation in the battery lifting use case is only relevant when the end-effectors are in contact with the battery, which suggests on-demand adaptations of monitoring instruments are desirable, both to conserve computing resources and to avoid collection of meaningless data.

**Execution context.** The execution context determines if the system is to be executed fully in simulation, partially in simulation and real world, or completely in the real world. This has an impact on points like setup and execution times (tend to be lower in simulation than in the real world), availability of data (usually better in

simulation) or quality of data (more representative in the real world). Section 5 discusses this design dimension and its joint points with other design dimensions in depth.

### 2.4.2 Aspects of cross-cutting concerns in the Executable Scenario

An exhaustive, in-depth analysis of all design dimensions and their interactions is prohibitively expensive. Hence, guided and motivated by the use cases and challenges defined in SESAME, in the following chapters and sections we investigate specific views on or slices through the design dimensions in more detail. Following similar terminology as in aspect-oriented programming [58], we call such a cross-cutting view an *aspect*.

Then, the remainder of this deliverable focuses on the following aspects:

1. A further analysis of the concepts that underlie the ExSce itself to provide first steps to a formalization of the ExSce (Section 3)
2. An analysis of future research and development directions of the ExSce guided by developer interviews that provide interesting insights into real-world testing of robotic systems and show opportunities for the ExSce, especially in the quality assurance (Section 4)
3. Transformation and simulated execution of scenarios for the mobile bases in the Locomotec use case (Section 5).

# 3 First Steps Towards a Formalization of the Executable Scenario

Scenario specification is at the core of the ExSce methodology, and is an activity that is cross-cutting across design dimensions and stakeholder views. As the first steps towards a formalization of the executable scenarios, we identify concepts to specify missions, environment, robots and safety concepts from prior and related work.

## 3.1 User story and behaviour-driven development meta-meta-model

We have already seen in Section 2 that the BDD community has spent significant time and effort to establish a meta-meta-model for user stories and scenarios that we are planning to reuse in the ExSce. The main concepts *are* the placeholders in the textual notation and include

- the `role` or stakeholder;
- the `activity` to be performed; and
- the `goal` or benefit from that activity;

The `narrative` is the composition relation of the previous three concepts. Next, BDD introduces the `acceptance criteria` as the composition relation of a set for each of the following three concepts

- the `given` initial situation;
- the `when` trigger or precondition of the scenario; and
- the `then` effect as the outcome of the scenario.

The `user story` is the composition relation between exactly one `narrative` and a set of `acceptance criteria`. While Gherkin also offers keywords such as `and`, `but`, `scenario template` or `background` those are just syntactic sugar over the core concepts.

We consider this to be a good meta-meta-model *because* it is simple and understandable in a time span of a few minutes. In the context of SESAME the next challenge is to build meta-models that compose this meta-meta-model with the concepts from the robotics domains. Additionally, we consider to compose the user story meta-meta-model with the DAG meta-meta-model to realize time-wise orderings between scenarios.

## 3.2 Scenario Specification Concepts

In SAFEMUV[14], we identified concepts and relationships for an inspection application of a multi-drone team, which shares some commonalities with the viticulture use case. Figure 4 shows the relations between the identified concepts. This graphical model offers a birds-eye view of the relationships between different concepts across design dimensions and domains. Whenever possible, we have attempted to generalize the concepts from the UAV-specific application to more abstract SESAME concepts. However, this generalization is not exhaustive: additional concepts and relationships derived from the design dimensions and identified in the ExSce methodology, as well as concepts for specific stakeholder views will be added, e.g. provenance concepts for quality assurance[15] and concepts for the execution context.

Tables 2, 3 and 4 show the definitions of robot, mission and environment, and safety concepts initially identified. Robotic specification concepts will be integrated into the system modelling. Environment and robotic models and meta-model s required for the execution of scenarios in simulation will be further discussed in Section 5.

[14] https://www.york.ac.uk/assuring-autonomy/demonstrators/remote-inspection-using-drones/

[15] https://www.w3.org/TR/prov-overview/

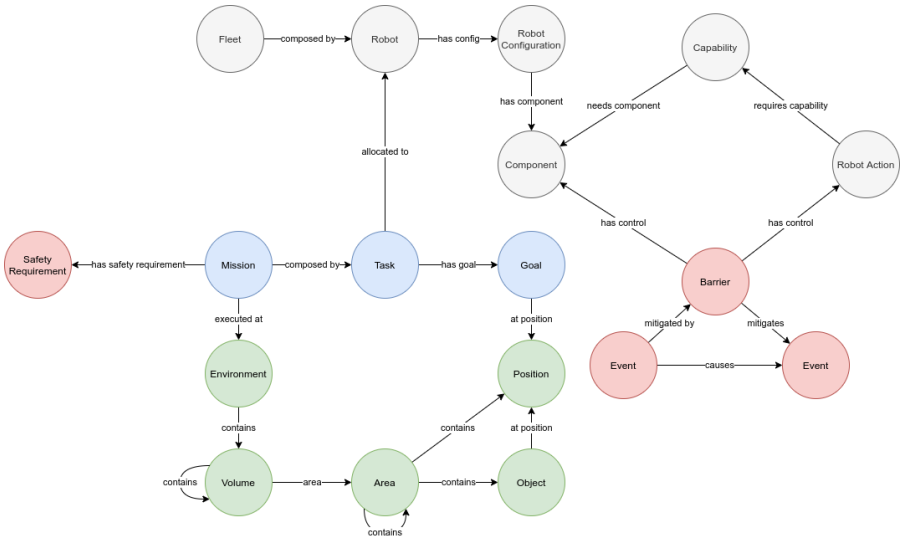



Figure 4: Concepts (circles) and relationships (directed arrows) for specifying scenarios: Missions (blue), robot (grey), environment (green) and safety (red) concepts.

Table 2: Robot specification concepts

| Concept | Definition |
|---|---|
| Fleet | A set of robots deployed for a specific application. |
| Robot | A specific instance of a robot as defined in [63]. Robot concepts should be interpreted as a (recursively-defined) composite. |
| Robot Configuration | A robot's composition at a particular point in time. |
| Component | Hardware, software or information objects according to the component taxonomy in [63]. |
| Capability | The ability of a robot to perform an action [63]. |
| Robot Action | An action performed by a robot as defined in [63]. |
| Operator Action | Actions operators, such as a pilot, can perform in teleoperated or semi-autonomous operations. |

For each use case, we will investigate which stakeholder views require additional concepts. From the perspective of a safety engineer, for example, application-specific risk assessment concepts derived from their regulatory requirements might need to be considered. In a more abstract level, Table 4 describes the basic concepts from Bow-Tie Diagrams – a graphical model that establishes relationships between a hazard and a top event (an undesired situation), and its causes (threats) and effects (consequences) – as a starting place. Events are related to **when** clauses in BDD as triggers for the scenarios (threats) and **then** clauses as the (undesired) outcomes. Barriers and controls are closely related to requirements and acceptance criteria, as they are test conditions that need to be verified and validated according to internal or external regulations.

It is important to emphasize that the concepts described above are the elementary concepts which will be formalized and extended as meta-models in Task 3.2 (Executable Scenarios Metamodels) of WP 3.

Table 3: Mission specification concepts

| Concept | Definition |
|---|---|
| Mission | A mission is a high-level task which should be achieved by the fleet. |
| Task | A unit of work that is *allocatable* to a robot or robots. |
| Goal | The specification of the post-conditions of an executed task. |
| Environment | A description of the physical location where a mission takes place. |
| Volume | A 3D region, usually with some semantic meaning attached, that restricts the spatial boundaries where a robot should operate, e.g. operational volumes for a mission or forbidden regions for a robot. |
| Area | A 2D region that restricts the spatial boundaries where a robot should operate. Often a simplified version of a volume with infinite $z$. |
| Object | Bodies on the environment with which a robot interacts during its tasks. Different types of objects are usually present in an environment, e.g. obstacles are objects that robots avoid. |
| Pose | A geometric description of a location. |
| Requirement | The criteria a mission should satisfy |

Table 4: Bow-Tie Diagram concepts from [32]

| Concept | Definition |
|---|---|
| Event | "An undesired system state where control over the hazard is lost". |
| Barrier | A collection of mitigation strategies that can be composed by multiple controls. |
| Control | "Any process, function, device, practice or other action that modifies a safety risk" |

# 4 Developer Interviews: Empirical Analysis of Executable Scenarios in the SESAME Context

An organization's testing policy depends on contextual factors, including operational constraints (e.g. available resources), organizational policies, internal and external standards, among others. In this section, we provide etnographic details of Locomotec and identify operational constraints that influence their test process. We also analyse additional factors in this subsection, based on interviews conducted with two developers at Locomotec to examine their quality assurance (QA) practices for their mobile disinfecction robot, and the test artefacts for their latest test campaign. The QA techniques employed by engineers, who both are in charge of development and testing, is key to ensure not only the correct functionality of the robot, but also to ensure that the safety features, e.g. mitigating the risks of overexposure to UVC light, are sufficient according to standards. Finally, we discuss development and testing challenges faced by engineers, and insights relevant for the ExSce in this section.

Locomotec has 20 software developers, out of which, five are in charge of developing the ROS packages and other software components used by the KELO AD robot. The experience level of the two senior developers is on average 15 years, while the three junior developers have on average 4 years of experience. The team does not strictly follow agile practices as part of their organization, but have adopted parts of them into their development workflow, e.g. the duration of the sprints is variable according to the feature or issues currently being addressed, and a subset of the developers is mostly in charge of the tests of all features (one senior and two junior developers).

## 4.1 Test Process

Given that their development life cycle follows a mixture of models, they have different test strategies for different parts of their life cycle. All their testing activities are motivated by the compliance with standards for the UVC robot, and developers had regular testing sessions in order to comply with the minimum number of testing hours required by internal guidelines as well. As part of their development process, and based on their experience, they set a certain number of Key Performance Indicators (KPI), like kilometers the robot should have navigated by the time they reach a given project milestone. This has the goal of exposing the robot to a wide variety of situations in the real-world and is used by developers to estimate the number of hours a week they should dedicate to testing activities. However, despite the considerable effort required for testing, it is unclear what type of evidence they need to provide for certification. On Figure 5 the two main test processes and its activities are illustrated.

On so-called *field tests*, engineers follow short, agile-like dev-test cycles. These testing sessions were held two to three times a week, and had a duration of 4 hours on average, where engineers split their time roughly 50-50 between development and testing activities. Test design and implementation activities were combined and tests were usually executed right after.

*Endurance tests* follow a more classical test process. These types of tests usually take place full-time over a week, where the pre-defined test cases are run multiple times. These sessions required a more structured planning process: engineers planned these sessions in a dedicated meeting where the test team defines the goals of the test campaign, which test suites to focus on, and which test cases to include. The exit criteria for the tests is mainly focused on robustness, i.e. the failure rates of components should be acceptable according to standards and the risk assessment of the robot, and is usually measured in mean time to failure, e.g. overexposure or collision. Engineers carry out the first four activities of the test process (cf. Figure 5, Endurance tests) in this meeting as well.

Test planning is informed by the standards the team is using as a baseline to argue about the safety of their system – three of those are related to the safety of mobile robots and two are specific to the UVC light emissions

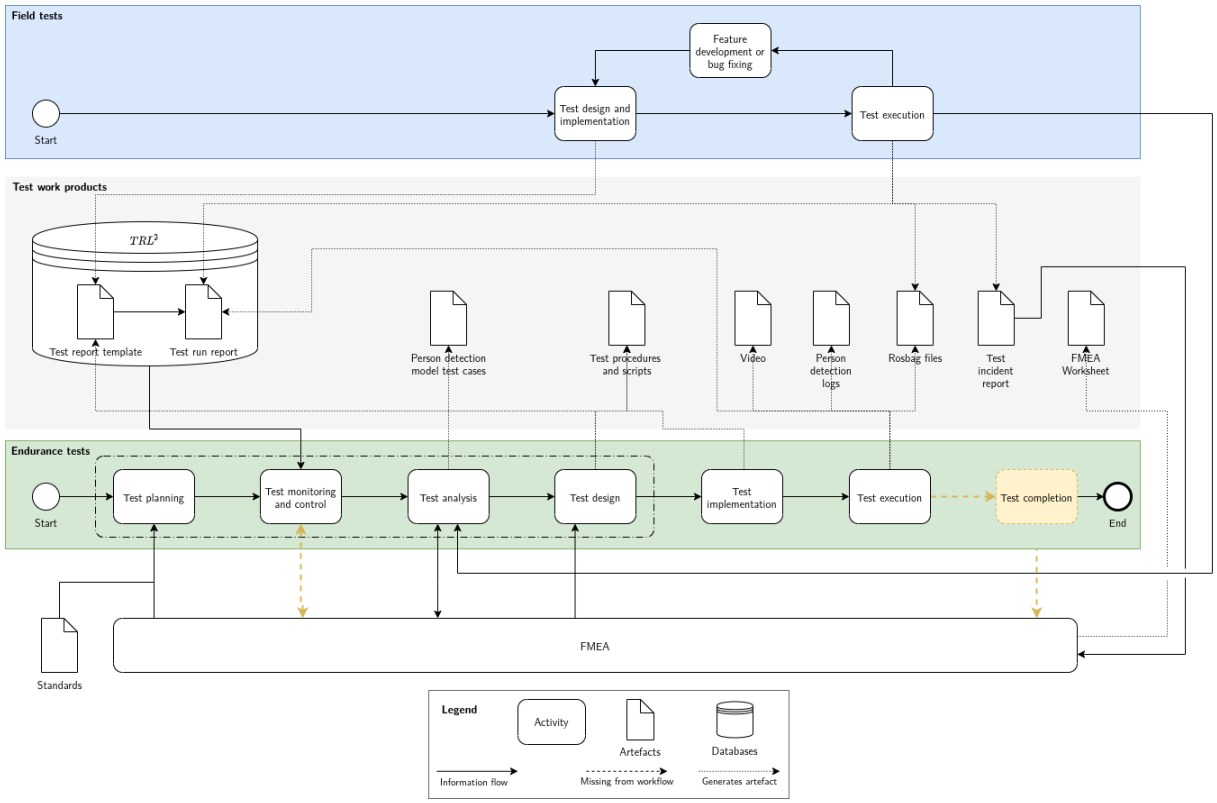



Figure 5: Test processes for the UVC Arodis robot, including the activities covered in the field (blue) and endurance test process (green). Partially implemented activities or information flow are in yellow. Test work products are in grey.

– and by the Failure Modes and Effects Analysis (FMEA) that is conducted in parallel by the engineers together with the hardware team.

The test monitoring and control activities are mostly carried out by the engineer leading the testing efforts, who examines test results and logs and assesses the progress of the test campaign according to the exit criteria. Together with the testing team, they also determine if additional test runs or new test cases are required. Test analysis is also tightly related to the FMEA process which heavily influences what to test. Both activities reveal defects of the system under test and help developers prioritize which features to test first. Test design techniques employed by engineers will be discussed in more detail in Section 4.2.

Test implementation activities take place shortly before the scheduled test dates where tests are finally executed. Test artefacts for these activities are discussed in more detail in Section 4.3.

## 4.2 Testing Practices

Conventional testing techniques, like the use of unit tests and continuous integration (CI) tools only play a smaller role on the ARODIS robot. Interestingly, these techniques have been used for many years on another robot developed by Locomotec. This choice is related to the cost of developing and maintaining testing infrastructure [5] and the return on investment of applying these techniques. Tests are conducted indoors, with network access and on a higher frequency than their other robot, which is tested outdoors on remote locations and with no network. In practice, this means that the time required to find and debug issues is much shorter for ARODIS and therefore the benefits of unit or integration tests that could be run on CI is much lower.

Experience-based testing is the predominant test design technique, engineers use their experience and knowledge to design and choose test cases. Exploratory testing and error guessing are the test design techniques

used at the current TRL: Rather than define clear exit criteria for their tests, engineers are more interested in evaluating the performance of the robot over time or find "ways to break the robo".

Equivalence partitioning is another technique used by developers to minimize test case explosion and target those use cases where they would gain the most (new) information. Testers divide possible test input variations into equivalent classes, taking into account information from safety risks identified through the FMEA and developers' experience. Discrete variables such as presence of occlusions, or the standing/sitting position are straightforward to identify as partitions. Continuous variables are partitioned with information from the FMEA, component specifications (e.g. UVC lamps and sensors), and engineers' experiences.

As an example, to identify partitions in the distance between robot and person for person detection tests, engineers created a table that listed possible sensor configurations as rows and the distance to people as columns. The latter had increments based on their experience and the boundaries at which the sensor could detect a person, as well as the range at which UVC light is no longer dangerous. Based on this information, cells were classified into high confidence of detection or no danger (green), uncertain about detection (yellow) and high confidence of failure to detect people (red). Yellow cells and their boundaries to green and red areas are the main focus of the tests cases, since they provide information about the boundaries at which the robot could operate safely.

## 4.3 Test Work Products

A number of test products are created as the result of activities in the test process as shown in Figure 5.

Test scripts include three classes of tests: person detection (TCP), navigation (TCC) and person detection and navigation combined (TCPC). Both TCP and TCC test cases describe component-level tests, whose goal is to evaluate the performance of those components. The former evaluates the people detection based on the number of false positives and false negatives, as well as the model's precision and recall. The latter evaluates the navigation component as it follows a disinfection route based on the time required to disinfect the area and the distance traveled. TCC is on occasion also used as an integration test between the navigation and the person detection components, without evaluating the performance of the latter. Finally, TCPC test cases are system-level tests with the objective of testing the full disinfection workflow, combining the disinfection task with its safe operation based on the people detection component. In addition to the performance metrics of the two prior test cases, they record the amount of UVC light a person is exposed to during the task. TCPC is a performance test where they measure how efficient the system is at disinfecting (by measuring the UVC intensity with the help of a hand held UVC light meter).

Each script contains the goal of the test class, which are the relevant statistics for these types of tests, and the test suites included on each class. Subsequently, each test suite includes its description, which raw information to record from the robot, which statistics to compute (based on the test class statistics, but if necessary adding relevant ones for the test suite), assumptions from the testers, and a table listing each test case, and the values of each input variable based on the equivalence partitions. Perception tests were the focus of this test campaign and clearly represent the largest proportion in number of test cases and variations to be tested. These test cases had the longest "lifespan" on average, i.e. the number of days between the first and last test run for a given test case, and the largest variations in the number of runs per test case.

Test reporting is done on an in-house tool called "Technology Readiness Level Test Report Library" ($TRL^2$). Once test cases are defined, the $TRL^2$ tool is updated based on the test scripts. The tool uses a formal model based on the ISO 29119 standard for software testing to define the test report template. After a run, these models are presented to the tester as a web form with fields for the tester to input information manually, and provide developers with a way to add observations to the test runs right after their execution. These reports can contain multiple runs allowing testers to collect data about different variations in the test suite based on their test case definition. The tool was developed to improve the traceability between requirements (FMEA), software (bugs and hardware issues) and test runs. However, the time required to record and annotate test runs,

along with the cost of development and maintenance of the tool itself are prohibitive for such a small team. This is in line with other criteria used by the team about the cost of testing [5].

As for log files, engineers rely on the rosbag tool[16] to record a subset of topics relevant to the test suite being tested. The replaying capabilities of rosbag files also play an important role when testing the people detection component, as they can be used to assess its performance on real-world data after models have been updated. Person detection logs contain statistics about the performance of the person detection component; this information is also contained in the rosbag files, but these CSV files provide engineers easier access to the data needed for analysis without having to replay each rosbag file. In addition to that, engineers store the video stream on some runs which can be reused to test the person detection models.

They also use a git repository to keep track of test incidents (bugs/issues that were not fixable in a field test session), mostly hardware related. In addition to some basic metadata, these incident reports contain the software versions and hardware configuration of the robot, links to other reports (test run reports, other test incidents, git issues) and the risk assessment of the failures. This assessment is quantified for the FMEA by computing the risk priority number (RPN) of the issue based on (i) the severity of the fault, (ii) the likelihood of it occurring and (iii) the likelihood that it can be detected. Once the fix has been implemented, the software and hardware versions of the components that include the fix, along with the updated risk assessment are added to the issue.

## 4.4 Development and Testing Challenges

Quality assurance processes can be supported by the ExSce methodology and tools (Figure 3), and have cross-cutting implications for the design dimensions of the ExSce. The interviewed engineers reported a high level of confidence on their test process, but acknowledge that the variations they have considered in the defined test cases might not be sufficient to assure that KPIs are reached in other environments. On the question of whether the amount of testing they had carried out was sufficient, an interviewee said:

> *No. [Navigation tests do not] account for changes in the environment. If there are now many tables and chairs suddenly in the corridor, this we didn't test. So I would like to have tested all possible combinations to be perfectly sure, but that is not feasible. [...] we have not explored simulation [to vary the environment] so I would like to have a better understanding of all kinds of clutterness, messiness, extremely long corridors and lots of rooms that you can not test in reality with a real field-test. [Under] normal conditions I feel reasonably comfortable, but everything that is a bit beyond what we have so far tested I just want to be on the safe side but this is very difficult to create. And for person detection I think we have already good coverage I have also a good feeling here, but I would like to have some tool that says you have indeed not overlooked a particular scenario or something that you missed completely. [...] tests are [carried out] under controlled conditions, but how does it scale up? I'm also rather confident that in the normal cases it works as expected but I would like to know what we have forgotten or corner cases. [...] something that is beyond what we have thought in our variability or we made this discretization in meters and maybe there was something we overlooked or we should have done it with more resolution or we should have tested smaller or longer [distances]. What I would really like is to have stressy environments that I cannot easily reproduce in reality, just to be sure that the performance is as in the other tests.*

This challenge is related to the variability of the environment, as engineers have a limited amount of time and resources and it is impossible to test all possible combinations. Even if resources were not a constraint, the tests engineers have conducted only seem sufficient for the variations the physical environment allows

[16] http://wiki.ros.org/rosbag/Commandline

them; engineers lack the ability or resources to manufacture real-world scenarios that cause corner-cases and environment variations beyond what their test bed offers. Test analysis and design activities can be supported by minimizing the bias of purely experience-based techniques, and data from executed scenarios can provide an overview of the test space.

While the current test processes are informed by risk assessment activities and standards, stakeholders rely on handcrafted methodologies and tools to integrate them. Assured scenarios can be used to generalize scenarios and to identify ExSce templates, enabling stakeholders to test a wider set of variations in simulation. Ideally, the information flow between the risk assessment processes (e.g. FMEA) and standards and activities in the test process would be bidirectional. For example, for the dashed arrow in Figure 5 between the test analysis and FMEA activities, the assurance of the FMEA risks would be included in the exit criteria, and the results and progress of the testing activities adds or updates risks in the FMEA worksheet. About the flow of information between test analysis activities and the FMEA, an interviewee said:

> *[...] we don't really do that so much, but this is how we would like to do that. The reason is that we don't have good tooling for that, but what I would like to do is we are working on the person detection component, and we know there are some failure modes, [e.g.] person is not detected and [...] we would like to have the statistics automatically linked to the likelihood of an event happening, so we can trace back that the likelihood is very low because of this test campaign.*

And when asked if there was information flow between test analysis (what to test) and test design (how to test) and the FMEA process, an interviewee answered:

> *[Test analysis and design is] based on mitigations and subcomponents of the FMEA. There is an assumption, this is kind of not standard, that we have a hierarchical decomposition of the system that is used for FMEA, which is by definition of FMEA not necessarily the case.*

For this, stakeholders need to maintain the traceability between (safety) requirements and executed scenarios. The ExSce methodology and tools can support these activities through the provenance data of executed scenarios. Provenance data allows stakeholders to trace back executed scenarios to requirements, scenario specifications and acceptance criteria. As a result, these data can also be used as evidence of a SUT's compliance with standards, and support stakeholders with test monitoring and control activities. When asked why did they choose to develop TRL instead of using another (automated) tool to manage their tests, engineers responded:

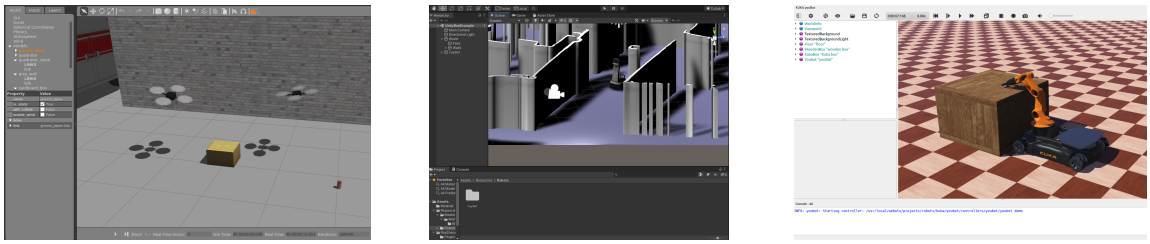
> *Other tools didn't fulfil our requirements at the time. The idea was that this would support the overall testing process. There is a feature in TRL [tool] where a test report templates are defined, and testers would follow some steps: define the template for each test, including its goals and the steps to follow, and then they would go ahead and fill out the test reports. It was supposed to have some automated information from the robot. At the time, there was no fully integrated system for this that linked all the information together. We don't fully exploit all the features because the current system is a bit tedious and requires a lot of [manual] effort, so only the test reports are used during prototyping. The goal of the [TRL tool] workflow was to first model the system for the FMEA, then define tests for [each of] the components, and generate reports for them. It was also integrated to issues on GitLab so you could create tests for a specific issue or once the test is done, it was also possible to create issues directly from the report interface or close related ones.*

An important aspect to consider as part of the process/methodology design dimension during the development of the ExSce methodology and tools is the return on investment for stakeholders. Given that the cost and resources of testing robotic systems are already a challenge for many [5], and is in line with the rationale for (not) adopting test practices by engineers as described in Section 4.2, the ExSce should support and integrate

well with existing development workflows, and the benefits and insights obtained by stakeholders from the ExSce methodology and tools should exceed their implementation and maintenance costs. Tailorability is also a key aspect of the generalization of scenarios, as it should support the reuse of models across applications (e.g. the ARODIS robot shares hardware components with other robot models for logistic tasks). Similarly, the ExSce tooling should support integration with existing development frameworks and enable developers to easily query information about executed and assured scenarios for analysis. Engineers explain why they create person detection logs even though the information is available on the rosbag files of their experiments:

> *It is just an easier way for me to create scripts and statistics because I just create all this CSV files and [load them] on a Jupyter notebook, process all the data, and create statistics for typical measures of object detection.*

Although rosbag is a common tool used by testers [5], its data is only available when replaying the files and therefore requires additional processing to make the performance metrics of components accessible for analysis throughout the test process.



(a) Gazebo (b) Unity (c) WeBots

Figure 6: Three representatives of multi-domain simulators currently available that enable the simulation of robotic platforms.

# 5 Scenario Execution and Simulation

Verification and validation (V&V) is a key step in the development of any software application. This is especially true for cyber-physical systems, as their interactions with the environment poses safety and security concerns that are stricter than in other software domains. Hence, numerous tests are needed to reach a level of confidence that all requirements are being fulfilled, including those regarding safety and security. This activity can be performed in the real world with the robot, or in a virtual world with an abstraction of the system and its environment. Both have their strengths and weaknesses, but in general they both can be used to reach the confidence level required to validate the application by finding faults [106]. To use a virtual option is to use a simulator, which can be used at various levels of abstraction of the system and during different stages of development.

Modelling the system and specifying the scenarios are fundamental steps in the development process to document the system and its working environment. Through the ExSce methodology, the process of verifying and validating can be streamlined by exploiting the models from the previous design stages. While scenarios can be executed in real and simulated environments, there is a real value in using simulation for the automatic set-up and execution of scenarios with faster execution than real-time. For this reason, this section focuses on simulators and their capabilities to implement scenarios.

Simulation as a tool is valuable for many reasons. It can be used to test the capabilities of the system before there is a physical platform, enabling prototyping and software building at early stages of development. It can also be used to assess functionality and find faults when the tasks to perform or the environment are considered unsafe, reducing economic losses and health risks [106]. Furthermore, simulation enables the automatic and unsupervised execution of scenarios, whose results can be automatically or manually graded. However, there are challenges that complicate the integration of simulation into the development process, such as the reality gap and the complexity of setting up the simulation environment [6]. There are numerous simulators available for commercial or research purposes, so developers who wish to take advantage of current simulators need to consider their options carefully. Between simulators, there are many differences which have an impact on the results of the simulated tasks, and therefore, in the confidence level of the results of the scenarios.

## 5.1 Overview of Current Simulators

In recent years many simulators have been released to the roboticist community. The efficiency and accuracy of these simulators depends on many factors; among them are the abstractions chosen by the designers and developers of two important components: the physics and the visualisation engines. Current simulators rely on physics engines to perform the calculations required to simulate the mechanical interactions of the real world through models based on the study areas such as rigid body dynamics, soft-body dynamics, fluid dynamics. Popular physics engines include the Open Dynamics Engine [102], Bullet Physics [26], and MuJoCo [107]. Engines can differ in their representation of bodies in space and their constraints, the calculation of velocities

and forces, and other abstractions required to approximate the behaviour of bodies in the real world. All of these aspects affect the fidelity and effectiveness of the engine.

In addition, most simulators rely on 3D engines to render the graphics for the visualisation of the simulation. Engines such as OGRE [17], Blender[18], and the Open 3D Engine [19] must perform complex operations, such as ray tracing, that take into account the attributes of the scene and the 3D models within it to generate the visuals. To achieve it, the engines consider the perspective of the camera, the textures of the object and their opacity, and the lighting of the scene and its effect on the objects. While this might seem trivial as many simulators can be run without visualisations (headless), the rendering process impacts the realism of the generated images, which are used for simulating camera sensors. As rendering is computationally expensive, simulators either do not run at real-time rendering photo-realistic scenes, or run with lower-quality rendering to keep real-time performance.

The efficiency of a simulator is a predominant concern for many developers of MRS. As the number of robots included in a scenario increases, the efficiency decreases. This is a result from an increase of state variables for the bodies (such as velocity and acceleration) that have to be updated, an increase of movement constraints (such as joints in a manipulator) that need to be checked, and a rise in collision tests that must be performed at the end of the time step during the simulation. Depending on the abstractions, the geometrical representation of bodies, and integration methods, the efficiency of simulators when more than one robot is simulated varies greatly [90]. As the overall efficiency decreases and resource consumption grows, more time it is needed to complete a single run, making simulation challenging for MRS with a high number of robots in the scenario.

Apart from the differences in performance, simulators may differ in the robotic platforms, tasks, and environments they can simulate. Some simulators offer narrow abstractions to simulate a specific type of tasks with high fidelity, such as GraspIt! [76], which enables the accurate simulation of manipulation-related tasks with various robotic grippers. Other simulators may focus on simulating specific platforms, such as UTSim [7], which supports the simulation of UAVs MRS by offering relevant abstractions for the type of application. This includes general attributes such as weather conditions and communication problems, as well as domain-specific entities such as ground control stations. There are also environmental abstractions that are required to accurately simulate an application. For instance in the case of underwater robots, the performance of their sensors depends on environmental conditions found in the water, including currents, temperature fluctuation, salinity and pressure [34].

The majority of commercially available robotic simulators are multi-domain simulators, as they offer abstractions to support a wide-range of applications with a sufficient fidelity for most use cases. Popular simulators in this category include Gazebo [61], Webots [75], MORSE [36], ARGoS [89], and CoppeliaSim [96] (formally known as V-REP). Their features facilitate the simulation of tasks such as navigation, planning, perception, manipulation, mapping, and learning. Even with similar features and capabilities, choosing one simulator to perform a certain test can still be a challenge, as performance can be impacted by the characteristics of the test and how are those characteristics handled by the simulator [82], [90], [11]. For instance, the hospital disinfection use case of Locomotec can be simulated in Gazebo and Unity, as both have the models necessary for simulating a wheeled robot with all the sensors of the real robot. At the time of writing, Locomotec uses both simulators for performing tests, but has chosen Unity as the main simulator since it offers better performance with multiple robots than Gazebo.

One challenge of using simulation is the lack of tailorability of the models required by the simulators. Every simulator will support different formats and models, so the effort required to set up a simulation can vary greatly depending on whether a model is available. Since every simulator supports different model formats, some models will not be easily reusable. Some models will require a transformation or a parser, but if this is unavailable then the models are locked to a single simulator software. This is not ideal, as high-precision modelling requires high amounts of effort.

[17] https://www.ogre3d.org/
[18] https://www.blender.org/
[19] https://o3de.org/

| Simulator | RSF supported | OS supported | License | Latest update | Robot platforms |
|---|---|---|---|---|---|
| *Gazebo* | ROS, Player, Yarp* | Windows, Mac, Linux (Ubunut, Debian) | Open source | 2019 | wheeled robots, legged robots, UAVs, Arms |
| *WeBots* | ROS, TCP/IP | Windows, Mac, Linux (Ubuntu) | Open source | 2020 | wheeled robots, legged robots, UAVs, AUVs, Aerospace vehicles, Arms |
| *MORSE* | ROS, YARP, socket-based protocol, and others | Windows†, Mac†, Linux | Open source | 2019 | wheeled robots, UAVs, AUVs, Arms |
| *CoppeliaSim* | ROS | Windows, Mac, Linux (Ubuntu) | Close source | 2021 | wheeled robots, legged robots, Arms |
| *Unity* | ROS* | Windows, Mac, Linux (Ubuntu) | Close source | 2021 | wheeled robots |

Table 5: Summary of support for operative systems (OS), Robot Software Frameworks (RSF), and robotic platforms offered by simulators. (*) Supported through library or plugin. (†) Not officially supported, not all capabilities available.

A comparison of commercially available simulators is presented in Table 5, and some of these simulators are illustrated in Figure 6. However, the overview provided is incomplete, as it does not take into account the abstractions available in the simulators. The *robot software framework (RSF)* and *OS supported* columns give an idea of how many developers are supported. It is also useful to consider the type of *license* used, *latest update*, and *robot platforms* supported as those aspects are relevant for deciding which simulator to use.

## 5.2 Variability in Scenario Execution

One of the challenges of testing, specifically with simulators, is the variation present at different dimensions of the testing process. For instance, there is variability on the objective of the scenario, which can go from a functionality test of a single component to the validation of the entire system. In addition, there is variation on the acceptance criteria used to measure the results, both in terms of dimensions (time, distance, correctness) and fulfillment criteria (boolean, percentages). Moreover, there can also be variability in the scenario, as it is a useful way to discover software faults and make sure the system is performing as expected in general and not in only one scenario [49]. Finally, there is variation on the functionalities of the simulator software.

Choosing a simulator for a scenario can be complex when taking into account the variability of the simulators, especially if a developer is trying to optimize the execution. That is, obtaining the most information on the behaviour of the system with the least number of simulation runs. The factors that make this decision difficult include:

- *The available abstractions*: The system-under-study imposes constraints on the simulator used to execute the scenario. A MRS consisting of drones in a hostile environment has simulation requirements different from a MRS of mobile robots in a hospital. Between these two applications there are requirements

regarding simulating hardware components such as specific sensors and actuators, and environmental conditions relevant for the application.

- *The objective of the scenario*: Simulators can be used for a variety of tests with different goals in mind. Depending on the goal, certain characteristics are required in a simulator: For functionality testing all the relevant hardware and software components should be able to be simulated within a degree of accuracy; for regression testing the simulator should be able to run headless and be configurable through an API; or for stress testing the simulator should be stable.
- *The level of abstraction*: The realism of a simulation is determined by the abstractions of the simulator models. Motion and other phenomena can be modeled by abstracting away aspects that influence its behaviour. Complex models that include many aspects can generate realistic behaviour at the cost of performance. No model can or should include all the aspects of a behaviour, but rather abstract away aspects in accordance with the goals of the simulator [15].
- *The level of granularity*: The abstractions used by the simulator can have different effects on the performance depending on how detailed they are [19]. More detail results in higher fidelity, but less efficiency, and vice versa. For instance, collisions between complex solids are often approximated using primitive shapes such as cuboids, cylinders, and spheres. When these calculations are performed with the meshes of the solids they are more accurate at the expense of the time required to compute it.
- *The acceptance criteria*: The metric used to evaluate a specific scenario has to be measurable by the simulator. In some cases, the measurement of a metric is not natively supported, and so the developer must use either a community-available plugin or write their own in order to extend the functionality of the simulator and measure the metric.
- *The simulation technique*: Simulation techniques such as hardware-in-the-loop, software-in-the-loop, human-in-the-loop, and co-simulation are often used in the testing process, and can be a deciding factor for developers.

As mentioned before, simulators offer a variety of functionalities to the users. With so many functionalities, how can a developer determine if a certain simulator is the best for a certain scenario? One way to do so is to look into the capabilities of the simulator. Simulator capabilities can be seen as a combination of the presence of a simulator model, the level of abstraction of the model, and the requirements of the application. This definition hints that there exist functional and non-functional aspects of the models that determine whether a simulator is able to execute a scenario, even though the line between the two can be blurred. To illustrate this point, consider the force of buoyancy, relevant for underwater robot applications. A simplistic model of buoyancy can be composed of a spring and a center of buoyancy, resulting in an efficient approximation [77]. Here, the presence of a model for this force is the functional aspect, and the accuracy and efficiency are non-functional. For a game engine this model is capable of producing the desired results. However, a simulator of a robotic underwater application requires a more detailed model to obtain trustworthy results. In this case, the requirements of the application regarding accuracy determine that the model is not capable of simulating buoyancy, even though the model is present in the simulation.

## 5.3 Simulator Models

A simulator is able to run a scenario when it uses appropriate models to simulate the behaviour and interactions of the real-life counterpart object or phenomenon. Defining the appropriate model is complicated. On the one hand, a very granular model might enable very accurate results; but on the other hand these intricate models are more computationally expensive, reducing the number of executions that can be completed in a certain time-frame, thus making simulation less appealing. Additionally, there is the question of whether a model has enough granularity to produce trustworthy results. The main sources for errors in simulation are numerical integration errors and model errors. The former relates to the errors obtained from the time steps selected, related parameters, and the integration method used to update the state of the system; whereas the latter relates to incorrect or incomplete assumptions in the mathematical models [37].

It should be noted that the *simulator models* are different from 3D models of structures or URDF files, the former are used to enable the simulator to recreate the behaviour of the real world and present the results [15], whereas the latter are used to describe the concrete system to be simulated. While the two are needed to create a test in simulation, the study will focus on the *simulator models* as these determine what can be simulated and what can be modeled. Models such as URDF and SDF are compositions that conform to the simulator models and can be composed of several model domains. For instance, the URDF description[20] presented in Listing 5 illustrates some of the models required to represent a robot in simulation.

```
<robot>
    <link name="base">
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="0.5 0.1 0.5"/>
            </geometry>
            <material name="white">
                <color rgba="1 1 1 1"/>
            </material>
        </visual>
        <collision>
            <geometry>
                <box size="0.5 0.1 0.5"/>
            </geometry>
            <contact_coefficients mu="0.3" kp="0.4" kd="0.0"/>
        </collision>
        <inertial>
            <mass value="5"/>
            <inertia ixx="0.1" ixy="0" ixz="0" iyy="0.2" iyz="0" izz="0.1"/>
        </inertial>
    </link>

    <link name="link" />

    <joint name="joint" type="revolute">
        <parent link="base"/>
        <child link="link"/>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <limit lower="-2.95" upper="2.95" />
        <dynamics damping="0.0" friction="0.0"/>
    </joint>
</robot>
```

Listing 5: URDF description for a simple mechanism that showcases different model domains

In the URDF file (Listing 5), a `<robot>` is defined by `<link>` tags connected by `<joint>` tags that constraint the link's movements. Each link is defined by `<visual>` and `<collision>` information, which require geometric information defined through the `<geometry>` tag. In addition, inside these tags related properties such as the material and contact coefficients can be specified. In the same way inertial properties such as the mass and the inertia tensor for the links can be defined with the `<inertial>` tag. This specification is used during the simulation, where the geometric, inertial, and constraint information is used by the physics engine to recreate the physical interactions of the real world.

Another format for describing robots is the simulation description format (SDF) [21]. Similarly to URDF, with the SDF format the geometric, visual, collision, inertial, and dynamic information can be specified. In addition,

[20] http://wiki.ros.org/urdf/XML/
[21] http://sdformat.org/

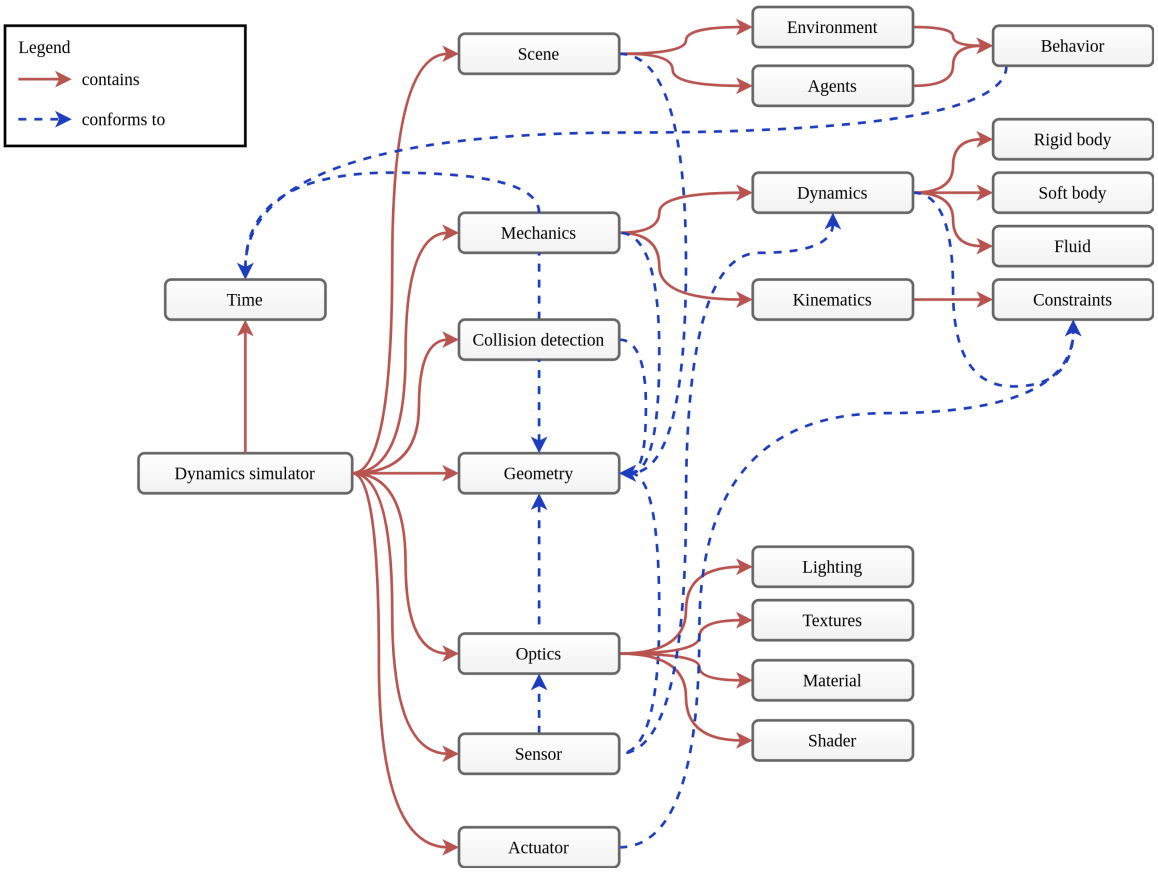



Figure 7: Common models present in dynamic simulator software and their relationships.

the format enables defining information on the environment and external agents. This description is richer than the URDF file, which concerns the robot properties exclusively. Both of the meta-models make assumptions of the simulator models that they have to conform to, as they are intended to be general. In other words, there can be aspects that need to be defined for a scenario that are not available among the concepts of the meta-models.

From this example it can be asserted that simulation involves simulator models from multiple domains, and that these models have relationships between them. An overview of the most common simulator models and their relationships can be observed in Figure 7. In the diagram, a *containment* relationship conveys that a model is composed of another model. For instance, the *optics* model is composed of models for *lighting*, *textures*, *material*, and *shader*. On the other hand, a *conforms to* relation indicates that a model conforms to any constraint imposed by another model. For example, the *constraint* model determines which movement constraints (such as joints) need to be considered in the *dynamics* model, and which *actuators* can be modeled.

Simulator models can vary in what is abstracted away to create the model, where the requirements of an application determines the level of abstraction. For a specific application the selection or design of a model is straight-forward, as the requirements are known. For a simulator this is different, as the simulator developers do not know the requirements of the simulation users, hence the developers have to decide the level of abstraction of the models based on target applications and the users determine if the fidelity of the model suits them. This poses another challenge of tailorability as one of the goals of SESAME is to make scenarios reusable between projects, but due to the variability in the models a scenario could be executable only in a group of simulators, which limits the reusability.

To determine if a simulator is capable of simulating a scenario, we must look into the simulator models used by the simulation software, and the requirements of the scenario imposed by the system, environment, agents, and acceptance criteria. Obtaining these models requires a domain analysis on current dynamic simulators. This

study can be expanded to include robotic simulators, hardware simulation, and control simulation software. For the analysis many dimensions have to be considered, and initially can include:

- *Mechanics*: The reproduction of physical phenomena is a fundamental part of simulation of robotic systems. Simulators need to perform calculations to reproduce behaviour that conforms to classical mechanics (kinematics and dynamics), including rigid-body dynamics, soft-body dynamics, and fluid dynamics.
- *Collision detection*: Although it is related to the mechanics domain, collision detection is often a separate engine. Its purpose is to determine the instant two or more bodies in the simulation make contact, and determine relevant information to simulate the reaction of the bodies.
- *Geometry*: Defines how objects are mathematically represented in space, including their pose (position and rotation), volume, and shape.
- *Optics*: Models in the domain are often used just for visualisation purposes, enabling the stakeholders to view the functionality of the system. Lighting, textures, and materials are common models available by the visualisation engines that can be composed into the scene. Additionally, shaders are models which are used to provide instructions on how to render the image. Together, these models influence the realism of the rendered scene, which can affect the performance of simulated perception tasks.
- *Sensors*: Every simulator offers a variety of sensors to compose the hardware platform. Their accuracy is determined by the logic behind the simulation, the mechanics, and optics. Common simulated sensors include cameras, laser sensors, force and torque sensors, and odometers. Simulators may also include noise models for these sensors, with the goal of recreating the imperfect readings of the real world.
- *Actuators*: Common actuators are also offered by simulators; however, these depend almost exclusively on the physics engines, as these engines provide the different types of constraints that are used to compose actuators.
- *Scene*: The environment and external agents compose the scene of a scenario, and they play a significant role in the performance of the robotic system. Not every element in the real world needs to be simulated, just the ones that can affect the performance of the robot system application. For instance, wind speeds and weather are necessary to test the controller of a UAV, but are not needed to test the control system of mobile robots in a hospital. Elements that are relevant can be not only those that can be perceived by the system, but also elements that are not usually measured but can still affect the system, such as temperature, pressure, or gravity. Another important factor to study is whether these elements remain static throughout the run, or whether they can change dynamically.
- *Communication*: Communication is a fundamental element in MRS, as the different subsystems require robust communication infrastructure to collaborate. As MRS are deployed in dynamic environments where the quality of communication channels can vary, it is of interest to simulate these conditions and observe the behaviour of the system and validate any fallback methods. Additionally, assumptions on the required data bandwidth for the application might not hold. Finally, there is also interest in simulating the physical communication channel and its interactions with the environment and actors.

Previous studies have focused mostly on the performance of the physics engines [37], [11]; a combination of features and performance metrics [90]; or focusing exclusively on the documented features of a few simulators [74]. The focus of this study would be the simulator models, as the goal is to identify the abstractions required to simulate an application. The performance in terms of time efficiency is left to the test designer to investigate and consider in their choice of simulator. From the analysis, a model detailing the capabilities of different simulators can be created.

## 5.4 Scenario Execution

Every stakeholder in the development process has a different perspective of the system, which is determined by their development roles. These perspectives govern the objectives, the acceptance criteria, and the scope

of the test. While the overall scenario can be developed and used by many roles, each role will have their distinctive focus. For instance, in the hospital disinfection use case, each role will concentrate on testing a different characteristic of the system:

- *Behaviour developers* will focus on tests that verify their task-plots, and that the correct behaviour is performed. They can test behaviour such as navigating room-to-room while avoiding obstacles.
- *Function developers* concentrate on the correct functionality of their functions. They can verify their functions using a variety of testing techniques going from unit testing to field testing. Even if the complexity of the test grows, their focus remains in the functions. They can focus on the algorithm for wall following for the disinfection task.
- *System architects* use testing to validate their designs, as they are responsible for providing a solution to the problem that meets all quality assurance criteria.
- *System builders* verify the functionality of the entire system and test the correct integration of all the components.
- *Performance designers* test different configuration of parameters to select the values that maximize performance measures.
- *Safety engineers* certify that the platform and its behavior comply with all the safety standards. These tests can focus on the high risk functionality or their mitigation strategies.

The different stakeholders can use the executable scenarios workbench to describe different scenarios according to their views and concerns. The scenarios described can be suitable for several stakeholders, allowing their reuse through the project, or a given scenario can focus on one concern of one stakeholder. After all the models are composed, the toolchain can generate software artefacts to facilitate the systematic testing of MRS in the real and simulated world. The generation of artefacts can be achieved through separate model-to-model and model-to-text transformations. For example, to assist the testing in simulation for the Locomotec use case several transformations can be used, such as the generation of a launch file for Gazebo with the world and task set up; or the generation of a hospital floor plan 3D model in STL format using procedural generation. The former is a *simulator-specific* transformation, as the artefact can only be read by Gazebo. Whereas the latter is a *simulator-independent* artefact as it can be used by several simulators as long as they read the STL format. One important remark is that the transformation of the environment specification to a concrete 3D model ready for simulation is a multi-step process, where a generator uses the specification to create general descriptions of environments that can later be transformed into a format that is readable by a simulator software.

The capability models designed from the study described in Section 5.3 could be used in several ways to extend the functionality of the tooling beyond the generation of artefacts. They can be used to determine if a specific simulator has the abstractions necessary to execute a test. For example, if a simulator package does not have models to simulate fluid dynamics, then this simulator might not be adequate to test AUVs systems.

# 6 Conclusions

The purpose of this deliverable is to motivate the need for establishing a scenario-based development methodology for MRS and to report about the research performed in the first 8 months of the project.

We summarize the major results in the following list:

- We assessed some of the SESAME use cases from the scientific challenges perspective and exemplified the need for an Executable Scenario concept and methodology to systematically deal with these challenges.
- We assessed related work in the domain of robot software engineering and recapitulate the role of different stakeholder in any robot software development process.
- We investigated actual MRS development practices for one of our use case partner and derived desirable features of the ExSce concept, methodology and tooling.
- We proposed the ExSce concept and methodology and discussed the relevant scientific concepts, ExSce design dimensions, concepts and aspects originating from the meta-modelling, robotics and development process domain.
- We derived an initial formalization of the Executable Scenario, which will be continued in Task 3.2 and Task 3.3.
- We discussed the idea to exploit simulators as a way to execute scenarios, and discussed challenges associated with the enormous variability present in simulators for using them as a tool to execute scenarios.

# References


[1] Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. ISO/IEC 25010:2011, International Organization for Standardization (ISO), March 2011.

[2] System life cycle processes. ISO/IEC/IEEE Std 15288:2015, International Organization for Standardization, May 2015.

[3] Tesnim Abdellatif, Saddek Bensalem, Jacques Combaz, Lavindra De Silva, and Felix Ingrand. Rigorous design of robot software: A formal component-based approach. *Robotics and Autonomous Systems*, 60(12):1563–1578, 2012.

[4] Sorin Adam, Morten Larsen, Kjeld Jensen, and Ulrik Pagh Schultz. Towards rule-based dynamic safety monitoring for mobile robots. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 207–218. Springer, 2014.

[5] Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. A Study on Challenges of Testing Robotic Systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 96–107. IEEE, oct 2020.

[6] Afsoon Afzal, Deborah S. Katz, Claire Le Goues, and Christopher S. Timperley. Simulation for Robotics Test Automation: Developer Perspectives. January 2021.

[7] Amjed Al-Mousa, Belal H Sababha, Nailah Al-Madi, Amro Barghouthi, and Remah Younisse. UTSim: A framework and simulator for UAV air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*, 16(5):172988141987093, September 2019.

[8] Stefano Di Alesio and Sagar Sen. Using uml/marte to support performance tuning and stress testing in real-time systems. *Softw. Syst. Model.*, 17(2):479–508, 2018.

[9] Diego Alonso, Cristina Vicente-Chicote, Francisco Ortiz, Juan Pastor, and Barbara Alvarez. V3cmm: A 3-view component meta-model for model-driven robotic software development. *Journal of Software Engineering for Robotics*, 1(1):3–17, 2010.

[10] Monica Anderson, Jason Bowman, and Paul Kilgo. RDIS: Generalizing Domain Concepts to Specify Device to Framework Mappings. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1836–1841. IEEE, 2012.

[11] Angel Ayala, Francisco Cruz, Diego Campos, Rodrigo Rubio, Bruno Fernandes, and Richard Dazeley. A Comparison of Humanoid Robot Simulators: A Quantitative Approach. In *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 1–6, October 2020.

[12] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.

[13] Geoffrey Biggs and Bruce A MacDonald. Specifying robot reactivity in procedural languages. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3735–3740. IEEE, 2006.

[14] Geoffrey Biggs and Bruce A MacDonald. Evaluating a reactive semantics for robotics. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1854–1859. IEEE, 2008.

[15] Louis G. Birta and Gilbert Arbez. *Modelling and simulation: exploring dynamic system behaviour*. Springer, 2007.

[16] Sebastian Blumenthal and Herman Bruyninckx. Towards a Domain Specific Language for a Scene Graph based Robotic World Model. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2013.

[17] Yury Brodskiy, Robert Wilterdink, Stefano Stramigioli, and Jan Broenink. Fault avoidance in development of robot motion-control software by modeling the computation. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 158–169, Cham, 2014. Springer International Publishing.

[18] Davide Brugali and Nico Hochgeschwender. Managing the functional variability of robotic perception systems. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 277–283, 2017.

[19] Herman Bruyninckx. *Building blocks for the Design of Complicated Systems featuring Situational Awareness*. Sep 2021.

[20] Herman Bruyninckx and Joris De Schutter. Specification of force-controlled actions in the "task frame formalism": A synthesis. *IEEE Transactions on Robotics and Automation*, 12:581–589, 1999.

[21] Herman Bruyninckx, Peter Soetens, and Bob Koninckx. The real-time motion control core of the Orocos project. In *IEEE International Conference on Robotics and Automation*, pages 2766–2771, 2003.

[22] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2004.

[23] Sachin Chitta, Kaijen Hsiao, Gil Jones, Ioan Sucan, and John Hsu. Unified Robot Description Format (URDF), 2011. Last accessed: 11. Sep. 2022.

[24] Federico Ciccozzi, Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Adopting mde for specifying and executing civilian missions of mobile multi-robot systems. *Journal of IEEE Access*, 2(1):6451–6466, October 2016.

[25] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004.

[26] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[27] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[28] Xiangtian Dai, Gregory Hager, and John Peterson. Specifying behavior in c++. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 153–160. IEEE, 2002.

[29] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. In Paolo Ciancarini, Alessandro Fantechi, and Roberto Gorrieri, editors, *FMOODS*, volume 139 of *IFIP Conference Proceedings*. Kluwer, 1999.

[30] Neil Dantam, Ayonga Hereid, Aaron D Ames, and Mike Stilman. Correct software synthesis for stable speed-controlled robotic walking. In *Robotics: Science and Systems*, 2013.

[31] Michael De Rosa, Jason Campbell, Padmanabhan Pillai, S Goldstein, Peter Lee, and T Mowry. Distributed watchpoints: Debugging large multi-robot systems. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3723–3729. IEEE, 2007.

[32] E. Denney, G. Pai, and M. Johnson. Towards a Rigorous Basis for Specific Operations Risk Assessment of UAS. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2018.

[33] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160. Springer, 2012.

[34] Sanjay K. Dhurandher, Sudip Misra, Mohammad S. Obaidat, and Sushil Khairwal. UWSim: A Simulator for Underwater Sensor Networks. *SIMULATION*, 84(7):327–338, July 2008.

[35] Benjamin Dittes and Christian Goerick. A language for formal design of embedded intelligence research systems. *Robotics and Autonomous Systems*, 59(3):181–193, 2011.

[36] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: MORSE. In *2011 IEEE International Conference on Robotics and Automation*, pages 46–51, May 2011.

[37] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, May 2015.

[38] Peter Feiler, David Gluch, and John Hudak. The architecture analysis & design language (aadl): An introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

[39] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1988–1993. IEEE, 2010.

[40] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, 2008.

[41] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 63–78, Phoenix AZ USA, June 2019. ACM.

[42] Marco Frigerio, Jonas Buchli, and Darwin G Caldwell. Code generation of algebraic quantities for robot controllers. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2346–2351. IEEE, October 2012.

[43] Marco Frigerio, Jonas Buchli, and Darwin G Caldwell. Model based code generation for kinematics and dynamics computations in robot controllers. In *Workshop on Software Development and Integration in Robotics, St. Paul, Minnesota, USA*, 2012.

[44] Loïc Gammaitoni and Nico Hochgeschwender. Rpsl meets lightning: A model-based approach to design space exploration of robot perception systems. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 75–82, 2016.

[45] Sergio García, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger, and Tomas Bures. High-level mission specification for multiple robots. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*, page 127–140. IEEE, October 2019.

[46] Sergio García, Daniel Strüber, Davide Brugali, Alessandro Di Fava, Philipp Schillinger, Patrizio Pelliccione, and Thorsten Berger. Variability modeling of service robots: Experiences and challenges. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*, VAMOS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[47] Malik Ghallab, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[48] Luca Gherardi and Davide Brugali. Modeling and reusing robotic software architectures: the hyperflex toolchain. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6414–6420. IEEE, 2014.

[49] Florian Hauer, Alexander Pretschner, and Bernd Holzmüller. Re-Using Concrete Test Scenarios Generally Is a Bad Idea. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1305–1310, October 2020.

[50] Nick Hawes, Chris Burbridge, Ferdian Jovan, Lars Kunze, Bruno Lacerda, Lenka Mudrová, Jay Young, Jeremy L. Wyatt, Denise Hebesberger, Tobias Körtner, Rares Ambrus, Nils Bore, John Folkesson, Patric Jensfelt, Lucas Beyer, Alexander Hermans, Bastian Leibe, Aitor Aldoma, Thomas Faulhammer, Michael Zillich, Markus Vincze, Muhannad Al-Omari, Eris Chinellato, Paul Duckworth, Yiannis Gatsoulis, David C. Hogg, Anthony G. Cohn, Christian Dondrup, Jaime Pulido Fentanes, Tomás Krajník, João M. Santos, Tom Duckett, and Marc Hanheide. The STRANDS project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine (RAM)*, 24(3), September 2017.

[51] Frederik Hegger, Nico Hochgeschwender, Gerhard Kraetzschmar, Jan Paulus, Michael Reckhaus, and Azamat Shakhimardanov. Specifications of Architectures, Modules, Modularity, and Interfaces for the BROCRE Software Platform and Robot Control Architecture Workbench. Deliverable d-2.2 (public), the brics project (fp7-ict grant agreement number 231940), Bonn-Rhein-Sieg University of Applied Sciences, Sankt Augustin, Germany, 2010.

[52] Tom Henderson and Esther Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193, 1984.

[53] Nico Hochgeschwender, Sven Schneider, Holger Voos, and Gerhard K. Kraetzschmar. Towards a Robot Perception Specification Language. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2013.

[54] Nico Hochgeschwender, Sven Schneider, Holger Voos, and Gerhard K Kraetzschmar. Declarative specification of robot perception architectures. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 8810 of *Lecture Notes in Computer Science*, pages 291–302. Springer International Publishing, 2014.

[55] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4146–4151. IEEE, 2001.

[56] J.F. Ingles-Romero, A. Lotz, C. Vicente-Chicote, and C. Schlegel. Dealing with Run-Time Variability in Service Robotics : Towards a DSL for Non-Functional Properties. In *Workshop on Domain-Specific Languages and models for Robotic systems*, number iv, 2010.

[57] C. Jayawardena, I. H. Kuo, U. Unger, A. Igic, R. Wong, C. I. Watson, R. Q. Stafford, E. Broadbent, P. Tiwari, J. Warren, J. Sohn, and B. A. MacDonald. Deployment of a service robot to help older people. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5990–5995, 2010.

[58] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[59] M Klotzbucher, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Reusable hybrid force-velocity controlled motion specifications with executable domain specific languages. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4684–4689. IEEE, 2011.

[60] Markus Klotzbücher, Geoffrey Biggs, and Herman Bruyninckx. Pure Coordination using the Coordinator – Configurator Pattern. In *Workshop on Domain-Specific Languages and models for Robotic systems*, volume 231940, 2012.

[61] 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)*, page 6. IEEE, 2004.

[62] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[63] Lars Kunze, Tobias Roehm, and Michael Beetz. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5589–5595. IEEE, 2011.

[64] Charles Lesire, David Doose, and Hugues Cassé. Mauve: a component-based modeling framework for real-time analysis of robotic applications. In *7th full day Workshop on Software Development and Integration in Robotics (ICRA2012-SDIR VII)*, 2012.

[65] Hongzhi Liang, Juergen Dingel, and Zinovy Diskin. A comparative survey of scenario-based to state-based model synthesis approaches. In *Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools*, SCESM '06, page 5–12, New York, NY, USA, 2006. Association for Computing Machinery.

[66] Martin Loetzsch, Max Risler, and Matthias Jungel. Xabsl-a pragmatic approach to behavior engineering. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5124–5129. IEEE, 2006.

[67] Douglas C MacKenzie, Jonathan M Cameron, and Ronald C Arkin. Specification and execution of multiagent missions. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 51–58. IEEE, 1995.

[68] Topi Maenpaa, Antti Tikanmaki, Jukka Riekki, and Juha Roning. A distributed architecture for executing complex tasks with multiple robots. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3449–3455. IEEE, 2004.

[69] Anthony Mallet, Cédric Pasteur, Matthieu Herrb, Séverin Lemaignan, and Félix Ingrand. Genom3: Building middleware-independent robotic components. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4627–4632. IEEE, 2010.

[70] Vikram Manikonda, Perinkulam S Krishnaprasad, and James Hendler. A motion description language and a hybrid architecture for motion planning with nonholonomic robots. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 2, pages 2021–2028. IEEE, 1995.

[71] Matthew T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems Man and Cybernetics*, 11:418–432, 1981.

[72] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[73] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger. Specification patterns for robotic missions. *Transactions on Software Engineering*, 2019.

[74] David S. Michal and Letha Etzkorn. A comparison of Player/Stage/Gazebo and Microsoft Robotics Developer Studio. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 60–66, New York, NY, USA, March 2011. Association for Computing Machinery.

[75] Olivier Michel. Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, March 2004.

[76] A.T. Miller and P.K. Allen. Graspit! A versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122, December 2004.

[77] Ian Millington. *Game physics engine development: how to build a robust commercial-grade physics engine for your game*. Morgan Kaufmann, 2nd edition, 2010.

[78] Luc Moreau, Paul Groth, James Cheney, Timothy Lebo, and Simon Miles. The rationale of PROV. *Journal of Web Semantics*, 35(4):235–257, dec 2013.

[79] Matteo Morelli and Marco Di Natale. Control and scheduling co-design for a simulated quadcopter robot: A model-driven approach. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 49–61, Cham, 2014. Springer International Publishing.

[80] Matteo Morelli and Marco Di Natale. Control and scheduling co-design for a simulated quadcopter robot: A model-driven approach. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 49–61. Springer, 2014.

[81] Hiroyuki Nishiyama, Hayato Ohwada, and Fumio Mizoguchi. Logic specifications for multiple robots based on a current programming language. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 1, pages 286–291. IEEE, 1998.

[82] Farzan M. Noori, David Portugal, Rui P. Rocha, and Micael S. Couceiro. On 3d simulators for multi-robot systems in ROS: MORSE or gazebo? In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 19–24. IEEE, oct 2017.

[83] Arne Nordmann and Sebastian Wrede. A Domain-Specific Language for Rich Motor Skill Architectures. In *Workshop on Domain-Specific Languages and models for Robotic systems*, Tsukuba, 2012.

[84] Arne Nordmann, Sebastian Wrede, and Jochen J. Steil. Modeling of movement control architectures based on motion primitives using domain-specific languages. In *International Conference on Robotics and Automation (ICRA)*, pages 5032–5039. IEEE, 2015.

[85] OMG. OMG Systems Modeling Language (OMG SysML), Version 1.3, 2012.

[86] Francisco J Ortiz, Diego Alonso, Francisca Rosique, Francisco Sánchez-Ledesma, and Juan A Pastor. A component-based meta-model and framework in the model driven toolchain c-forge. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 340–351. Springer, 2014.

[87] Samuel Parra, Sven Schneider, and Nico Hochgeschwender. Specifying QoS requirements and capabilities for component-based robot software. In *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*, pages 29–36. IEEE, 2021.

[88] SESAME Project Partners. Project Requirements. Deliverable d-1.1 (ec distribution), secure and safe multi-robot systems (ict-46-2020-101017258 grant agreement number 101017258), 2021.

[89] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, December 2012.

[90] Lenka Pitonakova, Manuel Giuliani, Anthony Pipe, and Alan Winfield. Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. In Manuel Giuliani, Tareq Assaf, and Maria Elena Giannaccini, editors, *Towards Autonomous Robotic Systems*, Lecture Notes in Computer Science, pages 357–368, Cham, 2018. Springer International Publishing.

[91] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[92] Arun Kumar Ramaswamy, Bruno Monsuez, and Adriana Tapus. Solution space modeling for robotic systems. *Journal for Software Engineering Robotics (JOSER)*, 5(1):89–96, 2014.

[93] Arunkumar Ramaswamy, Bruno Monsuez, and Adriana Tapus. An extensible model-based framework development methodology for robotic systems. In *Journal of Software Engineering for Robotics*, pages 154–163, December 2017.

[94] Andres J Ramirez and Betty HC Cheng. Automatic derivation of utility functions for monitoring software requirements. In *Model Driven Engineering Languages and Systems*, pages 501–516. Springer, 2011.

[95] M. Reckhaus and N. Hochgeschwender. A Platform-Independent Programming Environment for Robot Control. *Workshop on Domain-Specific Languages and models for Robotic systems*, 2010.

[96] Eric Rohmer, Surya P. N. Singh, and Marc Freese. CoppeliaSim (formerly V-REP): A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, nov 2013.

[97] Andrey Rusakov, Jiwon Shin, and Bertrand Meyer. Simple concurrency for robotics with the roboscoop framework. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1563–1569. IEEE, 2014.

[98] Christian Schlegel, Alex Lotz, Matthias Lutz, and Dennis Stampfer. *Composition, Separation of Roles and Model-Driven Approaches as Enabler of a Robotics Software Ecosystem*, pages 53–108. Springer International Publishing, 2021.

[99] Christian Schlegel, Andreas Steck, Davide Brugali, and Alois Knoll. Design abstraction and processes in robotics: From code-driven to model-driven engineering. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 324–335. Springer, 2010.

[100] Sven Schneider, Nico Hochgeschwender, and Gerhard K. Kraetzschmar. Structured design and development of domain-specific languages in robotics. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 8810 of *Lecture Notes in Computer Science*, pages 231–242. Springer International Publishing, 2014.

[101] Enea Scioni, Nico Huebel, Sebastian Blumenthal, Azamat Shakhimardanov, Markus Klotzbücher, Hugo Garcia, and Herman Bruyninckx. Hierarchical Hypergraphs for Knowledge-centric Robot Systems: a Composable Structural Meta Model and its Domain Specific Language NPC4. *Journal of Software Engineering for Robotics (JOSER)*, 7:55–74, 2016.

[102] Russell Smith. Open Dynamics Engine. https://www.ode.org/, 2008.

[103] Peter Soetens and Herman Bruyninckx. Realtime hybrid task-based control for robots and machine tools. In *IEEE International Conference on Robotics and Automation*, pages 260–265, 2005.

[104] Daniel Terhorst-North. What's in a story? Online, 2007. Online. Last accessed: 30. September 2021.

[105] Ulrike Thomas, Gerd Hirzinger, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. A New Skill Based Robot Programming Language Using UML/P Statecharts. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 461–466. IEEE, 2013.

[106] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues. Crashing Simulated Planes is Cheap: Can Simulation Detect Robotics Bugs Early? In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 331–342, April 2018.

[107] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2012.

[108] Steve Tousignant, Eric Van Wyk, and Maria Gini. An overview of xrobots: A hierarchical state machine based language. In *ICRA-2011 Workshop on Software Development and Integration in Robotics, Shanghai, China May*, pages 9–13, 2011.

[109] Steve Tousignant, Eric Van Wyk, and Maria Gini. Xrobots: A flexible language for programming mobile robots based on hierarchical state machines. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1773–1778. IEEE, 2012.

[110] Hans Utz, Gerhard Kraetzschmar, Gerd Mayer, and Günther Palm. Hierarchical behavior organization. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2598–2605. IEEE, 2005.

[111] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, June 2000.

[112] Matt Wynne, Aslak Hellesøy, and Steve Tooke. *The Cucumber Book, Second Edition. Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2017.