SESAME
SECURE AND SAFE MULTI-ROBOT SYSTEMS

Project Number 101017258

D8.9 Integrated Platform – Final Version

Version 1.0
30 December 2023
Final

Public Distribution

ATB

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

PROJECT PARTNER CONTACT INFORMATION

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
PAL Robotics Gizem Bozdemir C/ Pujades 77-79, 7-7 08005 Barcelona Spain E-mail: gizem.bozdemir@pal-robotics.com	**The Open Group** Scott Hansen Rond Point Schuman 6, 7th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org
Technology Transfer Systems Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com	**University of Hull** Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk
University of Luxembourg Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu	**University of York** Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial draft based on deliverable D8.3	28 September 2023
0.2	First inputs collected from contributor partners	3 November 2023
0.3	Document harmonization and inclusion of further inputs	30 November 2023
0.4	Requirements update	8 December 2023
0.5	Final updates integration and harmonisation	19 December 2023
0.9	Final edits and updates version	22 December 2023
1.0	QA version for submission	30 December 2023

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

The deliverable includes the integrated versions of the SESAME solutions, reports on main features and installation/customization guidelines. The report documents the integration of the final versions of the SESAME tools and modules. The report covers the descriptions of the integrated platform architecture with a detailed description of all integrated components. The report also includes the basic workflow of the SESAME tools and components, and the description of the SESAME integrated platform. The continuous integration and deployment process is highlighted as well. The report provides the guide for the installation and configuration of the SESAME components. The deliverable is in line with the project plan for delivering the SESAME technologies, based on the selected Adaptive Project Management approach, the Incremental Integration Strategy (IIS) methodology, and the planned features of the initial (M18, D8.3 [1]) and final versions (M30) of the SESAME components and platform. The deliverable is the updated of the initial version of the SESAME integrated platform that was issued in the deliverable D8.3 at M18. In this document, the requirements that were planned for the SESAME components are analysed, indicating to which extend the requirements have been fulfilled. The modelling and tooling in the project are intended to be robot operating system agnostic and therefore able to support multiple industrial robotics platform.

1. INTRODUCTION

This deliverable documents the SESAME task 8.3, Platform Integration and Evaluation Support. This task aims to integrate the technical contributions of WP2-7 into a unified development studio and deployment platform, and to provide support to the demonstrator use-cases. Our methodology is based on the Incremental Integration Strategy (IIS), prioritising the integration of main components and interfaces. Following this, we continuously integrate software and hardware components through incremental builds, performing integration tests and fixing any identified integration errors, until we produce the integrated SESAME technological solution and integrate it In the use case partners' infrastructures. This task also documents the minimum preparatory activities to be performed for certain use cases in order to install the system and instantiate it for operation and in particular, D8.3 Integrated Platform–- Initial Version [1] documents all this for the initial versions of the SESAME tools and modules, and D8.9 (the current file) documents the final versions of the SESAME tools and modules and of the integrated system. These deliverables (D8.3 and D8.9) strongly relate to the previous deliverable D8.1 [2] in which we have delineated our plan for delivering the SESAME technologies, justifying the reasons underpinning the selection of the Adaptive Project Management approach and detailing the features comprising the initial (M18) and final (M30) version of the SESAME platform. The attention is given to the fulfilment of the requirements that were defined by both the use case partners and RTD partners at the beginning of the project,

1.1 DOCUMENT STRUCTURE

The deliverable consists of the following sections:

- Section 2 describes the integrated platform architecture and describes the final versions of SESAME components and tools. It also includes tables with the requirements, as defined at the beginning of the project in the deliverable D1.1, with assessment of the fulfilment of the requirements and with an analysis of the requirements which are not fully fulfilled.
- Section 3 presents the workflows of the SESAME tools and components.
- Section 4 describes the SESAME integrated platform.
- Section 5 describes the continuous integration and deployment process.
- Section 6 presents a quick start guide for the installation and configuration of the SESAME components.
- Section 7 concludes the report.

1.2 RELATIONSHIP TO OTHER DELIVERABLES

This deliverable provides guidelines and constraints for the tasks of all the technical workpackages of SESAME (i.e., WP2-WP7). Therefore, the contents of this document are relevant to all deliverables of those WPs. Moreover, this document is related to the following deliverables:

- **D1.1 - Project Requirements:** To come up with an architecture that satisfies the needs of the users, we analysed the user and technical requirements of the project.
- **D1.2 - Evaluation Plan:** To come up with an architecture that satisfies the needs of the users, we also analysed the specific use cases that we will use in SESAME to evaluate the technology offering.
- The following deliverables present the complete initial description of each of the SESAME solutions:
  - **D2.3 Collaborative Sensor Fusion**;
  - **D3.2 Executable Scenarios Workbench (Initial Version);**
  - **D3.4 Executable Scenarios Workbench (Final Version);**
  - **D4.2 Safety-Targeted ODE and EDDI specification;**
  - **D4.3 Safety-Security Co-Engineering Framework;**
  - **D4.4 Tools for Automated Safety Analysis of MRS and for Production of EDDIs (Initial Version);**
  - **D4.6 Tools for Automated Safety Analysis of MRS and for Production of EDDIs (Final Version);**
  - **D5.2 Security-Targeted ODE and EDDI Specification;**
  - **D5.3 Tools for Automated Security Analysis of MRS and for Production of EDDIs;**
  - **D5.4 Tailorability of EDDIs;**
  - **D5.6 Tools for Automated Security Analysis of MRS and for Production of EDDIs (Final Version)**
  - **D6.1 Assurance of Data Driven and Learning Components of EDDIs;**
  - **D6.2 Simulation-Based Testing Methodology for EDDIs;**
  - **D6.3 Tools for Automated Quality Assurance of EDDI-Supported MRS (Initial Version);**
  - **D6.6 Multi-staged Quality Assurance Methodology for EDDI-Supported MRS**
  - **D6.7 Tools for Automated Quality Assurance of EDDI-Supported MRS (Final Version);**
  - **D7.1 Runtime Safety and Security Concept - EDDI Runtime Model Specification;**
  - **D7.2 Tools for Generation of Runtime EDDIs;**
- **D8.1 – Architectural Guidelines Report;** Presented the architecture of the project's components.
- **D8.3 - Integrated Platform (Initial Version):** first implementation of the SESAME offerings.
- **D8.4-D8.8 and D8.10-D8.14**: In these deliverables, the proposed architecture and the related implementation are evaluated on the project's use cases.

Additionally, the architecture of the SESAME technology offering is relevant to all the management-related deliverables (WP10) and the dissemination deliverables (WP9).

2. INTEGRATED PLATFORM ARCHITECTURE

This section describes the integrated platform architecture and describes the final versions of SESAME components and tools. The SESAME platform architecture is shown in the following Figure 1.

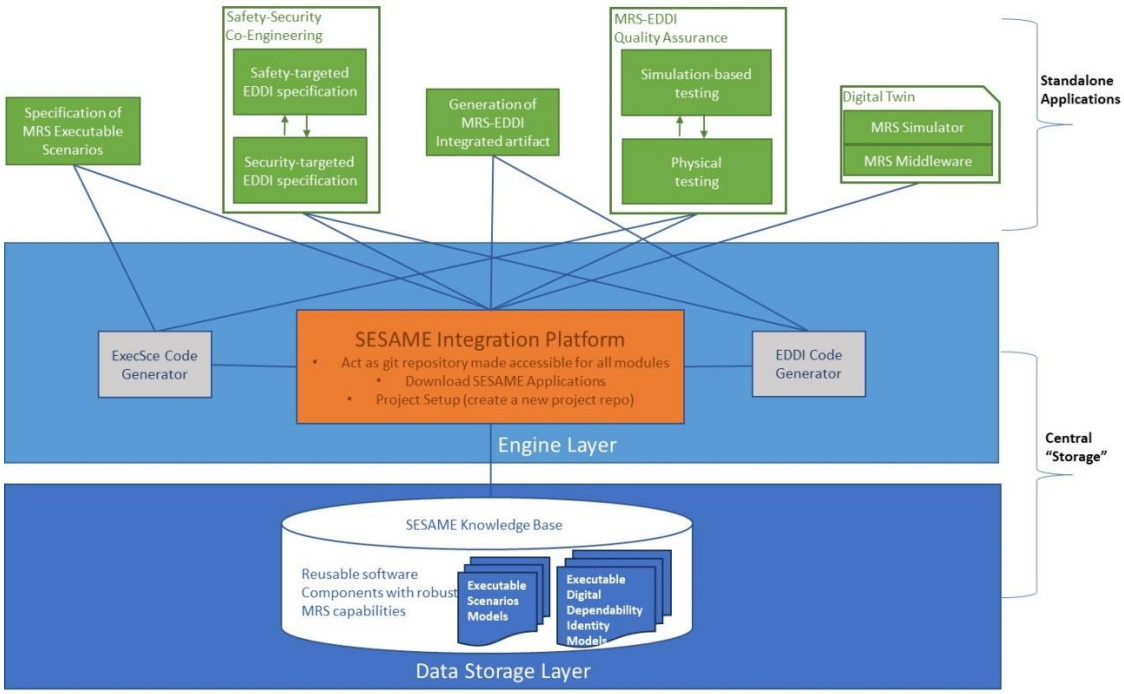


Figure 1: SESAME platform architecture

A detailed description of the SESAME Integration Platform implementation is presented in Section 4 SESAME Integration Platform.

2.1 COMPONENTS OVERVIEW

2.1.1 Collaborative Sensor Fusion

This component is used to provide perception information in front of the drone. It consists of two sub-components:

- **Perception**: The perception sub-component is used to provide information of the surrounding of the drone. This sub-component detects and tracks the target drone in the input images captured by the camera and outputs the relative positions of the detected drone to the sensor fusion sub-component.

- **Sensor fusion**: The sensor fusion sub-component is used to provide estimation and association of information coming from onboard and offboard sensors (e.g.

onboard IMU and offboard position estimation from the camera of another drone) that improves the current state estimation of the target drone.

Table 1: Requirements for Collaborative Perception

Req. No.	Requirement	Priority	Resp. WP	Status
D1	Provides support for collaborative sensor-fusion within an MRS to increase the accuracy of robot localization.	SHALL	2	Done
D2	Sensor fusion capabilities are able to use obstacle detection sensor data to improve localization and representations of MRS environments.	SHALL	2	Done
D3	Sensor-fusion capabilities are able to adapt to sensor variability and availability.	SHOULD	2	Done
D4	Sensor-fusion capabilities are able to more accurately represent the operating environment of the MRS.	SHOULD	2	Done
D5	Sensor fusion capabilities are able to use RGB camera data to improve localization and representations of MRS environments	SHOULD	2	Done
D6	A protocol for data exchange and monitoring with sensors (and other components) having different communication protocols is provided by EDDI Tools.	SHOULD	4, 5	Done
D7	A protocol for data exchange and monitoring with sensors (and other components) having different communication protocols is provided by ExSce Workbench.	SHOULD	3	Done

Table 2: Requirements for Collaborative Intelligence Analytics

Req. No.	Requirement	Priority	Resp. WP	Status
D33	The collaborative intelligence analytics component should conform to the ExSce methodology.	SHALL	2	Done
D34	Provide concepts to express scenario expectations which conform to meta-models developed in the ExSce approach.	SHALL	2	Partial support
D35	Provide means to manually associate scenarios and their expectations with instrumentation requirements.	SHALL	2	Done for ROS-based systems
D36	Provide means to manually instrument scenarios for collecting information relevant for assessing different aspects of collaborative intelligence.	SHALL	2	Done for ROS-based systems
D37	Provide means to persistently store experiences of executed scenarios.	SHOULD	2	Done for ROS-based systems
D38	Provide means to automatically instrument scenarios for collecting information relevant for assessing different aspects of collaborative intelligence.	SHOULD	2	Partially
D39	Provide a runtime capable API to query experiences associated with scenarios.	SHOULD	2	*Done*
D40	Provide methods to reason about deviations between specified scenarios, expectations and experiences.	SHOULD	2	Partially done for experiences related to scenery.
D42	Provide methods to derive explanations why	MAY	2	Partial

Req. No.	Requirement	Priority	Resp. WP	Status
	certain scenarios are deviating from other scenarios.			support
D43	Interface or exchange format to enable embedding of EDDIs for the sake of scenario instrumentation with ExSce Workbench.	SHALL	3	Partial support

2.1.2 Trajectory Planning and Tracking

Trajectory planning is one of the most important capabilities of SESAME to be addressed, i.e., to find an optimal path to the destination of MRS. The trajectory can be defined as "*a time parameterized motion reference, i.e., geometric values of position, heading, derivatives associated with time law, passing through the waypoints*". One of the main objectives is to integrate perception-awareness into MRS trajectory planning, i.e., the construction of the trajectory on which the perception metric is incorporated. On the other hand, one of the main features of the trajectory planner is collision avoidance with other obstacles, humans, and other robots. These, generally, can be titled as static and dynamic obstacles. So, we implement the updated information of the obstacles into the proposed planner. Furthermore, perception-awareness and safety/security are to be considered in the planning. Our main task falls at the planning level that we are going to address the above-mentioned aspects on versatile MRS tasks. Another distinct feature to be implemented is the concept of situational awareness. This may include localization, geometric mapping, semantic mapping, obstacle detection and tracking, etc. Indeed, it can be stated as collaborative and perception-aware trajectory planning, which minimizes environmental uncertainty by using data provided by other robots in a team combined with data collected by the perception components of the robot, provided by Collaborative Perception component.

In addition to the trajectory planning, one significant capability is the trajectory tracking control, i.e., the design of the control commands to make the MRS stable as well as to track the trajectory (planned in the previous step) as close as possible. More importantly, the aspect we address is the structural design of planning and tracking parts, if we address and design them separately or simultaneously, considering the priority. This implies another preliminary step to be taken, as "*the general planning and tracking structural scheme, for the given MRS and use-cases, addressing the parts to be (de)centralized and on/offline*".

Finally, within the framework of autonomous robotic systems, the features of safety and security are vital. However, considering the available literature, the devised approaches are mainly stemmed from the corresponding definitions of safety and security, i.e., the definition implies the corresponding solution. The security can be the resilience of the solution to unauthorized access to communication channels. On the other hand, the safety can be interpreted as tolerance against a family of faults, avoidance from a given area, restriction of MRS states, and safely bound on the tracking drift. Cumulatively, we present the solution as reliable autonomous robotic systems with quality assurance, risk assessment and trust level.

These two components are illustrated in the following Figure 2.

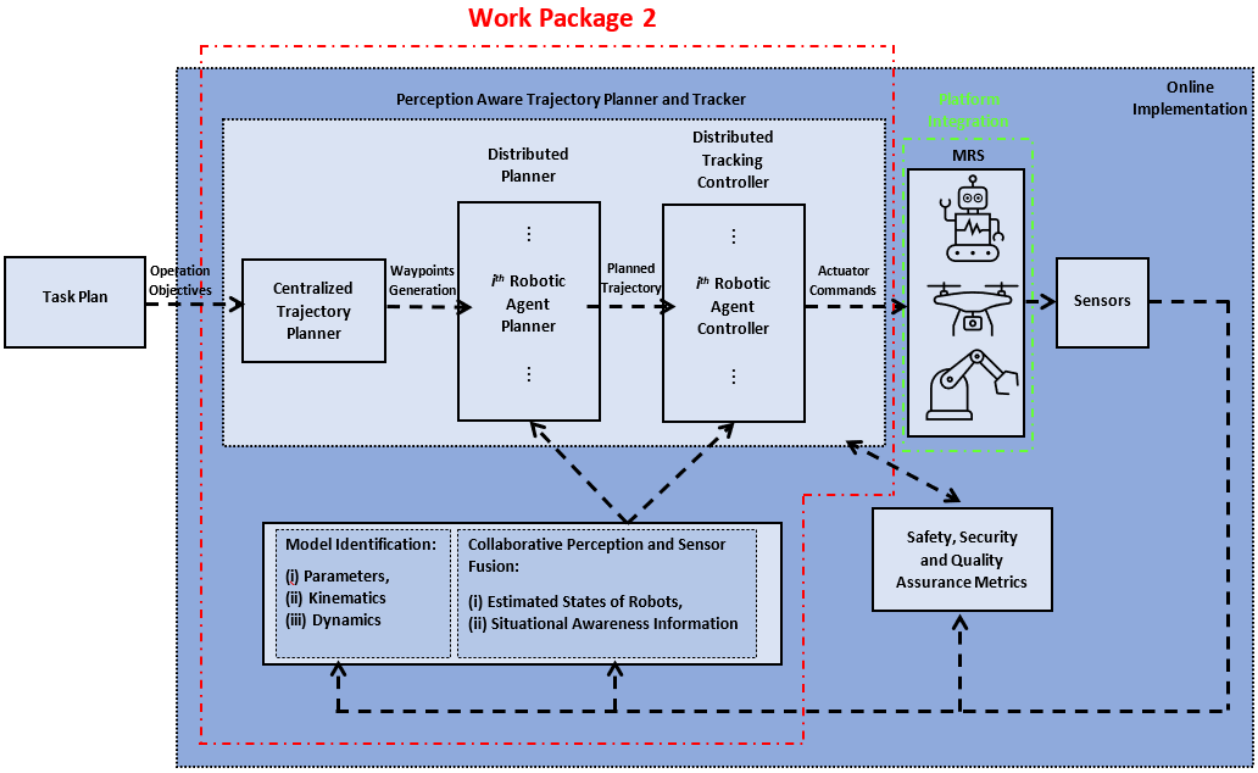


Figure 2: Trajectory Planning and Tracking components overview

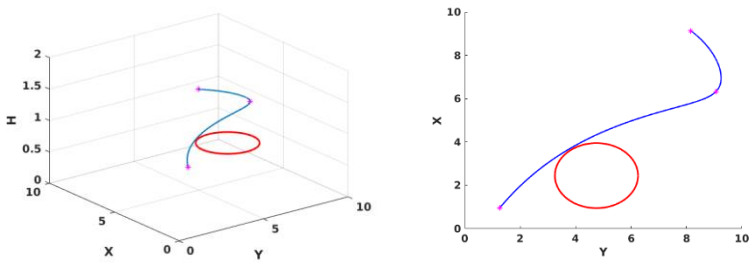


Figure 3: Example of UAV model to pass through some given waypoints

As an example, we have considered a UAV model to pass through some given waypoints, while staying outside of the safety region around the obstacle. In the following figures we have presented the planned trajectory which is followed by the UAV. The red circle presents the safety region and the purple stars are the waypoints.

For this, examples of the optimization times are given below.

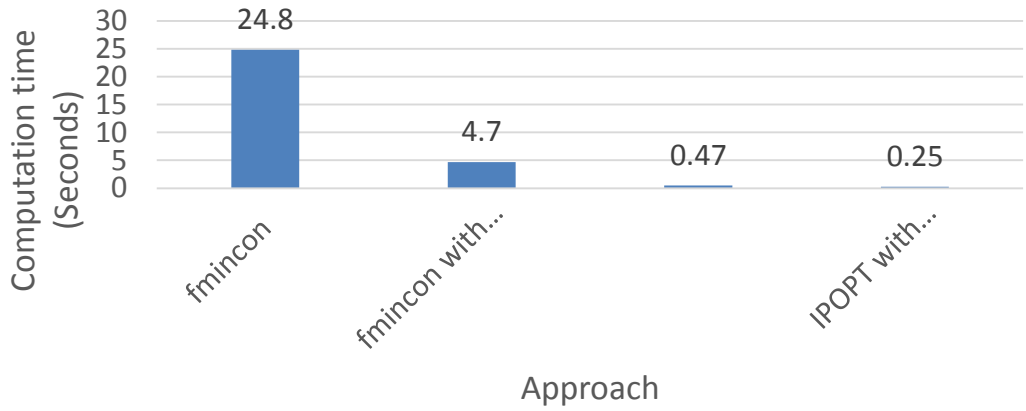


Figure 4: Examples of the optimization times (where the x-axis represents the solver with the specifications)

Moreover, in the following graphics (Figure 5) we have considered two obstacles and we have perturbed their positions (dashed black circles). Then the trajectory is replanned to avoid the new position of the obstacles (green dashed lines). The initial and replanning times are also given.

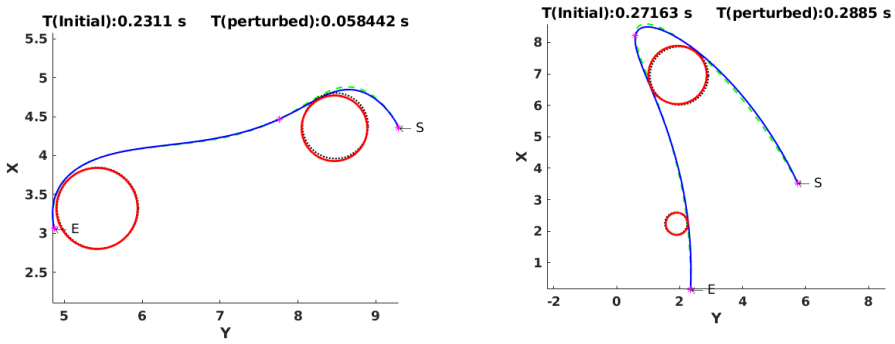


Figure 5: Examples of trajectories with obstacles

Table 3: Requirements status for Trajectory Planning and Tracking

Req. No.	Requirement	Priority	Resp. WP	Status
D8	Provide planned trajectories as motion references (e.g. paths) to achieve the given task plans.	SHALL	2	Done
D9	The planned trajectories achieve the given metrics on safety, security and quality.	SHOULD	2	Done
D10	The planned trajectories are given as time-parameterised motion references.	SHOULD	2	Done
D11	The planned trajectories are collision-free, by taking into consideration the metric-semantic model of the environment (e.g. obstacles).	SHALL	2	Done
D12	The planned trajectories are feasible, by taking into consideration the kinematic and dynamic information of the robotic agents.	SHALL	2	Done
D13	The planned trajectories are perception-aware, by taking into consideration the information of the sensors.	SHALL	2	Done
D14	The planned trajectories are risk-aware, by taking into consideration the safety, security and quality	SHALL	2	Done

Req. No.	Requirement	Priority	Resp. WP	Status
	assurance metrics.			
D15	The component supports multi-robot trajectory planning.	SHOULD	2	Done
D16	The component supports real-time re-planning.	MAY	2	Done
D17	Provide robot commands in the form of actuator/driver commands to the robotics platforms.	SHALL	2	Done
D18	A robot-agnostic interface for the outputted actuator/driver commands to the robotics platforms.	SHALL	2	Done
D19	A robot-agnostic interface is provided for the outputted planned trajectories.	SHOULD	2	Done
D20	ExSce provides detailed information on the robot dynamic and kinematic models and their restrictions.	SHALL	2	Done
D21	ExSce provides detailed information on the list of sensor and their perception models.	SHALL	2	*Done*
D22	Collaborative Perception provides the topologic, semantic (e.g. scene understanding, points of interest, QRs, ...) and geometric map of the environment with fused sensorial information updated in real-time.	SHALL	2	Done
D23	Collaborative Perception provides a real-time estimation of the state (e.g. pose, velocity, etc.) of each robot.	SHALL	2	Done
D24	Collaborative Perception provides sensorial, perception and situational awareness metrics for perception-aware planning, e.g. covariances, fields of view, etc.	SHOULD	2	Partial support *The metrics are linked to the sensor limitations. The experiments have shown these limitations even if only simulation environments were promised at this point.*
D25	ExSce and/or Collaborative Intelligence provide information of the task plans with metrics on safety, security and quality assurance.	SHALL	2	Partial Support
D26	EDDI provide safety, security and quality assurance metrics for a requested planned trajectory.	SHALL	2	Partial Support
D27	Interface to exchange information of the dynamics and kinematics of the robots with ExSce Workbench.	SHALL	3	Done
D28	Interface to exchange information of the sensors of the robots with ExSce Workbench.	SHALL	3	Done
D29	Interface to exchange information of the task plans with ExSce Workbench.	SHALL	3	Done
D30	Interface to exchange information of the task plans with Collaborative Intelligence.	SHALL	2	Done *The relative position estimation is*

Req. No.	Requirement	Priority	Resp. WP	Status
				provided via ROS topic using PoseStamped messages.
D31	Interface to exchange information of the complete situational awareness with Collaborative Perception.	SHALL	2	Done *The position of other drones and objects are provided using ROS topic and PoseStamped messages.*
D32	Versatile interface to request safety, security and quality assurance metrics for a tentative trajectory with EDDI Tools.	SHALL	4, 5	Done

2.1.3 Executable Scenarios Workbench

The Executable Scenario (ExSce) Workbench[1] is a collection of metamodels and tools for supporting stakeholders in carrying out various activities in the ExSce methodology, which includes specification, transformation, execution, assurance, and generalization of MRS scenarios. More details on what these activities may entail can be found on the online definition of the methodology[2]. The tools currently available in the workbench support one or more of these activities for specific aspects of an MRS scenario, namely requirement and acceptance criteria, environment, and the system's kinematics and dynamics. The rest of this section will describe these tools in more details.

```
Scenario: pickup scenario
  Given an object is located on the table
  When the robot starts picking
  Then the object is held by the robot
```

An important aspect of assuring the quality of any system is evaluating whether the system satisfies stakeholders' acceptance criteria (AC). To this end, bdd-dsl[3] provides a domain-specific language (DSL) to specify AC following the Behaviour-Driven Development [4](BDD) approach using the JSON-LD[5] syntax. bdd-dsl allows the specification of generic scenario templates that can be extended with information about the environment, task or agent unique to the specific scenario variants. The models of these scenario variants can then be transformed into Gherkin[6] feature files, which can then be used for automated acceptance testing. A tutorial showing how to specify a BDD template and variant for a simple pickup task such as one shown in the BDD

[1] https://sesame-project.github.io/exsce/exsce-workbench.html
[2] https://sesame-project.github.io/exsce/terminology.html
[3] https://hbrs-sesame.github.io/bdd-dsl/
[4] https://dannorth.net/introducing-bdd/
[5] https://json-ld.org/
[6] https://cucumber.io/docs/gherkin/

example above, as well as to generate Gherkin feature files from these models is available online[7].

We employ the model-driven approach to design a modelling language for describing indoor environments: the `Floor Plan DSL`. The textual language enables developers to specify the environment as a composition of both static elements such as spaces, entryways, windows, as well as dynamic elements such as revolving doors. The DSL enables developers to transform the descriptions into a composable representation as an interchange format to communicate with other tools, including those in the ExSce workbench. This paves the way to transform the interchange representation into 3D meshes and occupancy grid maps, which can be used to simulate robot navigation tasks in most robot simulators (see Figure 6). A tutorial showing how to create floor plan models as well as how to transform them into various artefacts can be found on the tool's GitHub repository[8]. In Task 3.3 we will extend the tool through means to model points-of-interest inside the environment as well as the possibility to create variations of the floor plans.

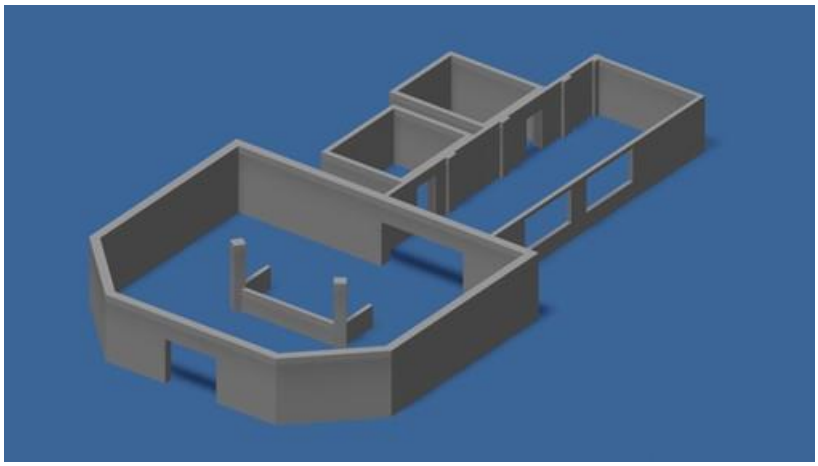


Figure 6: 3D meshes and occupancy grid maps examples

[7] https://hbrs-sesame.github.io/bdd-dsl/bdd-tutorial-feature-gen.html
[8] https://github.com/sesame-project/FloorPlan-DSL/blob/main/docs/Tutorial.md

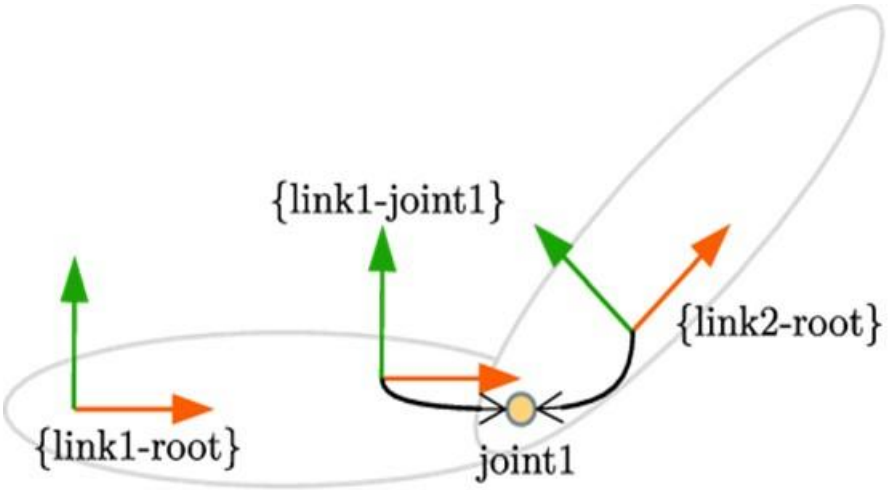


Figure 7: Example of a minimal chain

Finally, the ExSce workbench also includes the kindyngen[9] toolchain, which consists of a metamodel for specifying composable models of kinematic chains, the tools to synthesize correct-by-construction kinematics or dynamics solver algorithms given a kinematic chain specification, and a code generator to transform the synthesized algorithms into executable code. The following JSON-LD excerpt exemplifies how the simple kinematic chain shown in Figure 8 can be realized using the kinematic chain metamodel in kindyngen, while Figure 9 depicts a graphical representation of this specification. Tutorials on how to use the toolchain for configuring the solvers[10], synthesizing solver algorithms and generating corresponding implmentations[11], as well as applying the solver in a control problem[12] are available on the project GitHub repository.

```
{
  "@context": [
    "/ontology/kinematic-chain/structural-entities.json",
    {
      "rob": "http://example.org/my-robot#"
    }
  ],
  "@id": "rob:chain",
  "@graph": [
    {
      "@id": "rob:joint1",
      "@type": [ "Joint", "RevoluteJoint" ],
      "between-attachments": [
        "rob:link1-joint1", "rob:link2-root"
      ],
      "common-axis": "rob:joint1-common-axis",
      "origin-offset": "rob:joint1-offset"
    },
    {
      "@id": "rob:kin-chain1",
      "@type": "KinematicChain",
```

[9] https://github.com/hbrs-sesame/kindyngen
[10] https://github.com/hbrs-sesame/kindyngen/blob/main/docs/tutorial_solver_configuration.md
[11] https://github.com/hbrs-sesame/kindyngen/blob/main/docs/tutorial_solver_robif.md
[12] https://github.com/hbrs-sesame/kindyngen/blob/main/docs/tutorial_controller.md

```
      "joints": [ "rob:joint1" ]
    }
  ]
}
```

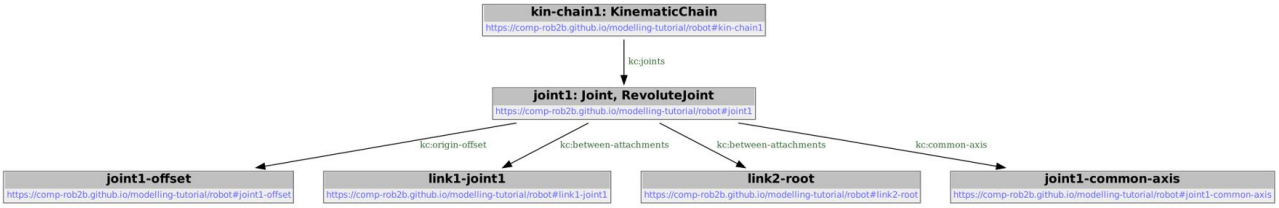


Figure 8: Graphical representation of the textual model.

Table 4: Requirements status for Executable Scenarios Workbench

Req. No.	Requirement	Priority	Resp. WP	Status
D44	Support a scenario-based development approach enhancing both validation and verification activities.	SHALL	3	Done
D45	Extend and improve RobMoSys meta-models and composition structures to support scenario-based development of MRS.	SHALL	3	Done
D46	Define an interface or protocol to exchange ExSce with other components.	SHALL	3	Supported by the composable model structures in JSON-LD format
D47	Provide means to specify selected MRS capabilities by extending RobMoSys meta-models of robotic algorithms.	SHALL	3	Done, supported by the kindyngen tool chain for kinematics and dynamics algorithms
D48	Provide means to specify scenarios on different levels of abstraction by composing mission-relevant and mission-plausible information.	SHALL	3	Done, Supported by ExSce Workbench for modelling varying scenario features
D49	Provide means to store and query reusable and composable scenario models.	SHOULD	3	Done, Supported by *SPARQL queries of scenario models and their executions represented in W3C provenance standard, as well as the*

Req. No.	Requirement	Priority	Resp. WP	Status
				Repository of ExSce
D50	Provide a composable capability model of selected robotic simulators for the sake of integration in the scenario-based methodology and for executing, validating and verifying scenarios.	SHALL	3	Done, *through configurable model transformation of floor plan models*
D51	Provide means to transform scenario models into both simulation and deployment models.	SHALL	3	Done, *through configurable model transformation of floor plan and BDD models*
D55	Provide means to attach expected observable outcomes to scenarios such that scenarios can be validated and verified.	SHALL	3	Done, through fluent concepts in BDD scenario models
D57	Provide means to derive scenario templates from executed scenarios.	SHOULD	3	*Done, through querying mechanisms available in the ExSce Management*
D58	Provide scenario templates as reusable models for specifying future scenarios.	MAY	3	*Partially done through ExSce Management*
D59	Provide explanations why scenarios are succeeding or failing.	MAY	3	Partially done through visualizing outcomes of scenarios in ExSce Management
D60	Interface or exchange format to allow information to be incorporated in the scenario-based approach with EDDI Tools.	SHALL	4, 5	Done, through generated floor plan artefacts used in integration platform
D61	Interface or exchange format to allow information from EDDI Security Analysis Tools and models to be incorporated in the scenario-based approach.	SHALL	5	Done through ExSce Management
D62	Scenario instrumentation means developed for Collaborative Intelligence need to be composable in the scenario-based approach.	SHALL	2	Done, floor plan models are exploited in Collaborative Intelligence
D63	Deployment plans of COSENS MRS capabilities need to be composable in the scenario-based approach with Collaborative Perception.	SHALL	2	Done

2.1.4 EDDI-based Safety Analysis Tools

The Executable Digital Dependability Identity — or EDDI — is a composable model-based artefact that contains dependability information about a system. Although they can serve as purely design-time artefacts, they are also intended to be executed at runtime onboard or alongside their target system to perform dynamic dependability management. An EDDI is therefore both an offline knowledge base storing dependability information about a system and, in its executable form, an online monitor that observes and manages its target system's safety and security.

EDDIs are based on the Open Dependability Exchange (ODE) metamodel, as defined in **D4.2/D5.2** [3] **— Safety/Security ODE and EDDI Specification**. The ODE is intended to be the common interchange format between the different EDDI-related tools, so that common models can be created and used to generate or interact with runtime EDDIs regardless of the original design-time tool.

Several different safety analysis tools were targeted for EDDI support, including safeTbox, BayesFusion GeNIe Modeler, HiP-HOPS, and Dymodia, while additional supporting tools were developed (ODE Tool Adapter & EDDI Editor/Model Converter). These tools allow the creation or import of architectural models of a system which can then be annotated with failure data to record the failure behaviour of the system. This data can then be analysed to produce safety analysis artefacts such as fault trees and FMEAs. More information about each tool can be found in D4.6 — Tools for Automated Safety Analysis of EDDIs [4] , but a brief summary is provided below.

2.1.4.1 SafeTbox

SafeTbox[13] is a commercial model-based safety engineering tool, implemented as an add-in to the Enterprise Architect[14] modelling tool. The tool supports architectural modelling (using a variant of the UML profile supported in Enterprise Architect), Hazard Analysis and Risk Assessment (HARA) using a table-based approach, Failure Analysis using Component Fault Trees [5], Safety Argumentation modelling using the Goal Structuring Notation [6], and Conditional Safety Certificates (ConSerts) [7]. Figure 9 depicts an example of architecture modelling in safeTbox.

[13] https://safetbox.de/
[14] https://sparxsystems.com/

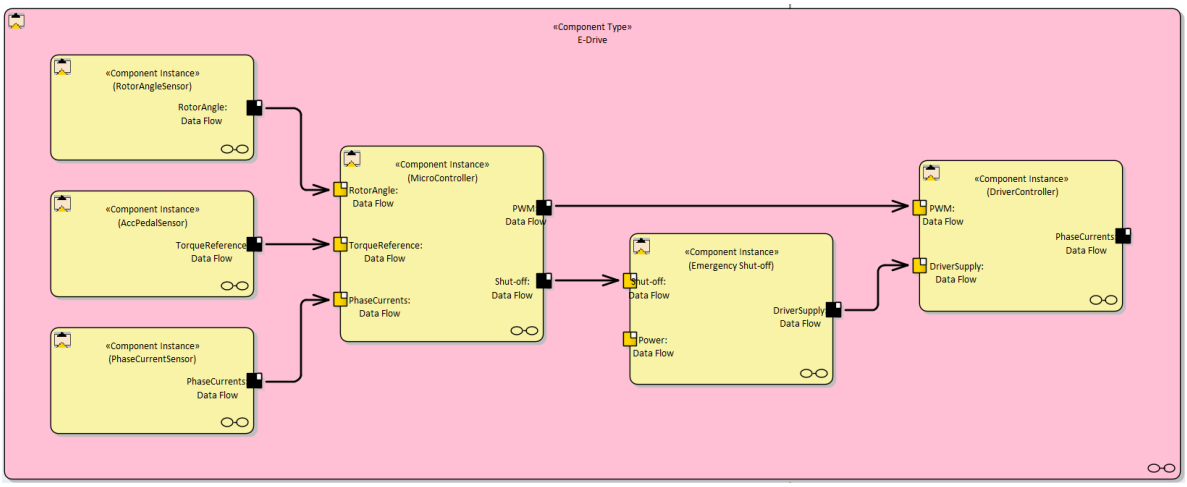


Figure 9: Example of safeTbox architecture modeling

Using safeTbox, it is possible to export models (and related artefacts) of the above features in the form of EDDIs. To achieve this, a tool adapter, originally implemented in the DEIS research project [8], has been extended to support the updated ODE (see D4.2 [3]). The tool adapter facilitates interoperability with other tools, which can be extended to receive the EDDIs as input.

An overview of how tool interoperability is supported can be seen in Figure 10, where a modelling tool (e.g. safeTbox) uses the Tool Adapter interface to import, export, or execute Epsilon scripts[15] on the subject EDDIs.

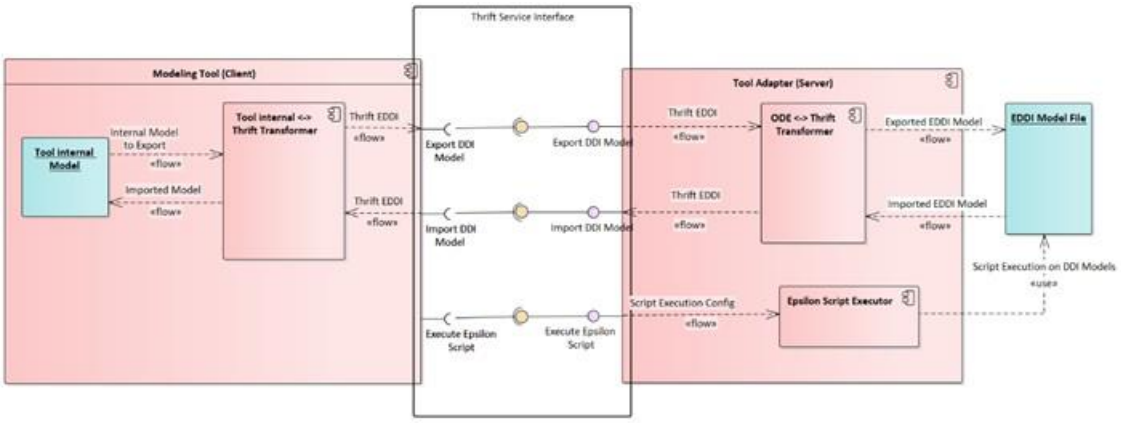


Figure 10: Overview of Tool Adapter

The generated EDDIs are in an XML-based format (.ecore format of the EMF[16]). More details on how tool interoperability is supported can be found in D5.6 [4]. More details on exploiting development-time EDDIs for producing runtime EDDI components can be found in D7.1 [9] and D7.2 [10].

[15] Epsilon (eclipse.org)
[16] Eclipse Modelling Project | The Eclipse Foundation

2.1.4.2 *BayesFusion GeNIe Modeler*

This is a commercial tool[17] **not** provided by IESE (a trial version and academic licenses are available), but used in the context of modelling Bayesian Networks (BNs). The tool's output format (.xdsl) is planned to be supported for conversion into/from EDDIs. IESE will make corresponding conversion tools available for this purpose. Equivalent tools that produce or convert to the same format can also be supported.

In Figure 11, there is an example of modelling a Bayesian network in GeNIe Modeler.

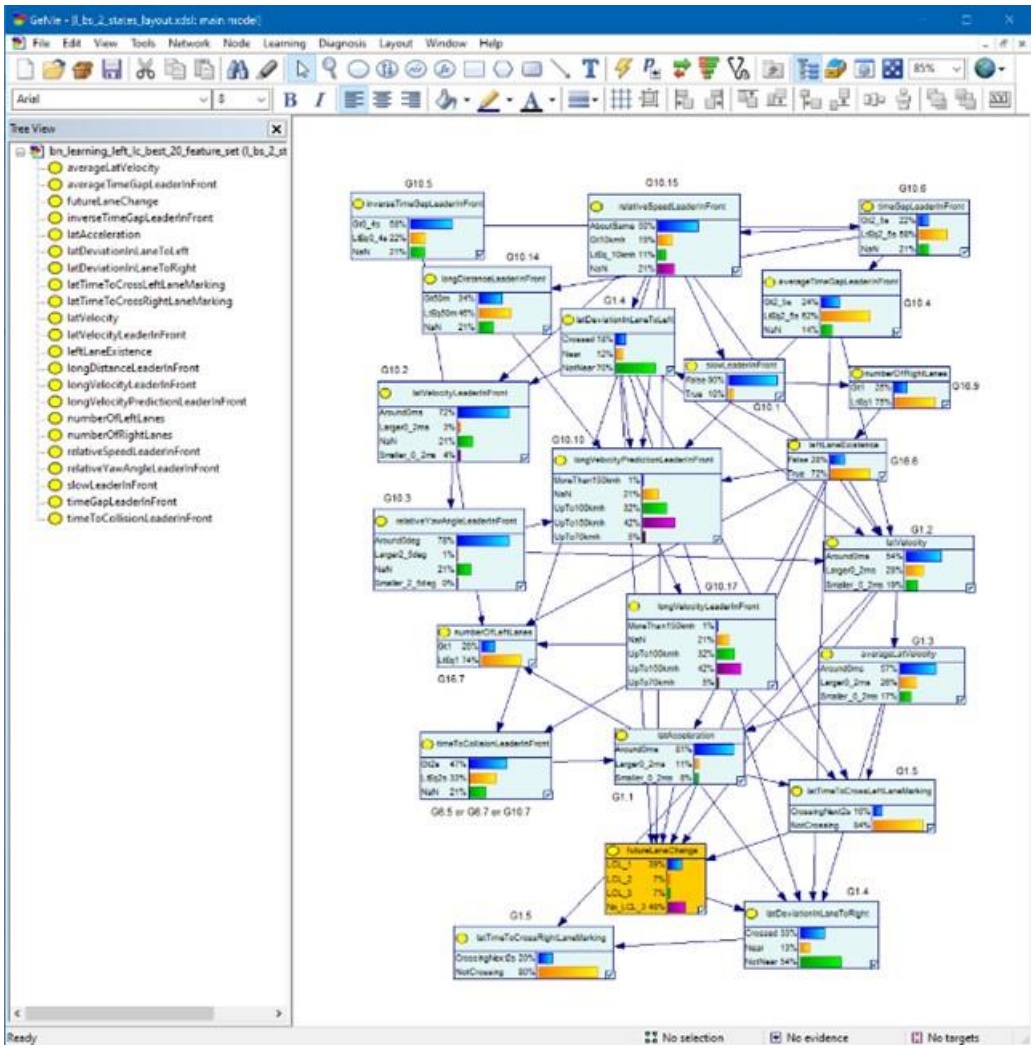


Figure 11: Example of Genie BN Modeling

2.1.4.3 *HiP-HOPS*

HiP-HOPS[18], or "Hierarchically Performed Hazard Origin & Propagation Studies" to give it its full title, is a comprehensive model-based safety analysis methodology with a tool of the same name. Originally developed in the late 1990s, it has been the focus of

[17] https://www.bayesfusion.com/genie/

[18] https://hip-hops.co.uk

continuous development over the ensuing 20+ years and its initial foundation has since played host to a wide range of advancements and additional functionalities.

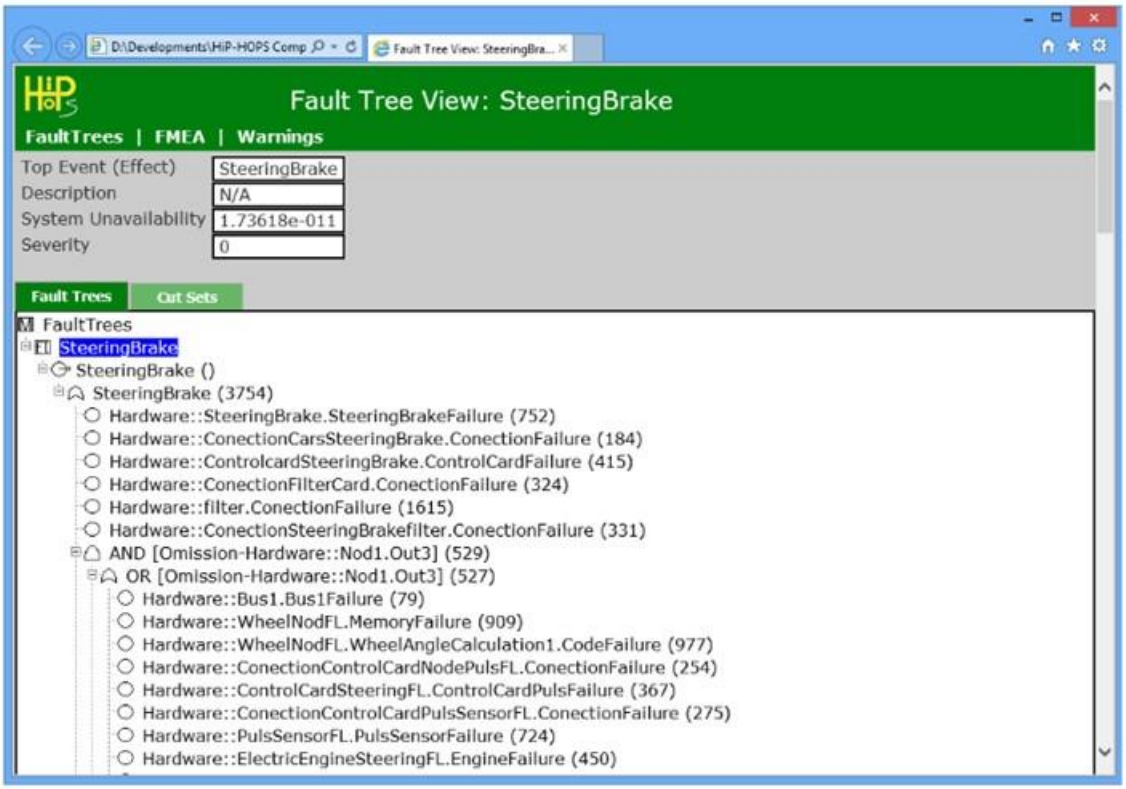


Figure 12: HiP-HOPS analysis output

The core HiP-HOPS methodology consists of four main phases: system modelling, failure annotation, synthesis of fault propagation models, and the analysis phase, which involves Fault Tree Analysis (FTA) & Failure Modes & Effects Analyses (FMEA) (Figure 12). Because HiP-HOPS is primarily an analysis engine, system modelling and failure annotation takes place in established modelling tools such as Matlab Simulink[19], SimulationX[20], and MetaEdit+ (with EAST-ADL)[21].

HiP-HOPS then imports the resulting model file, synthesises failure models that trace the propagation of failures though the system, and performs analyses on them to obtain fault trees and FMEAs. The output are XML files that can be viewed in a web browser or Excel spreadsheets containing the same data.

HiP-HOPS also offers support for other experimental functionality, including architectural optimisation, automatic allocation of safety requirements, dynamic failure analysis, and more. A full manual is available on the HiP-HOPS website[22].

By combining the architectural input model and the analysis results, a complete model can be generated as a basis for EDDIs.

[19] https://www.mathworks.com/products/simulink.html
[20] https://www.esi-group.com/products/system-simulation
[21] https://www.metacase.com/solution/east-adl.html
[22] HiP-HOPS website

2.1.4.4 Tool Adapter & EDDI Editor

For these tools to support ODE models, converters are required. A standalone model converter with basic editing capabilities, the EDDI Editor, has been developed to support conversion from HiP-HOPS and Dymodia (for state machines) This converter imports a HiP-HOPS or Dymodia file (input, output, or both) and generates an ODE model from it, which can be saved in XML format. It can also open ODE files exported from safeTbox and merge files from different tools into a single model. This tool is described further in section 6.5.2.5.

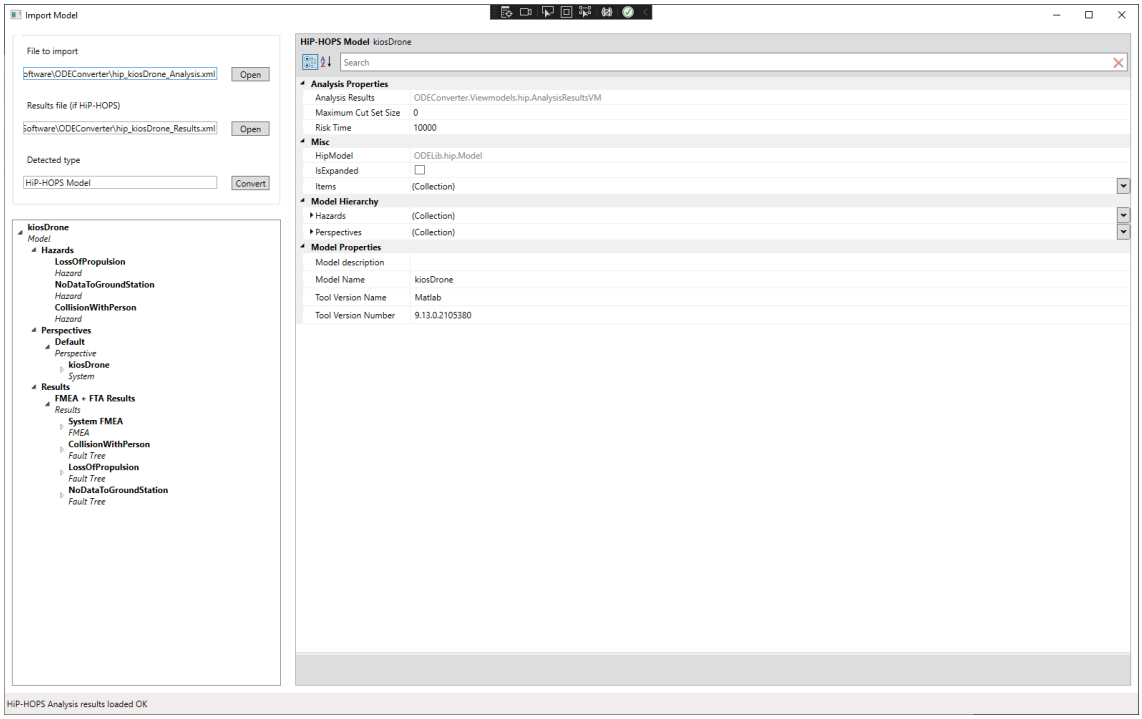


Figure 13 - EDDI Editor

In addition, the Common Tool Adapter is based on the Apache Thrift interface definition framework (see Figure 14).

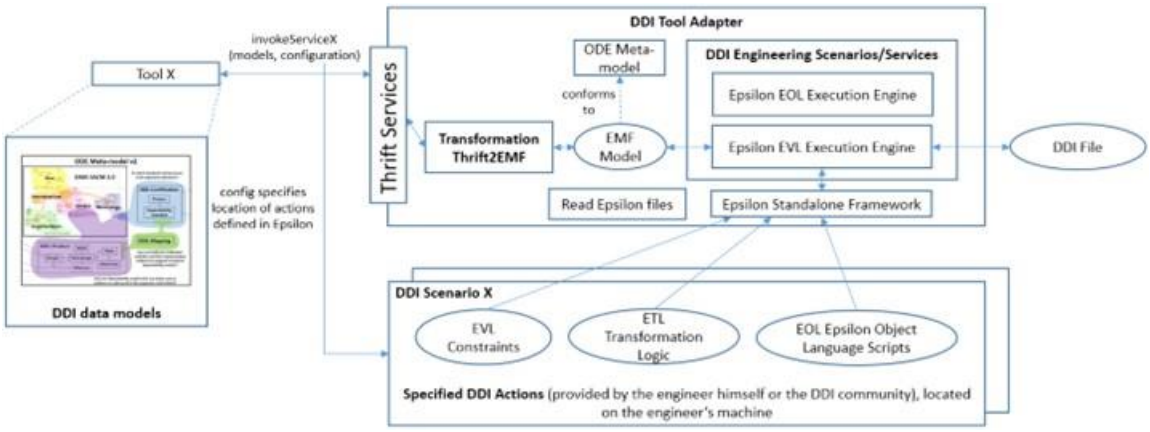


Figure 14: Common Tool Adapter

Thrift is a cross-language, cross-platform framework. Its type system allows ODE data structures to be defined and exchanged across tools. It offers a service-based approach to allow definition of ODE-oriented services such as validation.

2.1.4.5 Requirements status

Table 5: Requirement status for EDDI-based safety analysis tools

Req. No.	Requirement	Priority	Resp. WP	Status
D64	Establish a dependability-driven methodology for the development of EDDIs.	SHALL	4	Done
D65	Extend the ODE metamodel to support EDDIs by incorporating new capabilities for safety, uncertainty, and runtime monitoring and diagnosis.	SHALL	4	Done
D66	Define an interface or protocol for the exchange of data between EDDIs or between EDDIs and other components	SHALL	4	Done
D67	Generate safety-targeted EDDI artefacts with tool support.	SHALL	4	Done
D68	Support hazard and risk analysis (HARA) of MRS.	SHALL	4	Done
D69	Support semi-automatic requirement allocation for MRS.	MAY	4	Partial support
D70	Support design-time semi-automatic synthesis of dependability models such as fault trees and FMEAs for MRS.	SHALL	4	Done
D71	Support dynamic safety analysis of MRS on the basis of combined architectural & behavioural modelling.	SHOULD	4	Done
D72	Provide a SafeML software component that can estimate the confidence in ML classification accuracy at runtime to support safety-related decision making.	SHALL	4	Done
D73	Expand the SafeML software component so that it gives the reasoning behind the confidence level assigned to the ML classifications it monitors (e.g. by highlighting problematic runtime inputs).	MAY	4	Done
D74	Provide a method and a software component that can translate confidence measures into metrics of the reliability of safety-related ML classification models.	MAY	4	Partial support
D75	Integrate the reliability metrics for ML classifiers with EDDIs that do probabilistic reasoning and decision making about safety at runtime.	MAY	4	Done
D76	Security EDDIs shall be based on the ODE to ensure compatibility and ease of import/export when taking into account the safety implications of security threats or violations.	SHALL	4	Done
D77	Runtime EDDI artefacts shall be compatible with the ODE to ensure ease of generation and execution using safety EDDI information.	SHALL	4	Done
D78	Bilateral interface or exchange format to allow information to be communicated with Simulation-based Testing Tools.	SHALL	6	Done
D79	Bilateral interface or exchange format to allow information to be communicated with ExSce Workbench.	SHALL	3	Done
D80	Bilateral interface or exchange format to allow information to be communicated with Trajectory Planning.	SHALL	2	Done
D81	Bilateral interface or exchange format to allow	SHALL	2	Done

Req. No.	Requirement	Priority	Resp. WP	Status
	SafeML to be used to analyse object identification models with Collaborative Perception.			

2.1.5 EDDI-based Security Analysis Tools

2.1.5.1 Security

The main goal of the EDDI-based security analysis tools is to collect data on the security condition of a specific system. Uncovering vulnerabilities within individual system components enables the identification of potential attacks that adversaries might attempt by exploiting the existing weaknesses in the system. The acquired information is then stored in an EDDI model-based artifact, encompassing all the dependability information of a system. The tools that are involved in the process described above are gathered below:

- **OpenVAS** is used for collection of security information of the target system. It is an automated scanner tool that can scan given network and/or subnetworks for available services. Using scanning tools offers the advantage of uncovering services actively running on devices within the target system. These tools have the capability to identify services that may be operational, even when the provider of the system information is unaware of their existence.

  OpenVAS' primary strength lies in its meticulous scanning of all ports on the target system to identify active services, presenting a comprehensive report detailing discovered assets, such as running software and specific version numbers. Moreover, it can launch attacks on identified services using a wide range of known exploits. The created reports on vulnerable services, include a high-level description of each vulnerability, its associated CVE, CVSS score, and severity level. OpenVAS exhibits advanced capabilities by incorporating wrappers for other vulnerability scanners, such as Nmap and wapiti, thereby expanding its coverage and increasing the variety and quantity of detected vulnerabilities. Additionally, OpenVAS offers predefined configurations tailored to common scanning scenarios, including options for fast, fast ultimate, deep, and deep ultimate scans.

- The process of requesting free repositories, catalogs and databases (CVE and RVD – D5.1) for known vulnerabilities of the recognized software, hardware and communication protocols is facilitated through the utilization of two parsers **CVE-search** and **RVD custom parser**. CVE-search is a tool designed to import CVEs and CPEs data into a MongoDB, streamlining the search and processing of CVEs. Its primary advantage lies in the creation of a local instance of CVE, catering to lookup requests. This approach reduces the need for direct queries to public CVE databases, enhancing efficiency. Simultaneously, local requests are handled fast, offering faster responses without exposing sensitive information to the internet. Key offerings of cve-search include: i) a back-end for storing vulnerabilities and related information, ii) an intuitive web interface for searching and managing vulnerabilities, iii) a suite of tools for querying the system, and iv) a web API interface.

  Moreover, an RVD custom parser has been developed, enabling the use of the "rvd list --dump --label vulnerability" command. This command retrieves all database

entries in RVD that are labeled as vulnerabilities. Each entry provides comprehensive information for robot vulnerabilities, encompassing related CVEs and CWEs, affected systems, severity scores such as RVSS and CVSS, as well as detailed descriptions of exploitation and mitigation measures. The comprehensive collection of robot vulnerabilities serves as the input for our custom parser. A set of Java classes has been crafted to store and handle the information associated with incoming robot vulnerabilities (refer to Figure 12, D5.6). The primary class, named "RvdVulnerability," anchors this structure, complemented by four essential subclasses: "Severity," "Exploitation," "Flaw," and "Mitigation." These subclasses play crucial roles in categorizing and managing specific aspects of the vulnerability data, ensuring a well-structured and organized representation within the Java framework.

- For the identification of potential attacks two tools are utilized, **CVE-search** and **CAPEC custom identifier**. CVE-search has also another capability to be queried for known attacks associated with a provided CVE-ID or a specific product (software/hardware). When requesting information for a particular vulnerability, especially if the output is specified in JSON format, a notable portion of the returned information includes a list of attacks directly linked to the specified vulnerability.

  The CAPEC custom identifier makes use of a local instance of the CAPEC catalog to retrieve information about known attacks associated with specific weaknesses. A set of Java classes has been developed to store and manage all the information derived from the CAPEC repository (refer to Figure 13, D5.6 [11]). The CAPEC repository structure encompasses an "AttackPatternCatalog," housing a comprehensive list of "AttackPatterns." The "AttackPattern" class corresponds to known vulnerabilities, encapsulating a wealth of information, including related weaknesses, associated attacks, and suggested mitigation actions. This Java class structure ensures a systematic representation and efficient handling of data sourced from the CAPEC repository within the application.

- The information generated utilizing the aforementioned tools is integrated into the **Security EDDI**, facilitating the transfer of this information to the runtime environment for effective threat mitigation. A crucial prerequisite in this scenario involves the continuous monitoring of security events during runtime. The tool responsible for monitoring incoming malicious packets on the network is an **Intrusion Detection System** (IDS).

  Snort is a widely recognized open-source IDS that has garnered popularity and has been extensively explored in the literature. It adopts a rule-based methodology to characterize malicious network activities, activating alerts when a specified rule is matched. Operating on a single-threaded architecture, Snort leverages the TCP/IP stack to capture and scrutinize network packets, encompassing both headers and bodies. Upon the detection of rule-satisfying events, Snort generates alerts and logs them, providing the basis for creating reports derived from these alerts. Its design makes Snort particularly well-suited for addressing lightweight IDS needs, offering an effective solution for organizations seeking a flexible and adaptable intrusion detection capability. Our decision was to integrate Snort into a suite of other tools, with the objective of building a robust and flexible framework for monitoring and mitigating network threats. This commonly adopted configuration combines Snort

with Barnyard and a relational database, presenting a comprehensive solution that organizations often choose to fortify their security against cyber threats. This integrated setup allows for efficient handling, analysis, and storage of Snort-generated alerts, contributing to a more effective and scalable approach to network security. Barnyard, a data handler and translator, reads Snort's binary logs, decrypt and organizes them into a more comprehensible and structured format. This processed data is then efficiently dispatched to a database for storage and subsequent in-depth analysis.

Table 6: Requirement status for EDDI-based security analysis tools

Req. No.	Requirement	Priority	Resp. WP	Status
D82	Develop a methodology for delivering security-based EDDIs.	SHALL	5	Done (D5.1)
D83	Extend the ODE metamodel to include support for security characteristics of a system.	SHALL	5	Done (D4.2 – D5.2)
D84	Develop an interface for data transfer between EDDIs and other SESAME or system components.	SHALL	5	Done Security EDDI sends data to Concert by publishing it to predefined ROS topics
D85	Produce security-targeted EDDI artefacts.	SHALL	5	Done (D5.6)
D86	Generate attack trees or other security-describing models in a semi-autonomous.	SHALL	5	Done
D87	Provide semi-autonomous security analysis during design time.	SHALL	5	Done
D88	Provide semi-autonomous security analysis during runtime.	SHOULD	5	Done (5.6)
D89	Provide a mechanism ensuring the security of EDDI executable.	SHALL	5	Done (5.5)
D90	Develop a ML-based Intrusion Detection System that recognizes malicious incoming traffic.	MAY	5	Not done
D91	Create labelled data that classify incoming traffic to benign or malicious.	MAY	5	Not done
D92	Determine the accuracy of the ML-based Intrusion Detection System classification model.	MAY	5	Not done
D93	Security EDDIs shall be compatible with the ODE for easier management.	SHALL	5	Done (D4.2 – D5.2)
D94	Runtime EDDIs shall be compatible with the ODE for easier management.	SHALL	5	Done –- (D5.6)
D95	Bilateral interface or exchange format to allow information to be communicated with Simulation-based Testing Tools.	SHALL	6	Partial support
D96	Bilateral interface or exchange format to allow information to be communicated with ExSce Workbench.	SHALL	3	Not done There is no communication between these components
D97	Bilateral interface or exchange format to allow information to be communicated with Trajectory Planning.	SHALL	2	Not done There is no communication between these components

2.1.6 Simulation-Based Testing of EDDI Tools

2.1.6.1 Overview of WP6 Simulation-Based Testing and Multi-Stage MRS Testing Methodology

The simulation-based testing platform proposed for WP6 interfaces with the MRS simulation and deliberately alters or manipulates the simulation state in order to assess how the system under test responds in various scenarios. This allows the behaviour of the simulation under such disruptions (corresponding to transient faults, or deliberate attacks) to be assessed by means of user-defined scenario-specific performance metrics. Logs are generated, both from the testing platform and from the MRS simulation itself, in order for SESAME users to assess the impact of their chosen simulation-based testing campaigns.

Depending on the intent and availability of use case partners, it may be useful to assess the behaviour of the discovered faults on the real physical MRS system. The SESAME platform also supports interconnection to physical MRS scenarios, to inject faults and noise in their behaviour and verify conformance between the behaviour of simulated and real systems under fault scenarios. The overall methodology incorporating a transition from simulation to physical testing is illustrated in Figure 15, which comprises a number of stages. The human symbol indicates those points which involve the use of human assessment or intervention (e.g., robotics and software engineers), and the gear symbol indicates automated processing or code execution. The robotic arm symbol indicates the stages in the methodology which require access to a lab environment and implementation of the robotic scenario in order to execute and assess it (which are also emphasised with a blue background).

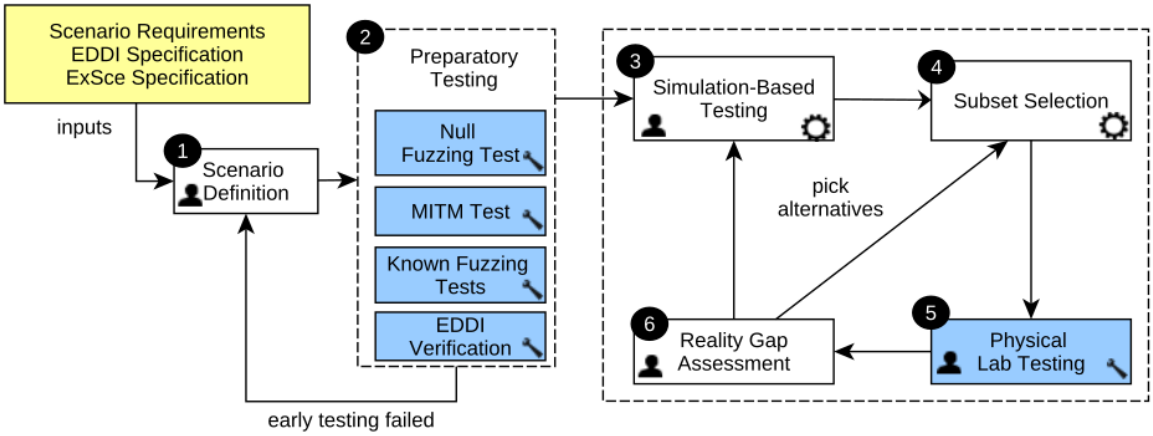


Figure 15: The integrated testing methodology, incorporating simulation-based testing and physical testing

Step 1: which precedes the simulation experiments, begins with the definition of the MRS scenario, its intent and the testing strategy, informed by the inputs to Step 1; including an Executable Digital Dependability Identity (EDDI) as defined in Section 2.1.8.1, and the Executable Scenario (ExSce) Specification. The testing strategy will involve the specification of performance metrics to quantify the MRS requirements, and

the selection of fuzzing operations to test specific system components by disrupting their internal messages. This work is performed in conjunction with industrial partners developing a particular scenario, in order to exploit their experience in the intended task of the scenario, the requirements it must fulfil, and the types of failure to which the system could be subjected. This step may be performed multiple times, considering the results of the validation process in Step 2.

Step 2: performs preparatory testing in a lab environment to validate the scenario defined in Step 1. A primary aim here is to verify conformance between the simulation and the physical system in a fault-free test case. Reality gaps between simulation and physical systems could result from a variety of sources: different delays in communication with the physical system and in simulation; simplified simulation algorithms and models; and noise/transient signals that trigger violations in a real system. After this, a series of staged tests will be performed, gradually introducing the WP6 testing platform into the simulation loop and verifying its behaviour on known tests. If any of the stages in Step 2 fail, then it may be necessary to return to Step 1 to attempt to identify the cause and compensate by scenario/fuzzing selection changes.

Step 3: performs simulation-based testing as detailed fully in Section 2.1.6.2. The testing platform will iteratively generate and refine tests in an evolutionary loop, starting with an initial random test selection and refining the population to generate those that expose the worst-case performance metrics (representing edge cases and maximum failures) at minimal fuzzing time exposure.

Step 4: selects a subset of configurations found from simulation-based testing for physical execution. Because the simulation-based testing process is much faster and more economical to execute than physical testing, a larger and more diverse set of configurations can be explored in simulation-based testing than could be tested physically. Therefore, the physical testing scenarios chosen for execution should be selected in order to carefully utilise the limited physical testing time and resources, while providing information on both the conformance between simulation and reality, together with validation of the real testing results.

This step also considers safety within the physical tests selected, since some simulated tests may not be viable to execute directly. This may occur since either the modifications which they introduce to physical system values could cause system instability, or the operations themselves could disable sensors or override motion in ways that cause a collision or other risk. Various ways exist to mitigate this, such as safety zone expansion, automated safety interlocks independent of the testing process, manual intervention by an operator such as a safety pilot, or introduction of virtual objects. These are fully detailed in our deliverable D6.6 [12].

Step 5: performs physical testing using the testing platform in the loop. The testing platform will interface with the physical simulations and inject a particular test configuration found from simulation, recording performance metrics.

Step 6: quantify and compensate for reality gaps between step 3 (simulation) and step 5. Simulation necessarily provides a simplified model of the real system, which means that physical system performance could deviate from reality either by false positives or false negatives. Reality gaps could be systematic, or localised (only occurring under specific fuzz testing configurations exhibiting combinations of operators, timing and other pa-

rameters). In the cases in which reality gaps exist, it is important to consider its source and how it can be handled. If the reality gap occurs as a result of inadequate modelling of a particular component, this model can potentially be improved and the simulation re-executed.

2.1.6.2 SESAME Simulation-Based Testing Methodology

The SESAME WP6 simulation-based testing component enables the evolution of fuzz testing configurations that reveal violations of system safety requirements, via a simulation-based testing loop. These testing campaigns correspond to edge scenarios that can be analysed by domain experts and inform the hardening of the MRS system scenario. Identifying these edge scenarios using our implemented framework, entails following a methodology presented with a number of steps (illustrated in Figure 16). We refer interested readers to the SESAME deliverable D6.6 [12] for further information about the developed simulation-based testing methodology.

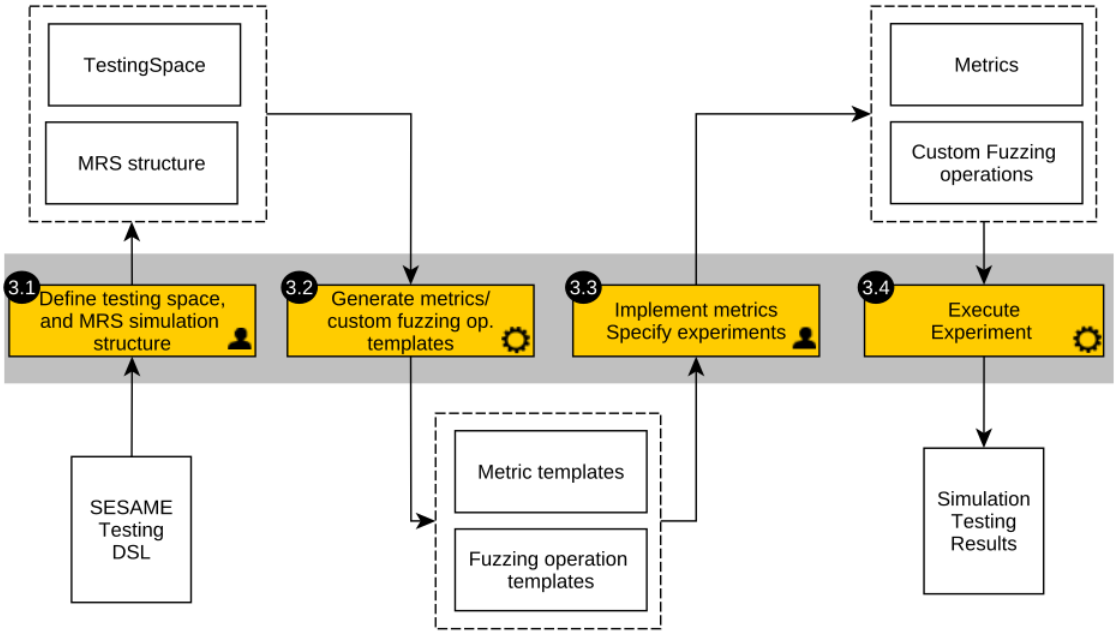


Figure 16: The simulation-based testing methodology for WP6

The steps defined in the methodology below are:

Step 3.1: Platform users (MRS test engineers) specify the testing space specification and MRS structure model. The MRS structure model corresponds to the information provided by the Executable Scenarios workbench (see deliverable D3.2 [13]) which can optionally be augmented from the information received from a model-to-model transformation. This MRS model encodes the characteristics of the target mission, including the robotic systems, the simulation structure, and the mission requirements. The MRS structure model provides an encoding of the earlier scenario setup step (Step 1) in the wider integrated methodology of Figure 16. The instantiation of the Testing DSL may also be informed by the definition of the EDDI, permitting, for example, the generation of automatic fuzzing operations from the EDDI specification.

Step 3.2: The Testing and MRS models are automatically transformed to mission requirement performance metric templates that provide the framework for quantifying

whether a mission or safety requirement is met, and if not, the extent and impact of the violation. Since the fuzzing operations selected and requirements quantification are mission and system-specific, users are responsible for populating these templates. The SESAME integrated online platform may also be used in performing these model-to-code transformations from an uploaded model.

Step 3.3: Users implement the previously defined metric templates, to create the required custom scenario-specific metrics. They specify via the SESAME DSL particular fuzzing test campaigns, corresponding to the particular experiments that they plan to execute. These constitute a selection of a particular set of fuzzing attacks defined in the testing space, and a subset of the defined metrics to assess requirement violations. The type of experiment and any experiment-specific parameters are also specified. For example, with genetic algorithms, experiment-specific parameters such as the number of generations and iterations can be included.

Step 3.4: The SESAME fuzzing engine consumes the SESAME Testing DSL model, metrics and selected experiment details and utilises the information from these models in an evolutionary optimisation step, during which it evolves a population of tests. Each test comprises a choice of fuzzing events, with events specifying the participating robots, the simulation messages to be fuzzed and their characteristics (including the fuzzing operation and timing constraints). The experiment runner which incorporates the evolutionary algorithm evaluates each test by first dynamically generating a specialised test runner which acts as a middleware, communicating with the low-level simulator via a simulation-specific interface, and using any custom supplied metric definitions provided in Step 2 to quantify the impact of the fuzzing test.

This information is communicated to the experiment runner and used to guide a multi-objective optimisation process. This process uses genetic operations such as mutation and crossover to create new tests, discarding the worst performing campaigns from the population. This iterative process continues until an experiment-specific termination criterion is satisfied, i.e., either the maximum number of permitted generations is reached, or no improvement occurs over a specified number of evolution rounds. Once the evolution terminates, SESAME produces an approximate Pareto optimal set of fuzzing campaigns, along with the associated approximate Pareto front of mission requirement values. SESAME also logs all intermediate results using the framework to the model, or to files, depending on the selection in the DSL.

2.1.6.3 Testing tool and simulation platforms interfaces for KUKA Use Case

Within the context of the Simulation Based Testing Framework, an important role is played by the interface between the Testing Tool and the simulation platforms. Upon the Java implementation of the testing framework developed by York, an abstraction interface layer (provided by ISimulator Java interface) is provided to support the integration with different (robotic) simulation environment, via specific classes in distinct packages that implement this generic interface.

In particular, the interfacing strategy of the simulation-based testing framework is based on the assumption that the simulated environment exposes model variables through a publish/subscribe mechanism. This is perfectly compatible with ROS compliant simulators, but it requires adaptation tiers to couple with simulation engines that are not equipped with a native ROS interface. For this reason, in the KUKA use case, intercon-

nection to the TTS DDD simulator has been focused on the implementation of a general purpose set of APIs, (SimlogAPI V2), together with a central shared memory interconnection component, that also permits interconnection to the EDDI and to the deployed robotic system as shown in Figure 17.

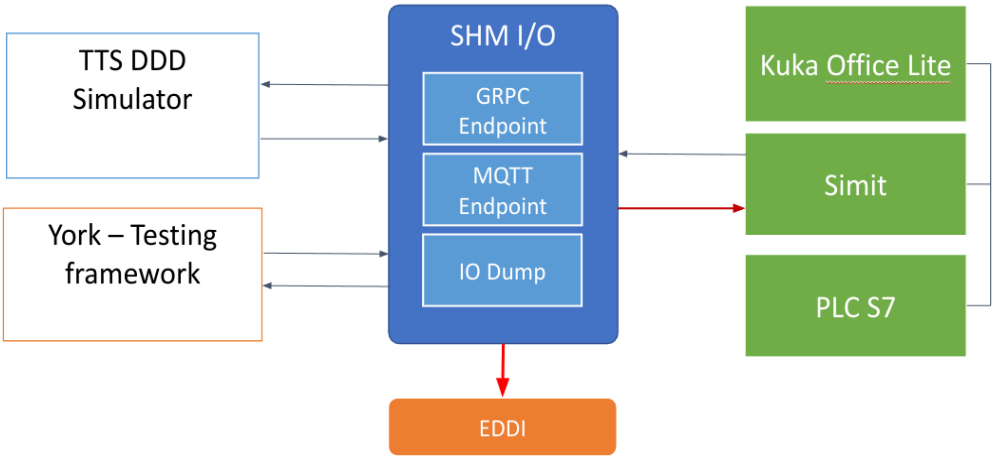


Figure 17: Generic scenario

2.1.6.4 The TTS SimlogAPI V2

The TTS Simlog API underwent an iterative process of improvement that led to the latest release currently applied for the integration of the components in the simulation based testing scenarios. Some of the previous features have been modified to comply with the evolved requirements of the systems during the advanced evaluation phase, while some others have been maintained and enhanced. The following list reports the main requirements fulfilled by the final version of the API:

- be cross-platform and cross-application, supporting the adoption by applications written in major programming languages (such as the testing platform in Java and EDDIs implemented in Python);
- ensure high performance communication, to support the efficient exchange of large sets of I/O signals between the testing framework and the TTS DDD simulation engine;
- comply with a publish/subscribe approach, to support the extensibility of the type and structures of the exchanged messages;
- support direct read/write operations of registries of the Shared Memory enabling synchronous polling and modification of data;
- support the Google proto-buffer library[23] of standard message definitions for the modelling of general purpose dynamic data structures capable of capturing the different needs to represent complex information exchange between components.

[23] https://github.com/protocolbuffers/protobuf

- support deterministic simulations by allowing precise control of simulation time via the testing platform, and ensuring proper sequencing of operations

- enable the full synchronisation of external components with the internal simulation thread through a controlled queueing of the read/write events.

The API has been based on gRPC[24] which is a high performance, open-source universal remote procedure call framework. It provides support to most of the common development platforms used for software engineering like C++, Java, C#, and others. The gRPC framework uses Protocol Buffers, an industry ready open-source mechanism for serialising structured data provided by Google, as both its Interface Definition Language (IDL) and as its underlying message interchange format. The interface for Protocol Buffers communication is defined by a proto definition file. From the architecture point of view, gRPC relies on the implementation of a server artefact and of client stubs that manage the function calls in a way that is transparent both to the server and to the client applications. The framework, starting from the Protocol Buffer defined interface, generates the server and stubs code in the specific target language of the hosting application, so that both the endpoints of the communication can use local compiled methods. Relying on a binary socket implementation, the procedure calls run not only locally but also remotely through the network. This technology brings a set of benefits that can be summarised in:

- *Transparency of the underlying transport protocol*: the application code doesn't have to comply with specific patterns like the stateless mode of the RESTful API;
- *Performance*: the default binary implementation of the Protocol Buffers is designed for performance so the impact on the normal operation times is negligible;
- *Vendor independence*: the possibility to use the proto definition file format as IDL to define the API, allows to open the specification to any vendor interested in integrating with the architecture;
- *Complexity management*: the possibility to define data structures and articulated method signatures brings within the integration layer the same design patterns that can be applied with normal OOP (Object Oriented Programming) code, i.e., the observer pattern.

The API definition is split into two major set of functionalities:

- **Simulation server API (SimServerAPI)**: it exposes the functionalities to control the time progression of the simulation, allowing an external API user to start, pause, step and end a loaded model, as well as receiving notification about the simulation clock steps. This API has been integrated as native remote control interface of the DDD Simulator and is available on all the simulation model distributions. This is the part of the API allowing the simulation-testing platform to control the advancing of time in the DDD Simulator, in order to provide deterministic simulations.

[24] https://grpc.io/

- **Simulation I/O API (SimlogAPI)**: it gives access to the internal Shared Memory of the simulation, through which all the components of the model and the external data sources exchange signals and internal states.

The two API sets correspond respectively to two separate proto files **SimServerAPI.proto** and **SimlogAPI.proto**, available at our SimlogAPI Github repository (branch V2)[25], that specifies the exposed functions and the exchanged data structures (called messages).

The SimServerAPI

```
/**
 * DDD Simulator remote control API.
 */
service SimServerAPI {

    /**
    * Steps the simulation
    */
    rpc step(StepRequest) returns (SimStatus);

    /**
    * Starts the simulation
    */
    rpc start(StartRequest) returns (SimStatus);

    /**
    * Stops the simulation and put it in finished status
    */
    rpc stop(google.protobuf.Empty) returns (SimStatus);

    /**
    * Pauses the simulation, execution can be resumed
    * with a subsequent call to start
    */
    rpc pause(google.protobuf.Empty) returns (SimStatus);

    /**
    * Subscribes to simulation status events. The subscriber will be notified
    * about start/stop/pause events occurring on the model either triggered
    * by an external component through the API, or by the end use by the GUI
    */
    rpc subscribe(google.protobuf.Empty) returns (stream SimStatus);

}
```

Figure 18: SimServerAPI methods

Figure 18 reports the specification of the main methods that compose the API:

[25] https://github.com/sesame-project/SimlogAPI

Method	Behaviour
start(StepRequest)	Starts the simulation, putting the status of the animation thread in automatic mode. The provided step request specifies the number how much a single step of the animation will last in milliseconds. If the simulation is already running, a call to start will not affect the execution. If start is called on a stopped simulation, the result will be invalid and the model will stay in finished status. If start is called on a paused model, the execution will be resumed with the new provided step size. The method returns a SimStatus message corresponding to the new status. The API caller can use the message to verify if the action succeeded.
pause()	Pause the execution of a running model. The animation thread won't be stopped and will be ready for subsequent calls to start(). If called on a paused or stopped model the method will have no effect. The method returns a SimStatus message corresponding to the new status. The API caller can use the message to verify if the action succeeded.
stop()	Stops and finishes the execution of a model. The model will complete the exit operations. The method can be called on a started or paused model. After a call to stop, no further calls to start or pause will have effect. The method returns a SimStatus message corresponding to the new status. The API caller can use the message to verify if the action succeeded.
step(StepRequest)	Perform a single simulation step advancing the time of the required amount of milliseconds. The model will complete all the logical and visual operations (included notifications) triggered during the specified amount of time and then will return in pause mode. If called on a running (started) or stopped model it will have no effect. The method returns a SimStatus message corresponding to the new status. The API caller can use the message to verify if the action succeeded.
subscribe()	Registers an external component to be informed on simulation status related events. The subscriber will receive notification when the simulation model changes state in reaction to a call

Method	Behaviour
	to the start/stop/pause methods or to an action of the end user on the GUI. The notified events are structured in a SimStatus message the contains detailed information about model (see message structures).

The following tables document the message structures defined by the SimServerAPI:

```
message StepRequest{

    int64 deltaTime = 1;
}

message StartRequest{

    int64 deltaTime = 1;
}

message SimStatus{

    int64 clock = 1;

    int64 deltaTime = 2;

    bool alive = 3;

    bool playing = 4;

    int32 exitCode = 5;

}
```

Figure 19: SimServerAPI messages

StepRequest		
deltaTime	int64	Amount of time in milliseconds the simulation clock should progress in response to a step() call.

StartRequest		
deltaTime	int64	Amount of time in milliseconds of each single step of the simulation clock during the auto-

		matic execution.

SimStatus		
clock	int64	Current value of the internal simulation clock. Value expressed in milliseconds since the initialization of the model.
deltaTime	int64	Current simulation step size in milliseconds.
alive	bool	True if the simulation model is ready for execution. When true start/step methods can be called.
playing	bool	True if the simulation is running in automatic mode. When true calls to start/step will have no effect, whale stop/pause can be called.
exitCode	int32	Exit code of the simulation, valid only when alive=false, playing=false. The meaning of the provided code is model specific.

The SimlogAPI

```
service SimlogAPI {

    /**
     * Gets a topi descriptor
     */
    rpc GetTopicInfo(TopicInfoRequest) returns (TopicInfo);

    /**
    * Creates a subscriber handle to use when issueing
    * new subscription reuquests with the Subscribe method
    */
    rpc CreateSubscriber(Subscriber) returns (stream SimlogMessage);

    /**
    * Subscribes to a topic descriptor
    */
    rpc Subscribe(SubscriptionRequest) returns (google.protobuf.Empty);


    /**
    * Inject a overwriting topic (shadow) on top of another topic (shadowed)
    * The method automatically creates an observer of the shadowed topic
    */
    rpc Inject(InjectRequest) returns (InjectResponse);

    /**
    * Opens a topic stream for writing
    */
    rpc Publish(stream PubRequest) returns (google.protobuf.Empty);

    /**
    * Executes a single write
    */
    rpc Write(PubRequest) returns (google.protobuf.Empty);

    /**
     * Executes a single synchronous read
     */
    rpc Read(ReadRequest) returns (ReadResponse);

}
```

Figure 20: SimlogAPI messages

Figure 20 reports the specification of the main methods that compose the API and that allow the testing platform to operate with the data exposed through the SharedMemory:

Method	**Behaviour**
GetTopicInfo (TopicInfoRequest)	Requires information about a topic. The returned data informs the caller if the topic exists, the actual topic path to be used for read/write or pub/sub operations, the type of expected value and the direction (IN/OUT/INOUT) of the topic.

Method	Behaviour
CreateSubscriber (Subscriber)	Creates a subscriber handle required to successfully call the Subscribe method. Each subscriber should use a single generated handle to be sure that all events will be notified in the correct sequence on a single socket. The method opens and returns the channel for messages notification specific to the calling subscriber.
Subscribe (SubscriptionRequest)	Registers a specific subscriber to a requested topic. The generated notifications will be sent on the channel opened with the CreateSubscriber method. The notified messages will comply with the topic description reported by the GetTopicInfo method.
Publish (stream PubRequest)	Opens a stream for asynchronous publication of messages on a specific topic. The opened stream will accept only messages matching the type contained in the provided TopicDescriptor instance.
Write(PubRequest)	Performs a single synchronous operation of writing on a specific topic. The payload of the sent message must comply with the expected topic value type returned by the GetTopicInfo method.
Read(ReadRequest)	Performs a single synchronous reading operation on a set of topics. The returned message contains the list of values corresponding to the requested topics. Each value is compliant with the topic description returned by GetTopicInfo.
Inject(InjectRequest)	Allows an external component to behave as a man-in-the-middle on a specific topic. This method has been designed to enable the testing platform to alter signal values directed to the simulation model or generated by it and consumed by other components like the Conserts. The method creates the shadowing topics required to implement the routing of the original messages to the caller of the inject and the altered messages to the original consumer(either simulation or other components).

The following tables document the message structures defined by the SimlogAPI:

```
message InjectRequest{

    string targetPath = 1;

    string shadowPathPrefix = 2;
}

message InjectResponse{

    /**
    * Descriptor of the shadow topic
    */
    string shadowPathIn = 1;

    /**
    * Descriptor of the shadowed topic
    */
    string shadowPathOut = 2;
}


message PubRequest {

    string topic = 1;

    SimlogMessage data = 2;
}

message Subscriber {

    string name = 1;

    string uuid = 2;
}
```

```
message SubscriptionRequest {

    // full path of the topic
    string path = 1;

    string subscriberUUID = 2;

}

message TopicInfoRequest {

    // full path of the topic
    string path = 1;

}

message TopicInfo {

    // full path of the topic
    string path = 1;

    ValueType type = 2;

    DirectionType direction = 3;

    bool exists = 4;

}

message SimlogMessage{

    Header header = 1;

    ValueType type = 2;

    google.protobuf.Value value = 3;
}

message Header {

    google.protobuf.Timestamp timestamp = 1;

    string path = 2;
}

message ReadRequest {

    repeated string path = 1;

}

message ReadResponse {

    repeated SimlogMessage messages = 1;

}
```

Figure 20: SimServerAPI messages

Subscriber		
name	string	Human readable name of the subscriber for logging and debugging purposes
uuid	string	UUID of the subscriber to be created. Each subscriber must have a unique ID in a single execution session.

SubscriptionRequest		
path	string	Full path of the topic for which the subscription should be created
subscriberUUID	string	A UUID identifying a subscriber created with the CreateSubscriber method. All the subscription requests issued with the same subscriberUUID will generate events on the same notification chanell.

TopiInfoRequest		
path	string	Full path of the topic for which information is required

TopicInfo		
path	string	Full path of the topic information is referred to
type	ValueType	Type of the value exchanged on the topic
direction	DirectonType	Direction of the topic.
exists	bool	Specifies if the topic exists, i.e. the corresponding registry exists on the Shared Memory

SimlogMessage		
header	Header	Header of the message containing meta-data
type	ValueType	Type of the value contained in this message
value	google.protobuf.Value	Actual value

Header		
timestamp	google.protobuf.Timestamp	Timestamp of the message
path	string	Full path of the topic

ReadRequest		
path	string[]	Array of full path of topics to be read.

ReadResponse		
messages	SimlogMessage[]	Arrays of messages containing the values of the requested topics.

InjectRequest		
targetPath	string	Full path of the topic whose values should be altered.
shadowPathPrefix	string	Prefix of the path of the shadow topics created to send the altered values.

InjectResponse		
shadowPathIn	string	Full path of the shadow topic the altered values should be sent on.
shadowPathOut	string	Full path of the shadow topic where the injection user will receive the original values.

PubRequest		
topic	string	Full path of the topic that must be written
data	SimlogMessage	Payload of the write request

The documented messages refer to the following enumerations when defining the type of exchanged values and the direction of the topics.

```
enum ValueType {
    UNRECOGNIZED_TYPE = 0;
    NULL = 1;
    BOOL = 2;
    NUMBER = 3;
    STRING = 4;
    STRUCT = 5;
    LIST = 6;
}

enum DirectionType {
    UNRECOGNIZED_DIRECTION = 0;
    IN = 1;
    OUT = 2;
    INOUT = 3;
}
```

Figure 21 SimlogAPI enumerations

Proof of concept Deployment Scenario

A reference implementation of the API at simulation side has been developed based on TTS's DDD Simulation platform, creating a dedicated communication library. The implementation has been tested with a preliminary proof of concept model extracted from the KUKA use case. In particular, the generalisation of the SimlogAPI has been further extended to prove the possibility to adapt the developed approach to complex and articulated scenarios where several components concur to the execution of a simulated tested session. In particular, the EDDI has also been interconnected with the V2 API, and demonstrated to transmit and receive data to and from the DDD Simulator.

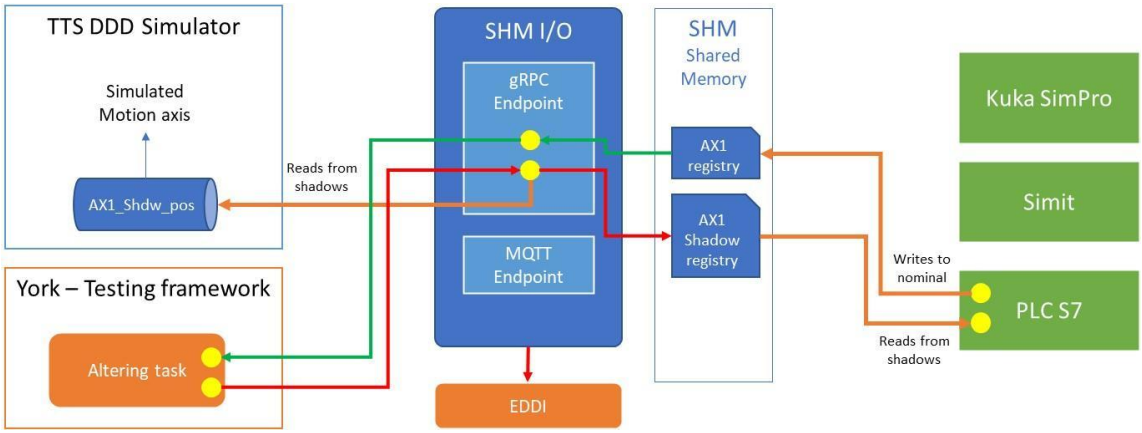


Figure 22: KUKA Proof of concept integration scenario

In Figure 22 the component SHM I/O acts as a brokering endpoint mirroring the Shared Memory used by Siemens and KUKA proprietary applications during a Virtual Commissioning session to the Testing Framework and to the TTS Simulation engine using a SimlogAPI implementation. In this way it is possible to perform several kinds of reliability tests and acquire custom metrics to measure the level of dependability of the simulated system.

2.1.6.5 Requirements status

Table 7: Requirement status for simulation based testing of EDDI tools

Req. No.	Requirement	Priority	Resp. WP	Status
D98	Support semi-automated transformation of ExSce definition into testing scenarios to be exercised in simulation.	SHALL	6	Done
D99	Support the execution of simulated ExSce with tracking of acceptance criteria and provenance data in the form of logs.	SHALL	6	Done
D100	Define an open API specification for the monitoring and manipulation of simulated ExSce at runtime.	SHALL	6	Done
D101	Support data stream acquisition from the real devices into a Digital Twin platform.	SHALL	6	Done
D102	Support integration of simulated ExSce with EDDI logic with bi-directional data exchange.	SHALL	6	Done
D103	Support the interpretation of ExSce.	SHALL	6	Done
D104	Support integration of different simulation platforms.	SHOULD	6	Done
D105	Support simulation model updating according to data stream analysis on the Digital Twin Platform.	SHOULD	6	Done
D106	Support Virtual Commissioning mode for hardware in the loop analysis.	MAY	6	Done
D107	Provide a configuration User Interface to ease the Model to Model transformation.	MAY	6	Done
D108	Support interfacing with EDDI.	MAY	6	Done
D109	ExSce shall generate the input required by the target simulator.	SHALL	3	Done
D110	ExSce shall be readable/queryable so that the testing DSL can be initialised.	SHALL	3	Done
D111	EDDIs shall be readable/queryable so that the testing DSL can be initialised.	SHALL	4, 5	Done
D112	A customisable logging mechanism shall record information related to the target mission and be linked with the target simulator.	SHALL	2, 3, 4, 5, 7	Done

2.1.7 Testing of ML Components Tools

The York team introduced an ML Testing Toolkit that encompasses the following tool-supported techniques:

1) DeepKnowledge: A white-box testing tool, which is an implementation of a new systematic testing approach for deep neural networks (DNNs) based systems. This is an assurance technique for data-driven and learning components.
2) GenRepair: A hardening and repairing tool, which incorporates a novel coverage-guided data augmentation fuzzing technique for Deep Learning models underpinned by generative AI. It provides a comprehensive methodology for repairing data-driven components. The tool incorporates a technique for model repair that strengthens the prediction capabilities of DNNs, enabling them to operate more safely when facing : (i) buggy inputs; and (ii) data distribution shifts.

Both tools operate synergistically, offering also the option for independent functionality.

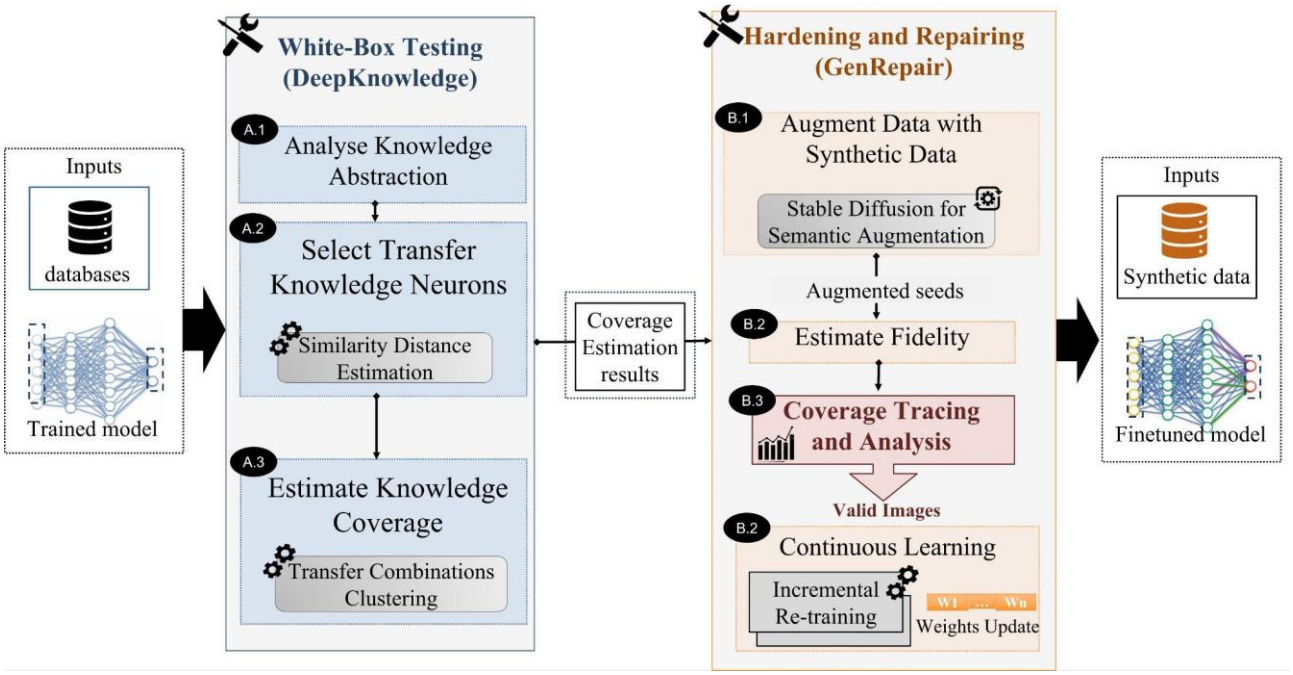


Figure 23: ML Testing ToolKit workflow

2.1.7.1 DeepKnowledge: A white-Box Testing Approach for Data-Driven Components

The DeepKnowledge tool-supported technology whose high-level workflow is shown on the left side of Figure 23, is a systematic testing approach for DNN systems. Using a trained DNN, an in-distribution (ID) and an out-of-distribution (OOD) dataset, DeepKnowledge analyses how the model learns and transfers knowledge abstractions under domain shift in step A.1 (Analyse Knowledge Abstraction). During this analysis, the learning problem is looked at through the prism of the DNN generalisation theory [14], based on which DeepKnowledge establishes a fundamental understanding of out-of-distribution generalisation at the neuron level and quantifies the individual contribution of each neuron to this process.

Step A.2 (Select Transfer Knowledge Neurons), involves a filtering process through which DeepKnowledge identifies a set of Transfer Knowledge (TK) neurons, which, given their contribution to the model's generalisation performance, are considered core DNN computational units. The rationale behind using out-of-distribution generalisation is that it enables the analysis of what part of the knowledge gained during DNN training

can be employed in a new domain without re-training or fine-tuning. Thus, it allows the simulation of real-world corner cases the model could encounter during deployment.

Finally, step A.3 (Estimate Knowledge Coverage) involves the execution of fine-grained clustering analysis to determine activation value clusters that reflect the changes in neurons' behaviour with respect to new inputs. The produced clusters of the transfer knowledge neurons are then used to assess the coverage adequacy of the test set, conforming to the combinatorial analysis method defined in DeepImportance [15].We refer interested readers to the SESAME deliverable D6.1 [16] for further information about the developed DeepKnowledge technology.

Analyse Knowledge Abstraction

Identifying the Transfer Knowledge (TK) neurons within the DNN trainable layers is a key principle of DeepKnowledge. Our approach begins by evaluating the prominence of each neuron and its connections to the OOD generalisation process. This analysis aims to identify neurons that can generalise knowledge abstracted during training and apply it to an OOD domain without re-training or fine-tuning.

These TK neurons are core components of the DNN, collectively contributing to its generalisation behaviour, and, consequently, have a positive effect on the DNN accuracy and robustness, enabling them to reach their peak performance.

Our method allows the identification of input instances that produce the highest activation values for a neuron. Then, we turn these representations into probability distributions to quantify the shift in abstracted knowledge by utilizing a statistical metric on the defined space of probability distributions at the individual level (i.e., per neuron). Neurons able to achieve a certain threshold (that is empirically defined) of the statistical measure are selected as candidate transfer knowledge neurons.

By this means we leverage a novel test adequacy criterion for testing DNN models by assessing test set quality and semantic diversity.

Estimate Knowledge Coverage

To increase our confidence in the robustness of learning-enabled components, i.e., DNN models, we assess the test set adequacy, i.e., how well the test set exercises the set of identified TK neurons.

DeepKnowledge determines regions within the TK activation value domain central to the DNN execution and clusters them into separate combinations. Each combination of clusters reflects different knowledge abstracted during the training phase.

We can assess the knowledge diversity of the test set by assessing if it adequately covers these combinations, similar to combinatorial interaction testing in conventional software testing. We posit that semantically similar inputs, i.e., inputs with similar features, generate activation values with a similar statistical distribution at the neuron level.

Software and machine learning teams can use this information to augment the dataset and increase the coverage, thus, increasing their confidence in the capacity of the DNN to function correctly when deployed in the real world.

2.1.7.2 GenRepair: A Coverage-guided hardening and Repairing Approach for Data-Driven Components

GenRepair, is a coverage-guided fuzz testing and repairing framework for learning-enabled components. In particular, GenRepair is suitable for hardening and retraining DNN models using the original training dataset augmented with the newly identified corner case inputs.

The underpinning approach applies a generative AI-based data augmentation strategy to generate new semantically tests, and leverage multiple extensible coverage criteria as feedback to guide the test generation, such that extensible DNN testing, hardening and repairing can be performed.

While most existing data augmentation methods focus only on automatically encoding buggy inputs through simple geometric and colour space transformations (e.g., changing brightness, rotation, blurring, etc.), GenRepair leverages semantic-based methods to generate inputs that have not been encountered during training, and which could yield potentially severe corner cases.

The general structure of our GenRepair approach is given on the right side of Figure 21. The tool-based technique iterates back and forth between two main components: a corner-cases generation component (steps B.1 and B.2) and a coverage-guided repair component (steps B.3 and B.4).

Augment Dataset with Synthetic Inputs and Estimate their Fidelity

In step B.1, we select input seeds from the initial dataset that may result in corner cases that resemble previously unseen data samples, potentially leading to unsafe DNN behaviour due to input transformations (e.g., corruption, perturbation). Then, we introduce an augmentation method to synthetically generate these corner cases using the selected input seeds. For this purpose, we leverage generative AI models, e.g., Inpainting using Stable Diffusion. Then, we iteratively execute a semantic-based data augmentation loop.

In step B.2, we introduce an automated fidelity estimation technique that filters out non-photo-realistic augmented seeds. This step allows to maximize the photo-realism of the augmented seeds with respect to the input seed. Since not all augmented seeds correspond to realistic images, our approach estimates the visual or textual fidelity of each augmented seed.

Coverage Tracing and Continuous Learning

Additional iterations are performed and extensible coverage criteria are deployed in step B.3 as feedback to guide the selection of augmented seeds from the previous steps. Crucially, a set of novel corner cases that is important for the performance of the DNN in its target operational environment, i.e., solve the oracle problem and increase the coverage score, is created. Finally, the initial dataset is augmented with corner cases from the data augmentation process which increase its diversity.

In step B.4, an iterative re-training mechanism is introduced, termed Coverage-Guided DNN Repairing, for which the collected corner cases are used as sources that enable the rectification of these failure patterns. We mainly deploy a continuous learning strategy

for the repairing phase. After the execution of generation and verification steps, the output of the procedure is a DNN model with repaired weights.

Our Generative repair tool efficiently generates photo-realistic corner cases, and deploy them to successfully repair DNNs achieving high accuracy without harming their initial performance on InDD data.

We refer interested readers to the SESAME deliverable D6.4 [17] for further information about the developed GenRepair technology.

2.1.7.3 Requirements status

Table 8: Requirements status for testing of ML components tools

Req. No.	Requirement	Priority	Resp. WP	Status
D113	Support white box analysis of the ML model given the training and testing sets.	SHALL	6	Done
D114	Quantify the adequacy of the test set for the target ML model.	SHALL	6	Done
D115	Synthesise new inputs using the training/testing sets as source.	SHALL	6	Done
D116	Select scenarios to be used in the online verification mode.	SHALL	6	Done
D119	ExSce shall generate the input required by the target simulator [ExSce Workbench].	SHALL	3	Partial This has been achieved for the Floorplan DSL, and additional interfaces for other parts of the ExSce and testing DSL, will be developed based on the use cases needs.
D120	ExSce shall be readable/queryable so that the testing DSL can be initialised.	SHALL	3	Partial This has been achieved for the Floorplan DSL, and additional interfaces for other parts of the ExSce and testing DSL, will be developed based on the use cases needs.
D121	EDDIs shall be readable/queryable so that the testing DSL can be initialised.	SHALL	4, 5	Partial Design time is complete and runtime

Req. No.	Requirement	Priority	Resp. WP	Status
				in progress
D122	A customisable logging mechanism shall record information related to the target mission and be linked with the target simulator.	SHALL	2, 3, 4, 5, 7	Done

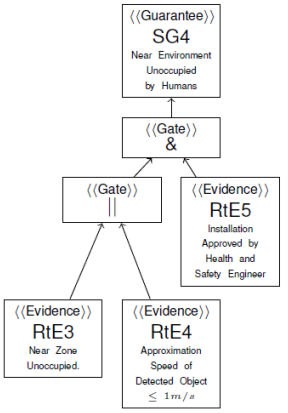
2.1.8 Runtime EDDI Components and Generation Tools

A brief overview of each of the runtime EDDI components is provided below. Further details can be found in D7.1 [9] and D7.2 [10].

2.1.8.1 Conditional Safety Certificates (ConSerts) ROS Component

ConSerts [7] are dynamic modular safety concepts, which allow systems to specify pre-assured configurations during development. At runtime, evidence collected from the operational context, e.g. through monitoring, and satisfied demands placed on external systems can trigger appropriate reconfiguration.

A brief example can be seen in Figure 24; on the left side of the figure, the graphical representation of the ConSert depicts how a guarantee, listed as safety goal (SG) 4, is established, based on live monitoring evidence, listed as runtime evidence (RtE) 3,4 and 5. Boolean logic gates connect the RtE to establish the SG. On the right side of the figure, the XML-based representation of the equivalent ConSerts is shown. Equivalent specifications (e.g. in YAML) can also be constructed similarly. Note that the figure shows a single ConSert of a solitary system. In the general case, the ConSert can also support demands, which are fulfilled by service guarantees from external systems.



(a) Graphical Representation

```
<conSert name="consert">
  <guarantees name="SG4" description="">
    <safetyProperties name="" description="">
      <refinement name="Near Environment Unoccupied by Humans" />
    </safetyProperties>
  </guarantees>
  <runtimeEvidence name="RtE3"
    description="Near Zone Unoccupied."/>
  <runtimeEvidence name="RtE4"
    description="Approximation Speed of Detected Object &lt;= 1m&#x2f;s"/>
  <runtimeEvidence name="RtE5"
    description="Installation Approved by Health and Safety Engineer"/>
  <guaranteePropagation
    sourceElement="//@configurations.0/@conSert/@runtimeEvidence.0"
    targetElement="//@configurations.0/@conSert/@gates.0"/>
  <guaranteePropagation
    sourceElement="//@configurations.0/@conSert/@runtimeEvidence.1"
    targetElement="//@configurations.0/@conSert/@gates.0"/>
    <!-- ... -->
  <gates name="Or_0" gateType="Or"/>
  <gates name="And_1" gateType="And"/>
</conSert>
```

(b) XML Representation

Figure 24: Example of ConSert, from [18]

Based on a ConSert specification, a corresponding runtime component can be generated. The runtime ConSert component offers the following features:

- It allows runtime composition checks from dependent systems to supporting systems' services. Only valid combinations of services (both in terms of function compatibility, and in terms of demand satisfaction) are allowed.
- It enables live evaluation of the guarantee(s) provided by a given ConSert, given the current RtE and demands available. Switching from one guarantee to another

can be used as a recommendation to the host system to reconfigure its behavior accordingly e.g. degrade performance to reduce safety risk due to increased operational risk perceived by RtEs or received demands.

2.1.8.2 Bayesian Network ROS Component

Bayesian Networks (BNs) [19] [20] are a (usually graphical) representation of joint probability distributions. Individual nodes on the graph represent factorizations of joint probability distributions e.g. joint probability P(E,A,B) could be represented by having nodes representing P(E|A,B), P(A|B) and P(B) for abstract effect E and causes A and B. Directional arrows connect nodes and represent the (probabilistic) causal dependency between nodes. This is illustrated as an abstract Bayesian network in Figure 25.

Figure 25: Abstract BN Example

BNs are very flexible in their usage; for our purposes, they can be used as (input to) monitor/analysis, by probabilistically estimating which state a system (or MRS) is currently in, based on related sensor input. This effectively evaluates the network in a forward direction along the arrows, based on causal conditions observed at runtime.

2.1.8.3 SafeML ROS Component

SafeML is relevant when the host application uses ML models. It can be deployed as a monitor which checks whether the runtime input data received by the ML model is statistically 'similar'[26] as (a subset of) the training data. If this check fails, it indicates that the ML model might be trying to provide an answer for a situation it has not been trained for, and might therefore be unreliable. An abstract example of how this process can be applied is shown in Figure 26. In the figure, input images received by the robot's embedded camera are provided to an ML-based detector for some task e.g. object detection. SafeML compares the input images with a set derived from the model's training images. The more dissimilar the input to the reference images, the lower the confidence in the ML's outcome. Different levels of confidence can then be associated with corresponding responses e.g. performing a minimal risk manoeuvre, notifying human operators etc.

[26] Is likely to belong to the same probability distribution.

Figure 26: Abstract Example of SafeML for Object Detection/Localization

2.1.8.4 EDDI Tailorability Support Tool for ROS Platform

In D5.4 [21], the MROS meta-control framework [22] is used to tailor EDDIs to MRS that use the Robot Operating System (ROS) platform. MROS can be used to automatically generate models of an existing ROS architecture i.e. an MRS' architecture. These models can then be further exploited by associated tools to generate appropriate configuration files for the EDDI runtime generators and the components themselves.

An overview of this process can be seen in Figure 27. Blue parallelograms correspond to process steps, and green rectangles to input/output files. Starting from the left side, MROS is applied to an existing ROS application, generating a 'ROSSystem Definition File' (RDF). An example of an RDF can be seen in Figure 28. For each EDDI component that is intended to be used, a corresponding RDF must also be created (currently, manually). The graphical modelling tool Eclipse Sirius[27] and MROS can be used to combine the set of RDFs into a combined application which integrates the EDDI runtime components into the ROS application. At the end of this step, a combined RDF is available. Finally, using the latter RDF, a mapping between the EDDI runtime components and the ROS application environment can be created, yielding an EDDI configuration file for each of the EDDI runtime components.

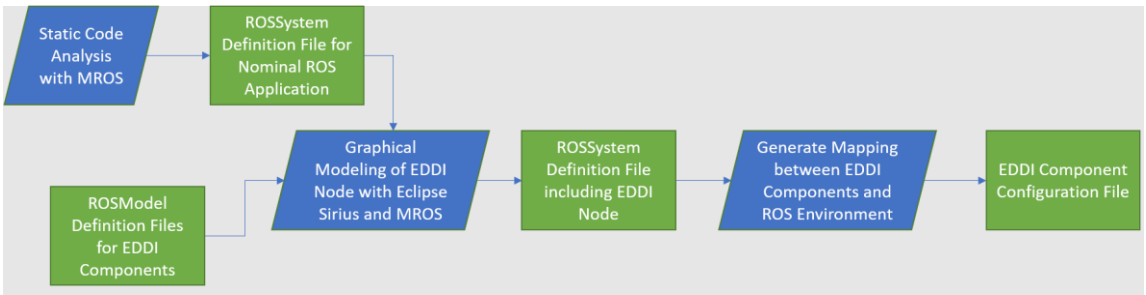


Figure 27: Overview of EDDI Tailorability for ROS Applications

[27] https://www.eclipse.org/sirius/

```
RosSystem { Name 'system_a'  RosComponents (
 ComponentInterface { name move_base
  RosParameters{
   RosParameter 'max_vel_x' { value 0.5 },
   RosParameter 'max_vel_y' { value 0.5 },
   RosParameter 'inflation_radius' { value 0.5 },
   RosParameter 'observation_sources' { value scan }}})
 Parameters {
  Parameter { name 'qa_safety' type Double value 0.41 },
  Parameter { name 'qa_energy' type Double value 0.48 }}}
```

Figure 28: Example of ROSSystem Definition File, from [22]

2.1.8.5 Requirements status

Table 9: Requirements status for EDDI components and generation

Req. No.	Requirement	Priority	Resp. WP	Status
D123	Support generation of ConSerts for dynamic risk assessment and management (and associated executable components) from EDDIs and ExSces.	SHALL	7	Done
D124	Support generation of Bayesian Networks for dynamic risk assessment and management (and associated executable components) from DDIs and ExSces.	SHALL	7	Done
D125	Support runtime estimation of uncertainty for ML components.	SHALL	7	Done
D126	Support distributed execution of real-time analytics and monitoring facilities.	SHALL	7	Done
D127	Support probabilistic reasoning for dynamic risk assessment.	SHOULD	7	Done
D128	Provide interface for data exchange to/from EDDIs.	SHALL	7	Done
D129	Provide interface for data exchange to/from runtime models.	SHALL	7	Done
D131	Bilateral interface or exchange format to allow information to be communicated with EDDI Tools.	SHALL	4, 5	Done
D132	Bilateral interface or exchange format to allow information to be communicated with ExSce Workbench.	SHALL	3	Done
D133	Bilateral interface or exchange format to allow information to be communicated with Trajectory Planning.	SHALL	2	Done
D134	Bilateral interface or exchange format to allow information to be communicated with Simulation-based Testing Tools.	SHALL	6	Done

2.1.9 Multi-Agent System for Security and Safety Management

By viewing MRS as Multi-Agent Systems (MAS), we can consider how the role of each changes dynamically as the mission context changes (usually with respect to system and operational context changes). Figure 29 depicts an abstract overview of how the MAS viewpoint relates to our EDDI runtime components. In the case of the figure, the ConSert, Dynamic Risk Assessment (DRA), and SafeML components are depicted, but this is arbitrary; other combinations are also possible. Additionally, the figure indicates

that our initial focus is on supporting ROS architectures, but the approach can be adapted to support other platforms.

Under this view, each MRS plays the role of an agent, providing either mission-level monitoring, analysis, and/or planning, and then circulating their observations to the rest in each iteration. The MAS roles can be interchangeable, and allocated according to the specific application. Arriving at this vision is still work-in-progress and technically depends on the individual EDDI runtime components being discussed in this document.

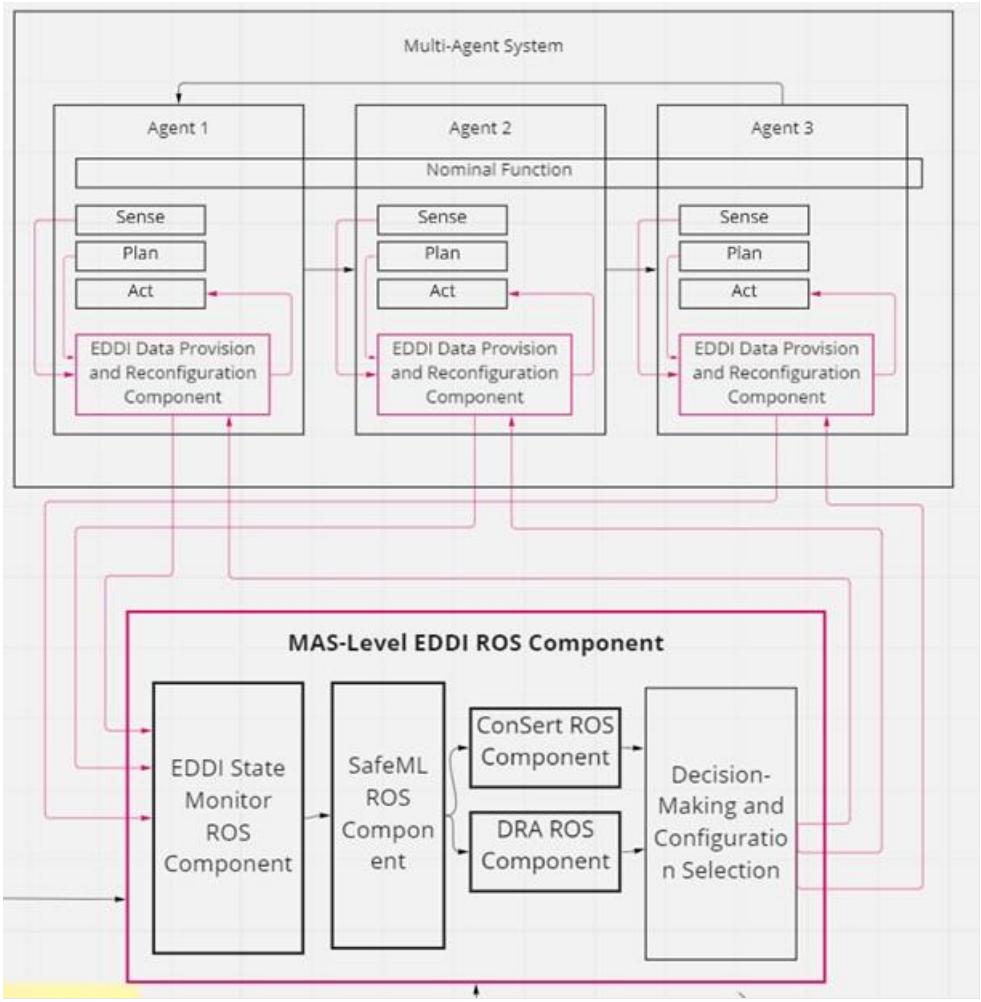


Figure 29: Mapping from MAS architecture to MAPE-K control loop

Table 10: Requirement status for multi agent system for security and safety

Req. No.	Requirement	Priority	Resp. WP	Status
D135	Support MAS-oriented deployment of dynamic risk management runtime models.	SHALL	7	Done
D136	Support MAS-oriented distributed execution of real-time analytics and monitoring facilities.	SHALL	7	*Done*
D137	Specification of secure protocol for MAS communication protocol.	SHALL	7	Partial
D138	Provide explainable feedback regarding the MRS and their operation.	SHALL	7	Done
D139	Provide situation-aware risk prediction service for MRS constituents.	SHOULD	7	Done

Req. No.	Requirement	Priority	Resp. WP	Status
D140	Support MRS resilience via individual agent role adaptation.	SHOULD	7	Done
D141	Support risk-aware coordination via risk source inference and collaborative risk management.	SHOULD	7	Done
D142	Support MRS resilience via MRS-wide adaptation.	MAY	7	Done
D143	Support MRS operation optimization with respect to dependability and mission efficiency.	MAY	7	Not done
D144	The MRS perception stack provides information regarding the MRS operational context.	SHOULD	2	*Done*
D145	The MRS perception stack provides information regarding the MRS planned tasks.	SHOULD	2	Done
D146	Safety models regarding the MRS are available at runtime from EDDI-based Safety Tools.	SHOULD	4	Done This is supported from the tool side, models need to be created for the use cases.
D147	Security models regarding the MRS are available at runtime from EDDI-based Security Tools.	SHOULD	5	Done
D148	Quality models regarding the MRS are available at runtime from Simulation-based Testing and ML Testing Tools.	SHOULD	6	Done
D149	MRS task planning, operational environment, and runtime capability models are available at runtime from Trajectory Planning.	SHOULD	2	Done From the trajectory planner, the trajectory and the actuator commands to follow that trajectory are provided. this is ready as shown in the simulations and experiments, for validation.
D150	MRS task planning, operational environment, and runtime capability models are available at runtime from Collaborative Perception.	SHOULD	2	Partial support
D151	MRS task planning, operational environment, and runtime capability models are available at runtime from Collaborative Intelligence Analytics.	SHOULD	2	Partial support

3. WORKFLOWS

Studying the SESAME tools and components, we have analysed the lifecycle of MRS from design, over development and up to use and identified several steps. The high-level, simplified workflow can be seen in Figure 30. However, the presented workflow can be instantiated in a flexible manner allowing different iterations.

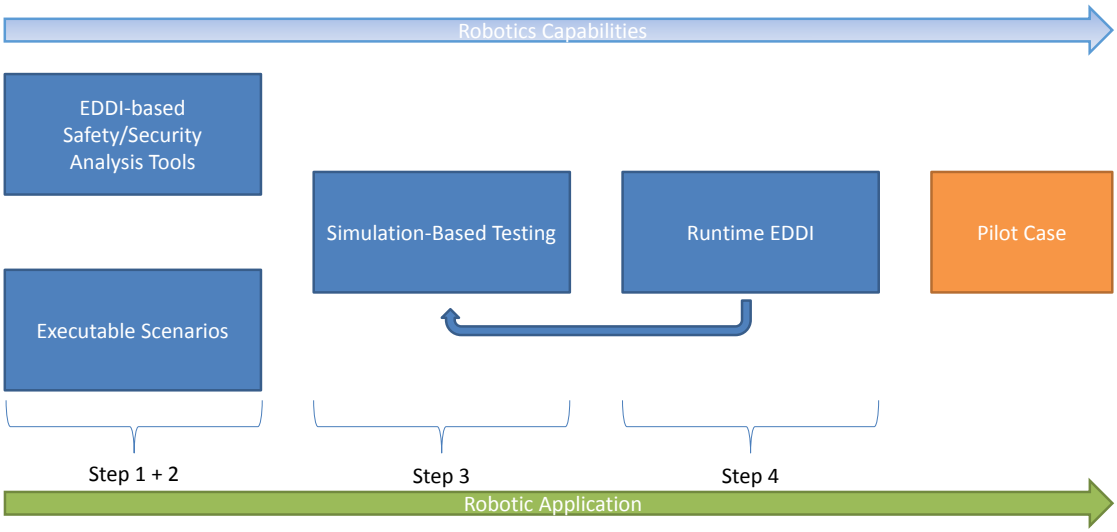


Figure 30: SESAME high-level Workflow

As we have described in D8.1 – Architectural Guidelines [2], the various SESAME components can be employed for the systematic development of dependable MRS. The workflow above supports this systematic development as follows:

- **Step 1**: Robotic users utilise configurable and extensible domain-specific languages to specify ExSce for MRS capturing both MRS mission-relevant information and mission-plausible information. This step is informed by the knowledge base of SESAME through the set of MRS capabilities available for the specific mission and ExSce templates.
- **Step 2**: Safety and security engineers, forming the dependability team, use ExSce to develop the EDDI, potentially leveraging EDDI templates from the knowledge base. This iterative safety-security co-engineering process enables establishing desirable tradeoffs between safety and security requirements, and other dependability features encapsulated within the EDDI. Once completed, a set of MRS mission-specific EDDI is generated.
- **Step 3**: Given the ExSce and EDDI, this step semi-automatically generates the EDDI-enabled MRS along with models, runtime monitors and interfaces needed for simulation and digital twin instantiation, quality assurance, and self-adaptation.
- **Step 4**: Through a systematic quality assurance process both in simulation and real-world settings, the EDDI- enabled MRS is evaluated. This process assesses the quality of the EDDI and ExSce, and the adequacy of AI-based perception components generating the data consumed by the EDDI. Depending on the outcomes, the ExSce or EDDI might require enrichment or refinement, leading to another design-time SESAME iteration.

In a more detailed overview of the whole system of tools and components (see Figure 31) we can identify the inputs/outputs of each tool and component as well as the interactions between them.

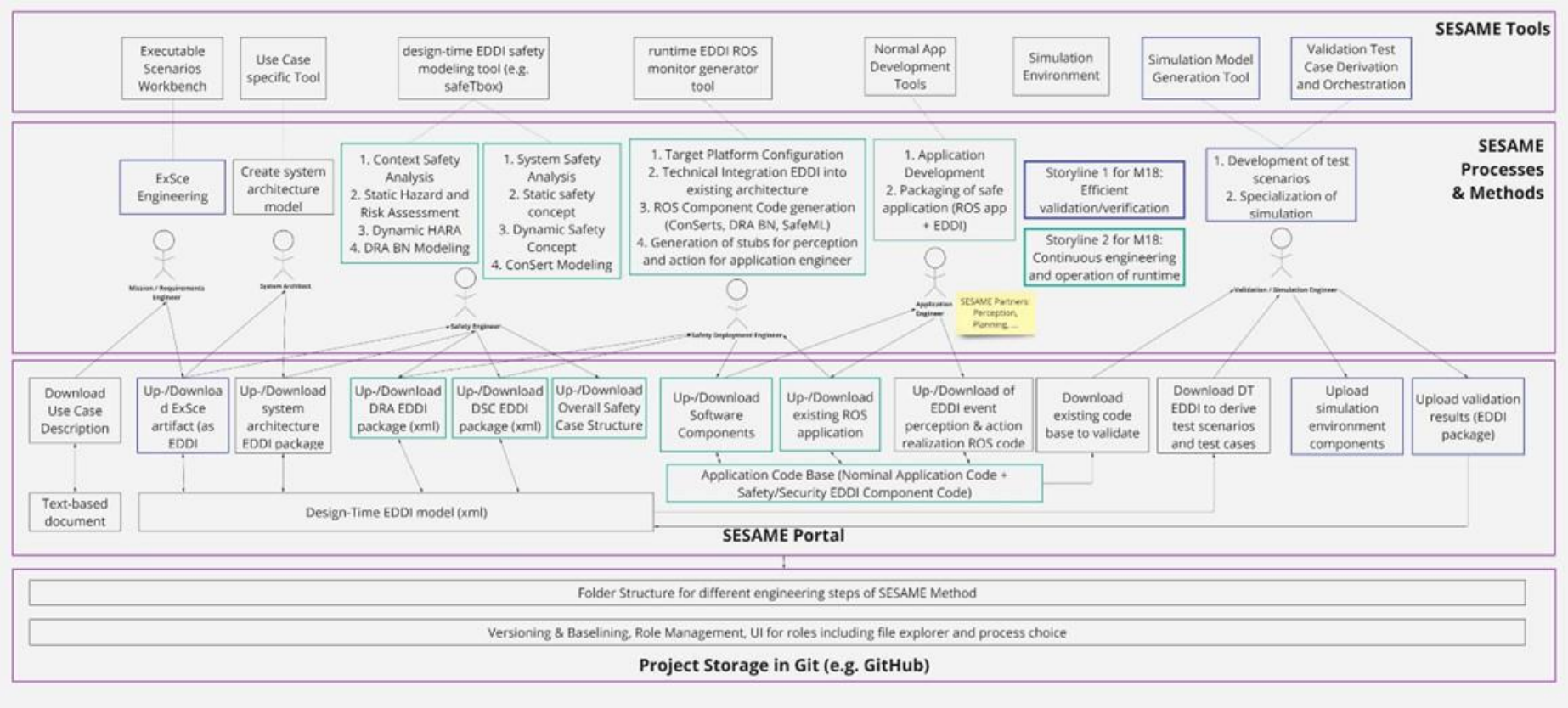


Figure 31: SESAME detailed workflow

4. SESAME INTEGRATION PLATFORM

The SESAME integration platform consolidates the software and tool developments done in the work packages (WP 2-7), and gives support to the demonstrator use-cases. The integrated platform will provide centralized access to all the technologies that produce during the project for developing dependable MRS and makes it readily available to the engineers, essentially being the central connector between the interconnected components.

4.1 TOOLS AND TECHNOLOGIES

The development of the integrated platform is carried out in a stepwise and iterative way. Several technologies and tools are needed to support the development of the platform and the integration of all provided models and artifacts of the components. Table 11 describes the selection of tools and technologies that are used in the scope of the platform development.

Table 11 - Tools and technologies to support the integrated-platform development

Description	Tool	Link
Version control (common source code repository and project storage repository)	Git	http://git-scm.com/
IDE	IntelliJ IDEA	https://www.jetbrains.com/idea/
Front-side programming language	JavaScript - Angular	https://angular.io/start
Back-end runtime environment	Node.js - Express.js	https://expressjs.com/

As described in the SESAME detailed workflow (Figure 31), the SESAME portal offers user interface (UI) to interact with the SESAME tools though different processes and methods by providing the options for uploading and downloading the various models and artifacts of the different components or modules for the projects (use cases). In addition, in the backend, the Git is chosen as data storage where the folder structure is followed for different engineering step of the SESAME processes and methods of the use cases. The implementation of t this platform was required to integrate with Git as project storage and implement graphical interfaces to expose these functionalities to the developer.

The final version of the SESAME platform is implemented according to the technologies architecture depicted in Figure 32. The Angular framework of JavaScript is chosen for the front-end development because of its compelling features that include templating, two-way binding, modularization, RESTful API handling, dependency

injection, and so on. The front-end interacts with Git (e.g., GitHub) through the RESTful API backend server developed using Express framework for Node.js.

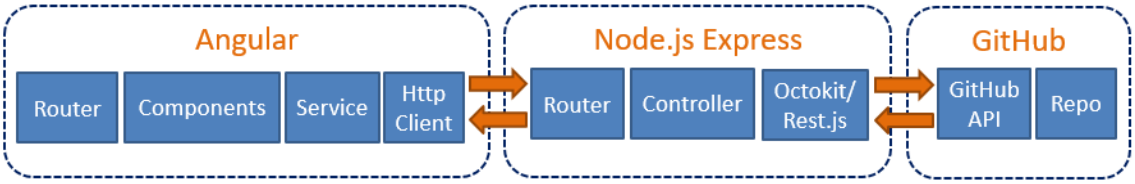


Figure 32: Front-end and back-end technologies of the SESAME platform

4.2 IMPLEMENTATIONS

The final implementation provides seamless navigation across three distinct pages, each serving a specific purpose.

Project Creation and Configuration:

The initial page, showcased in Figure 33 offers users the ability to create and configure projects. By clicking the 'Create Project' button in the upper-right corner, a pop-up window appears, allowing users to define a project name and select from models like EDDI or ExecSce.

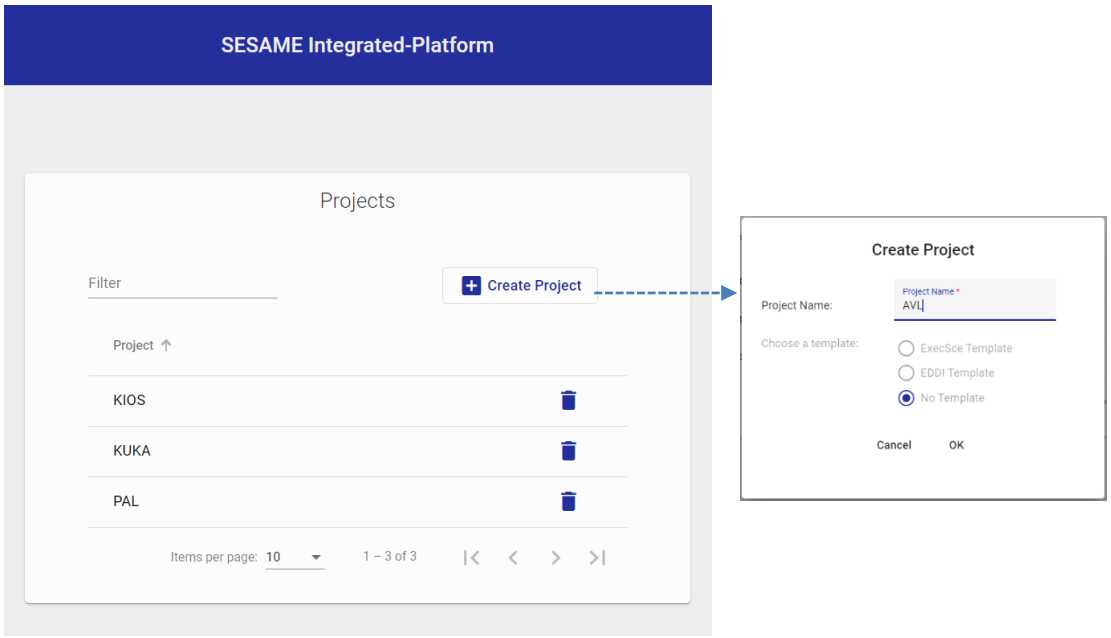


Figure 33: Project page (left-side) and create project pop-up window (right-side)

In the backend, a new directory is automatically created within our model storage Git repository for the new project. To enhance the user experience, we have incorporated features such as a project filter (located in the upper-left corner) and options for sorting projects in ascending or descending order at the top of the project table. Additionally, users have the ability to delete entire project.

Adding Modules/Components to Projects:

Clicking on a specific project redirects users to the second page, illustrated in Figure 34, which is designed for adding and removing modules/components to the selected project, and these modules are related to various work packages. The upper section provides a dropdown module list, enabling users to add the modules they wish to incorporate into their project. In the lower section, all existing modules are displayed in a table, where users can delete each module as needed. In the backend, adding a module creates a specific module directory within the project directory in our model storage Git repository.

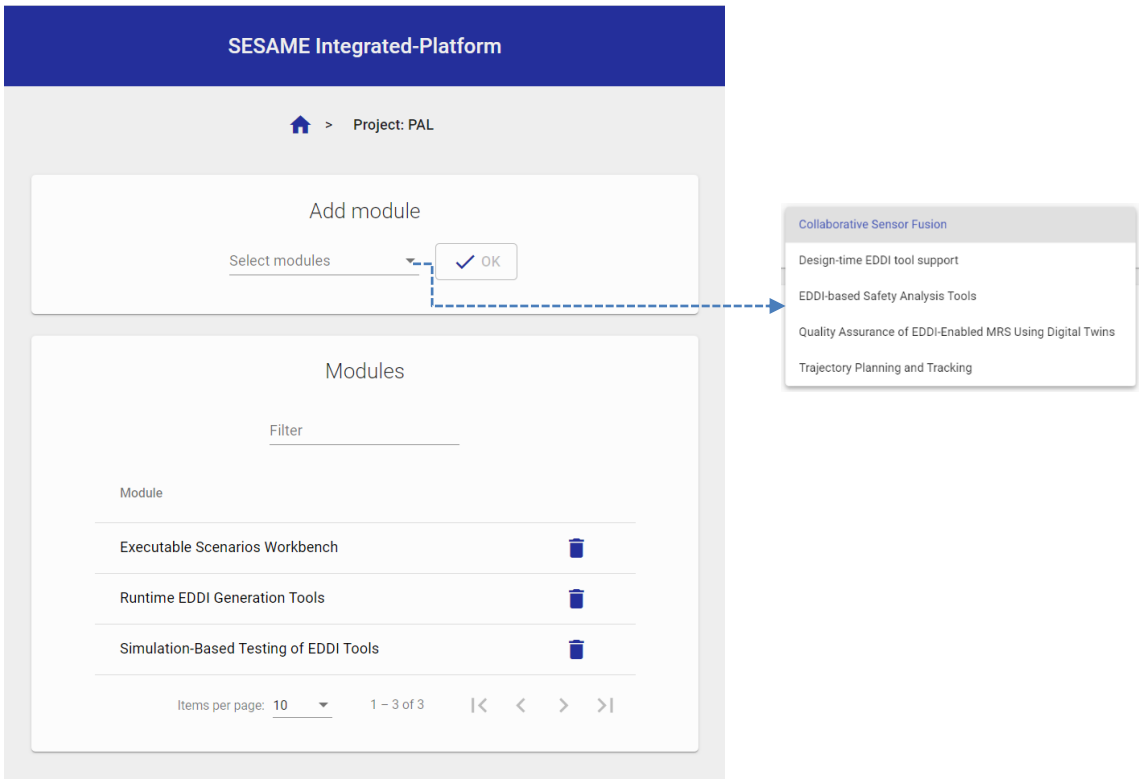


Figure 34: Module page (left side) and select modules list (right side)

Uploading and Downloading Models and Artifacts:

Clicking on a particular module redirects users to the third page, depicted in Figure 35. This page is dedicated to uploading new models and downloading existing ones. In the upper part, users can select the model or artifact files they want to upload. Furthermore, the system incorporates robust error-handling functionality, which is showcased on the right side of the figure, to capture and communicate file upload errors effectively. In the lower section, all existing models are displayed in a table, giving users the option to download or delete each model.

In the backend, uploading a model adds the model file to the specific module and project directory in the Git repository. This page, too, features a model filter and options for sorting in ascending or descending order to optimize the user experience.

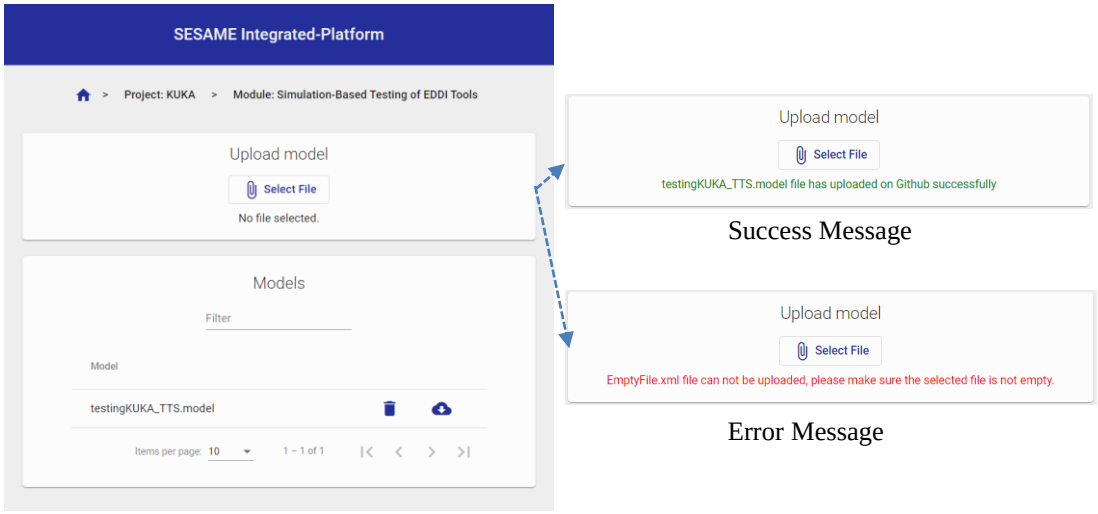


Figure 35: Upload model page (left-side), and file upload success/error message (right-side)

When uploading models to specific modules, various tool execution workflows (GitHub Action) will be triggered. Detailed descriptions of the tool integrations and workflow execution can be found in Section 4.3. Certain tool executions have specific prerequisites that must be met before commencing the execution process. For instance, when dealing with the 'Runtime EDDI Generation Tools', users are required to upload both processing and configuration files prior to uploading the model. These files are essential for the proper functioning of the tool and workflow execution. These prerequisites are clearly outlined on our 'Upload Model' page, as illustrated in Figure 36.

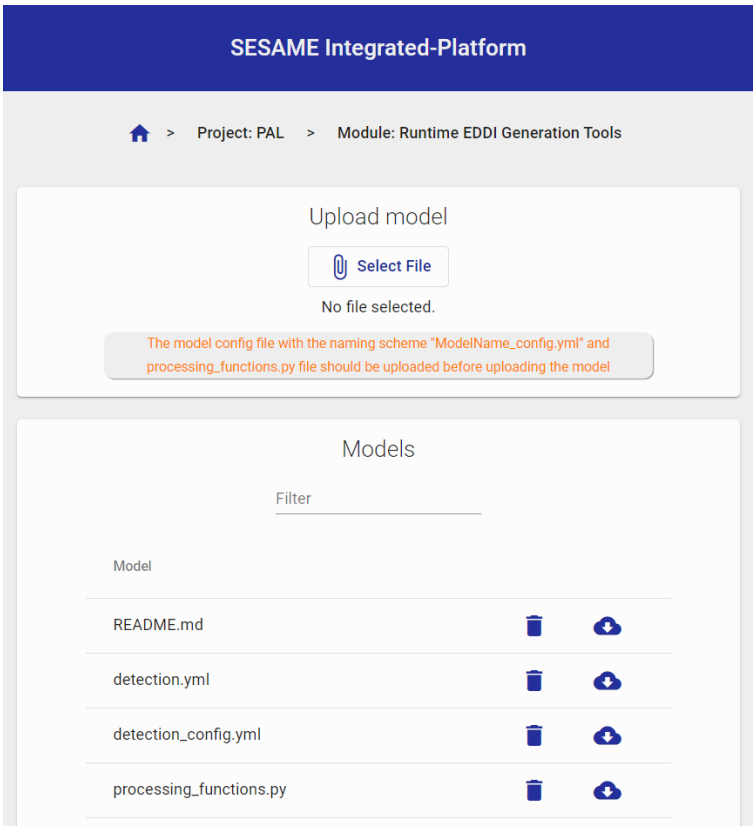


Figure 36: Upload model page with a model specific instructions

By adhering to these specified requirements and following the detailed tool execution instructions, users can ensure a smooth and efficient execution of developed GitHub Actions workflows for added models and utilizing the associated tools within the specific modules and projects.

4.3 INTEGRATION OF TOOLS

In this section, we will demonstrate the integration of tools across different components of the work packages into our integrated platform.

For tool integration, we have developed a versatile and modern continuous integration process leveraging GitHub Actions. GitHub Actions is a powerful feature that automates tasks and workflows within the GitHub repository. We have created workflows to automate tasks such as fetching the latest code from specific tool repositories, running the tools, and uploading the resulting outputs as artifacts for different tools. Furthermore, these workflows can be initiated by adding or uploading input models to various modules within the project on our platform. One can access all the developed workflows in our sesame-model-storage repository within the sesame-project[28]. The details of tool integration and workflow development are as follows.

FloorPlan-DSL Tools:

The FloorPlan-DSL, belonging to work package 3, serves as a modeling language tailored for indoor environments. It empowers users to craft 3D models and maps, instrumental in simulating robot navigation across various simulators. This tool offers the capability to create diverse environment variations and design simulation scenarios. One can find the repository for these tools at https://github.com/sesame-project/FloorPlan-DSL, and for comprehensive instructions on how to execute this tool, please refer to Section 6.4.2. To accommodate these functions of the tool, we have devised two distinct workflows, outlined as follows:

FloorPlan-DSL: Variation

Utilizing the Variation DSL, a complementary language, users can specify variation points for all spatial relations through probability distributions. The tool then generate concrete floor plans by sampling from these distributions.

[28] https://github.com/sesame-project/sesame-models-storage/tree/main/.github/workflows

To streamline the tool execution process, we have created a specialized workflow, as depicted in Figure 37. This workflow is triggered automatically once users upload a '.variation' model with the 'number of variation' suffix inside the 'Executable Scenarios Workbench' module within the project on our platform. Furthermore, it is important to note that in order to generate variations of the floorplan model, the floorplan model should be uploaded prior to the variation model.

```
# Checkout FloorPlan-DSL repo
- uses: actions/checkout@v3
  with:
    repository: sesame-project/FloorPlan-DSL
    token: ${{ secrets.ACCESS_TOKEN }}
    path: floorPlan-DSL-repo

- name: Set up Python
  uses: actions/setup-python@v3
  with:
    python-version: '3.8'

- name: Install requirements
  run: |
    cd $GITHUB_WORKSPACE/floorPlan-DSL-repo
    sudo apt-get update
    sudo apt-get install -y blender
    sudo apt-get install -y python3-pil
    pip install matplotlib numpy textX[cli]
    pip install -e .

- name: Run FloorPlan-DSL variation tool
  run: |
    mkdir output
    cd $GITHUB_WORKSPACE/models-repo
    SESAME_MODEL_FILE=$(basename "${SESAME_MODEL_PATH}")
    echo "model file: ${SESAME_MODEL_FILE}"
    variation_num=$(echo "${SESAME_MODEL_FILE}" | rev | cut -d'_' -f1 | rev | cut -d'.' -f1)
    echo "variations: $variation_num"
    textx generate $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_PATH }}" --target exsce-floorplan-dsl --variations "$variation_num" --output $GITHUB_WORKSPACE/output
    ls $GITHUB_WORKSPACE/output

- name: Upload Generated variations
  uses: actions/upload-artifact@v3
  with:
    name: Generated-variations
    path: output
```

Figure 37: A segment of the 'run-floorPlanDSL-variation-tool' workflow

The workflow begins with a trigger event, which occurs when a user upload variation model in the '.variation' file format to the designated module, 'Executable Scenarios Workbench' within the project environment. Upon triggering, the workflow initiates an input validation process. It checks if the model file is added and confirms that it adheres to the expected structure.

If the input passes validation, the workflow proceeds to next steps, as illustrated in Figure 37. In these steps, prior to running the variation tool, you must configure Python and Blender and install the requisite dependencies. Subsequently, execute the FloorPlan-DSL variation tool, which generates different floorplan models and saves them in the output folder. These generated floorplans are then uploaded as workflow artifacts for further utilization.

FloorPlan-DSL: Simulation Environment

This function facilitates the transformation of floor plan descriptions into 3D models in the widely recognized STL format, compatible with numerous simulators.

To streamline the execution process, we have formulated a dedicated workflow, visually depicted in Figure 38. This workflow is automatically activated when users upload a

‘.floorplan’ model within the ‘Executable Scenarios Workbench’ module of the project on our platform.

```
# checkout FloorPlan-DSL repo
 - uses: actions/checkout@v3
   with:
     repository: sesame-project/FloorPlan-DSL
     token: ${{ secrets.ACCESS_TOKEN }}
     path: floorPlan-DSL-repo

# blender 2.82a inbuilt has python 3.7 thus we are using this specific version (https://docs.blender.org/api/current/info_tips_and_tricks.html#bundled-python-extensions)
 - name: Set up Python
   uses: actions/setup-python@v3
   with:
     python-version: '3.7.17'

 - name: Install Blender
   run: |
     cd $GITHUB_WORKSPACE/floorPlan-DSL-repo
     wget https://download.blender.org/release/Blender2.82/blender-2.82a-linux64.tar.xz
     tar -xf blender-2.82a-linux64.tar.xz
     rm -rf blender-2.82a-linux64/2.82/python

 - name: Set Python environment variables
   run: |
     echo "PYTHONHOME=$RUNNER_TOOL_CACHE/Python/3.7.17/x64" >> $GITHUB_ENV

 - name: Install requirements
   run: |
     cd $GITHUB_WORKSPACE/floorPlan-DSL-repo
     sudo apt-get update
     sudo apt-get install -y python3-pip
     sudo apt-get install -y python3-pil
     pip install matplotlib numpy textX[cli] pyyaml
     pip install -e .

 - name: Run FloorPlan-DSL simulation tool
   run: |
     cd $GITHUB_WORKSPACE/floorPlan-DSL-repo
     blender-2.82a-linux64/blender --background --python src/exsce_floorplan/exsce_floorplan.py --python-use-system-env -- $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_PATH }}"

 - name: Upload Generated floorplan simulation
   uses: actions/upload-artifact@v3
   with:
     name: Generated-floorplan
     path: floorPlan-DSL-repo/output
```

Figure 38: A segment of the ‘run-floorPlanDSL-simulation-tool’ workflow

Similar to the *floorplan-DSL Variation* workflow, the workflow begins with a trigger event and checks if the model file is added then workflow proceeds. In the simulation tool execution, setting up Python and Blender and installing the necessary requirements is a prerequisite. Following this, you can execute the FloorPlan DSL simulation tool, which generates ‘.stl’ 3D models and saves them in the output folder as workflow artifacts, ready for further utilization.

Simulation-Based Testing Tool:

This tool is part of work package 6, and the initial phase code generator is seamlessly integrated into our platform. It is designed to generate metrics and experiment runners based on the model provided. The experiment runner is responsible for creating tests according to the strategy defined for the experiment's test campaign, and these tests are executed locally for thorough validation. One can find the repository for this tool at https://github.com/sesame-project/simulationBasedTesting, and a technical description of this process in Section 6.7.1.

A glimpse of the developed workflow for this tool is depicted in Figure 39, which is automatically triggered when users upload a ‘.model’ file to the ‘Simulation-Based Testing of EDDI tools’ module within their project.

```
# java set up
- uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '17'

# get release info using the tag and then using the release info download the zip file
- name: Download Asset zip file
  run: |
    tag="2023-08-28_39161dc8"
    echo "Downloading generator-$tag-release.zip ..."
    wget https://github.com/sesame-project/simulationBasedTesting/releases/download/$tag/generator-$tag-release.zip
    echo "tag=${tag}" >> "$GITHUB_ENV"

- name: Run Java Application
  run: |
    unzip "generator-${{ env.tag }}-release.zip"
    cd generator-${{ env.tag }}/uk.ac.york.sesame.testing.standalone.generator
    java -jar generator-${{ env.tag }}.jar $GITHUB_WORKSPACE/generator-${{ env.tag }}/ $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_PATH }}" $GITHUB_WORKSPACE/StandaloneGenTest

- name: Upload StandaloneGenTest java project with generated Java files for the experiment runner and metrics
  uses: actions/upload-artifact@v3
  with:
    name: Java-project-with-experimentRunner-metrics
    path: StandaloneGenTest
```

Figure 39: A segment of the 'run-simulationBasedTesting-tool' workflow

In this workflow, the tool execution process begins with setting up Java. Subsequently, we download the tool's assert zip file from the SimulationBasedTesting repository release page (https://github.com/sesame-project/simulationBasedTesting/releases). The next task involves unzipping the release file and running the Java application, resulting in the creation of a new Java project named 'standaloneGenTest' with generated Java files for both the experiment runner and metrics. Finally, this project is uploaded as artifacts for further model testing and validation.

Design-time EDDI Tools:

These tools are part of work package 7 and one can find the repository for these tools at https://github.com/sesame-project/design_time_eddis. It comprises three distinct tools: BayesianNetwork2EDDI, EDDI2BayesianNetwork, and eddis2consert. All these three tools are integrated within our platform. A detailed usage instructions and technical information provided into Section 6.9.

BayesianNetwork2EDDI

BayesianNetwork2EDDI is a versatile tool that facilitates the transformation of runtime Bayesian network models (in '.xdsl' format) into EDDI representations of the model. This enables the conversion of generated networks, be it through machine learning algorithms or expert knowledge, into interchangeable EDDIs.

To streamline this process, we have developed a workflow, a segment of the workflow is depicted in Figure 40. This workflow automatically triggers when users upload Bayesian network models in the '.xdsl' format to the module 'Design-Time EDDI Tool Supports' of the project interested in employing this tool.

```
- name: Check for added model file
  run: |
    if ("${{ env.SESAME_MODEL_PATH }}" -eq "") {
        Write-Output "No model file was added"
        exit 1
    }

# check out design-time-eddis repo
- uses: actions/checkout@v3
  with:
    repository: sesame-project/design_time_eddis
    token: ${{ secrets.ACCESS_TOKEN }}
    path: design-eddies-repo

- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: 3.8

- name: Install dependencies
  run: py -m pip install --upgrade pip

# java set up
- uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '17'

- name: Convert BayesianNetwork to EDDI
  run: |
    cd ${{ github.workspace }}\design-eddies-repo
    move .\tool_adapter\ode.tooladapter.jar .\BayesianNetwork2EDDI
    cd ${{ github.workspace }}\design-eddies-repo\BayesianNetwork2EDDI\licenses
    echo "${{ secrets.PYSMILE_LICENSE }}" > pysmile_license.base64
    certutil -decode pysmile_license.base64 pysmile_license.py
    cd ${{ github.workspace }}\design-eddies-repo\BayesianNetwork2EDDI
    java -jar ode.tooladapter.jar & pip install --index-url https://support.bayesfusion.com/pysmile-A/ pysmile
    pip install thrift==0.11.0
    dir licenses
    python .\xdsl_to_ddi.py -bn ..\..\models-repo\"${{ env.SESAME_MODEL_PATH }}" -out .\example\output.ddi

- name: Upload generated EDDI model for testing
  uses: actions/upload-artifact@v3
  with:
    name: generated-EDDIModel
    path: |
      ${{ github.workspace }}\design-eddies-repo\BayesianNetwork2EDDI\example\output.ddi
```

Figure 40: A segment of the 'transform-bayesianNetwork-to-eddi' workflow

The first transformation step involves checking out the latest version of the design-time-eddis repository. Subsequently, the workflow configures the necessary Java and Python environments. The next task involves the conversion of the Bayesian network into an EDDI model. Finally, the workflow concludes by uploading the newly generated EDDI model as an artifact, ready for further testing and utilization.

EDDI2BayesianNetwork

This tool allows the generation of runtime Bayesian network models for given an EDDI representation of the model. The runtime representation is needed for a subsequent generation of a Bayesian network runtime monitor.

The developed workflow for this transformation will automatically triggers when users upload an EDDI model with the 'BN' suffix in the model name and in either '.yml' or '.ddi' format to the 'Design-Time EDDI Tool Supports' module for the project interested in employing this tool. A segment of the workflow is depicted in Figure 41.

```
# check out design-time-eddis repo
- uses: actions/checkout@v3
  with:
    repository: sesame-project/design_time_eddis
    token: ${{ secrets.ACCESS_TOKEN }}
    path: design-eddies-repo

# java set up
- uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '17'

- name: Convert EDDI to BayesianNetwork
  run: |
    cd ${{ github.workspace }}\design-eddies-repo\EDDI2BayesianNetwork
    java -jar egl.jar -e egl\ddi_bn_to_xdsl.egl -m ..\models\generatedMergedODE.ecore -x ..\..\models-repo\"${{ env.SESAME_MODEL_PATH }}"
    mkdir bayesianNetworkModel
    move *.xdsl bayesianNetworkModel

- name: Upload generated Bayesian-network models for testing
  uses: actions/upload-artifact@v3
  with:
    name: generated-BN
    path: |
      ${{ github.workspace }}\design-eddies-repo\EDDI2BayesianNetwork\bayesianNetworkModel
```

Figure 41: A segment of the 'tranform-eddi-to-bayesianNetwork' workflow

Similar to the *BayesianNetwork2EDDI* workflow, the first transformation step involves checking out the latest version of the 'design-time-eddis' repository. Subsequently, the workflow configures the necessary Java environment, and the next task involves the conversion of the EDDI into Bayesian network model. Finally, the workflow concludes by uploading the newly generated Bayesian network model as an artifact, ready for further testing and utilization.

eddis2consert

This tool enable a convenient transformation of an EDDI '.xml' to ConSert '.yml(s)'. The transformed ConSert model will be the runtime representation, which is needed for a subsequent generation of a ConSert runtime monitor.

The workflow designed for this transformation will automatically activate when users upload an EDDI model with the 'consert' suffix in the model name, and the file is in either '.yml' or '.ddi' format within the 'Design-Time EDDI Tool Supports' module of the project. A portion of this workflow is illustrated in Figure 42.

```yaml
# check out design-time-eddis repo
- uses: actions/checkout@v3
  with:
    repository: sesame-project/design_time_eddis
    token: ${{ secrets.ACCESS_TOKEN }}
    path: design-eddies-repo

# java set up
- uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '17'

- name: Convert EDDI to Conserts
  run: |
    cd ${{ github.workspace }}\design-eddies-repo\eddi2consert
    java -jar egl.jar -e egl\model_to_yml.egl -m ..\models\generatedMergedODE.ecore -x ..\..\models-repo\"${{ env.SESAME_MODEL_PATH }}"
    mkdir consertModels
    move *.yml consertModels

- name: Upload generated consert model for testing
  uses: actions/upload-artifact@v3
  with:
    name: generated-conserts
    path: |
      ${{ github.workspace }}\design-eddies-repo\eddi2consert\consertModels
```

Figure 42: A segment of the 'tranform-eddi-to-conSerts' workflow

Here also, the transformation process starts with checking out the latest 'design_time_eddis' repository version, configuring Java environments. Then converting the EDDI model to a ConSert model, and concluding by uploading the newly generated ConSert model as an artifact for further testing and use.

Runtime-time EDDI Tools:

This tool is part of workpackage 7, and its repository (https://github.com/sesame-project/runtime_eddis) offers EDDI generation tools for Bayesian networks and ConSerts models, along with a ROS wrapper for integrating EDDI monitors in a ROS environment. Each model requires specific preparation steps, which are elaborated in Section 6.9. Both models have been successfully integrated into our platform.

We have consolidated the EDDI runtime generation for these models into a single workflow. In our platform, when users add or upload '.yml' or '.xdsl' models to the 'Runtime EDDI Generation Tools' module within their project, the workflow is automatically triggered. In addition, before uploading the model file, one need to ensure that the configuration and processing files are uploaded first. The actions in this workflow are tailored to the added model. A segment of this workflow is illustrated in Figure 43.

```
# check out the code generator (runtime_eddis) repo
- uses: actions/checkout@v3
  with:
    repository: sesame-project/runtime_eddis
    token: ${{ secrets.ACCESS_TOKEN }}
    path: generator-repo


- name: run consert monitor
  if: ${{ env.SESAME_MODEL_EXTENSION == 'yml' }}
  run: |
    cd $GITHUB_WORKSPACE/generator-repo/conserts
    sudo apt-get update
    sudo apt-get install wine
    wine conserts.exe compile --py -i $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_PATH }}"

- name: Install rust
  if: ${{ env.SESAME_MODEL_EXTENSION == 'yml' }}
  run: |
    cd $GITHUB_WORKSPACE/generator-repo
    sudo apt-get install -y curl build-essential
    sudo rm -rf /var/lib/apt/lists/*
    curl https://sh.rustup.rs -sSf | bash -s -- -y

- uses: PyO3/maturin-action@v1
  if: ${{  env.SESAME_MODEL_EXTENSION == 'yml' }}
  with:
    command: build
    args: --release -i python3.11
    maturin-version: 0.12.20
    working-directory: generator-repo/conserts/target/${{ env.SESAME_CONSERT_MODEL }}

- name: Install generated Wheel
  if: ${{ env.SESAME_MODEL_EXTENSION == 'yml' }}
  run: |
    cd $GITHUB_WORKSPACE/generator-repo/conserts/target/${{ env.SESAME_CONSERT_MODEL }}/target/wheels
    SESAME_WHEEL_FILE=$(ls *.whl)
    echo "Generated Wheel: $SESAME_WHEEL_FILE"
    pip install "$SESAME_WHEEL_FILE"

- name: Upload generated wheel for testing
  if: ${{ env.SESAME_MODEL_EXTENSION == 'yml' }}
  uses: actions/upload-artifact@v3
  with:
    name: generated-wheel
    path: generator-repo/conserts/target/${{ env.SESAME_CONSERT_MODEL }}/target/wheels

- name: Run the code generator
  run: |
    cd $GITHUB_WORKSPACE/generator-repo
    pip install pgmpy pyyaml
    mkdir -p catkin_ws/src
    python generator.py -w ./catkin_ws -c $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_CONFIG_PATH }}" -m $GITHUB_WORKSPACE/models-repo/"${{ env.SESAME_MODEL_PATH }}"

# code-generator generate EDDI monitor and the ROS node, which are uploaded as artifact for the testing
- name: Upload catkin_ws (Generated ROS node) for testing
  uses: actions/upload-artifact@v3
  with:
    name: EDDI-monitor-and-ROS-node
    path: generator-repo/catkin_ws
```

Figure 43: A segment of the 'execute-code-generator-runtimeEDDI' workflow

To execute the EDDI code generator, the first step is to check out the 'runtime_eddis' repository and perform Python setup. For the ConSert model, two outputs are generated: one for creating a wheel for the monitor and the other for generating ROS packages and EDDI ROS nodes. To create the ConSert monitor, it is necessary to install Rust and Maturin and generate the wheel, which is then uploaded for further testing. In the last two tasks, the code generator creates ROS packages and EDDI ROS nodes for both ConSert and Bayesian network models, with the generated output being uploaded for further testing.

5. CONTINUOUS INTEGRATION & DEPLOYMENT PROCESS

To ensure fast, reliable and continuous delivery of the latest updates to teams involved in testing and development of the SESAME tools, we designed and implemented a versatile and modern continuous integration and deployment process which is based on Github Actions[29].

5.1 OVERVIEW

The whole procedure consists of 3 stages as show in Figure 44. Initially, a developer pushes changes to the master branch of a SESAME Github repository. Our continuous integration infrastructure detects these changes and triggers a validation process, in order to confirm that everything works as expected without any issues. When validation finishes, all appropriate users receive an automated email that informs them about the validation result. The next action undertaken by the Continuous Integration Infrastructure is to create new versions for each tool, by building them using the updated source code.

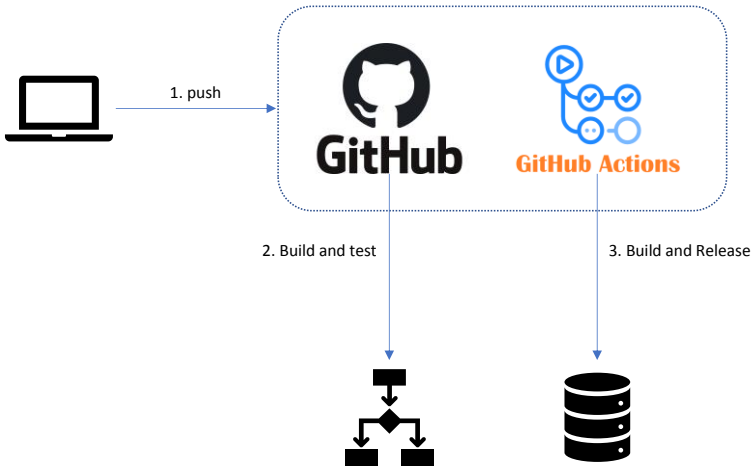


Figure 44: SESAME CI/CD workflow

Every SESAME tool follows a slightly different approach regarding the continuous integration and deployment process. However, for each tool the result will be a deployable / installable tool. Exchange of data (e.g. models) will be realised through the SESAME Integration Platform.

5.2 GITHUB INTEGRATION

Projects for the SESAME tools use GitHub repositories for the storage and versioning purposes of their source code. Our Continuous Integration infrastructure must be able to initiate the integration and deployment processes for each project as soon as any new source code is pushed to the master branch of a project repository. This requirement is fulfilled thanks to the GitHub web-hooks. They allow external services to be notified when certain events, related to a GitHub project, happen. In this case, when a new commit is pushed to the master branch, our Continuous Integration infrastructure receives a call to a specific endpoint and triggers the appropriate process. As soon as

[29] https://docs.github.com/en/actions

this process finishes, the corresponding project readme files are updated so that they display the latest build status of the project.

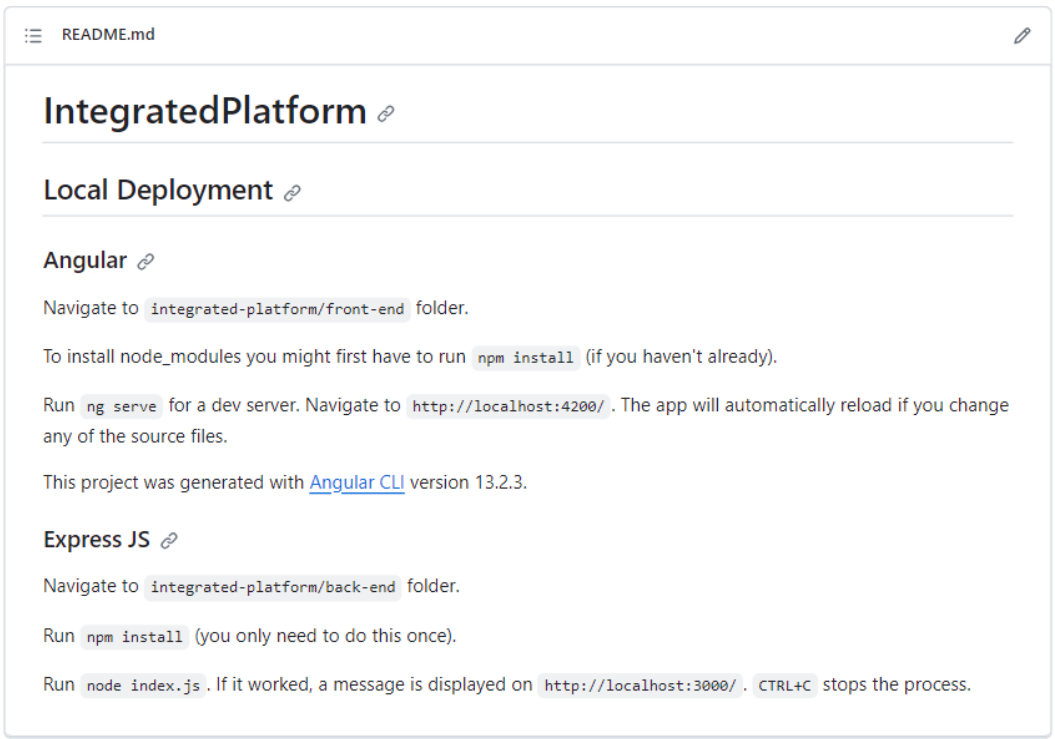
6. INSTALLATION AND CONFIGURATION OF SESAME TOOLS

This section presents a quick start guide for the installation and configuration of the SESAME components.

6.1 SESAME INTEGRATION PLATFORM

The SESAME Integration Platform serves as a web-based portal, encompasses a collection of SESAME components from various work packages. In Section 4, we showcase the functionality that has been implemented and seamlessly integrated into this web-based platform. This platform is exclusively accessible within the scope of the SESAME project, and you can access the repository for this platform via the following link: https://github.com/sesame-project/integration-platform

For comprehensive installation instructions, please refer the README.md file located within the integrated platform repository (Figure 45). The codebase is structured into two main parts: the front-end (Angular) and the back-end (Express JS). As per the provided instructions, users can initiate the backend server using the command 'node index.js', while the front-end Angular code can be launched with 'ng serve'. The integrated platform can be accessed via port 4200.


README.md

IntegratedPlatform

Local Deployment

Angular

Navigate to `integrated-platform/front-end` folder.

To install node_modules you might first have to run `npm install` (if you haven't already).

Run `ng serve` for a dev server. Navigate to `http://localhost:4200/`. The app will automatically reload if you change any of the source files.

This project was generated with Angular CLI version 13.2.3.

Express JS

Navigate to `integrated-platform/back-end` folder.

Run `npm install` (you only need to do this once).

Run `node index.js`. If it worked, a message is displayed on `http://localhost:3000/`. `CTRL+C` stops the process.

Figure 45: A snapshot of the README.md file from the integrated platform code

As detailed in Section 4, the platform is interconnected with a GitHub repository, which serves as its backend. This repository is also available within the SESAME project's

domain, and you can find the repository link as follows: https://github.com/sesame-project/sesame-models-storage, Furthermore, the workflow for all the integrated tools can be located within the same repository: https://github.com/sesame-project/sesame-models-storage/tree/main/.github/workflows. One can make use of the integrated tools by simply uploading the model files to the designated modules within the project. To understand how to utilize these integrated tools within our platform in detailed, please refer to the instructions provided in Section 4.3.

6.2 COLLABORATIVE SENSOR FUSION

Catkin repository to store ROS packages for the development environment of the collaborative perception component.

Requirements

- Ubuntu 20.04 ROS Noetic
- Nvidia Jetson Xavier NX
- Intel RealSense D435i (Camera with IMU)

Dependencies

- External dependencies required.
- Install the YOLOv5, Open VINS, RealSense ROS and its dependencies from:

https://github.com/ultralytics/yolov5

https://github.com/rpng/open_vins

https://github.com/IntelRealSense/librealsense

https://github.com/IntelRealSense/realsense-ros

Installation

We assume users have the ROS workspace in the ~/catkin_ws folder link and you have installed all the dependencies.

Software

Clone the repository:

```
  $ cd ~/catkin_ws/src
  $ git clone https://github.com/sesame-
project/collaborative_perception.git
```

Compile the sofware:

```
  $ cd ~/catkin_ws/src
```

```
$ catkin_make
```

Launch:

To launch the collaborative perception, users need two bash terminals.

In first terminal, launch the RealSense D435i Camera and IMU launch file:

```
$ roslaunch realsense2_camera rs_d435_camera_with_imu.launch
```

In second terminal, launch collaborative perception file:

```
$ roslaunch collaborative_perception sensor_fusion.launch
```

6.3 TRAJECTORY PLANNING AND TRACKING

This component is provided as an algorithm, which is coded and installed on either onboard or offboard computers, given the information on

- information of the robot dynamics and kinematics and limitations (by BRSU)
- task plans (either as waypoints or higher-level tasks. in latter case, the higher level planner breaks down the tasks into waypoints) (by BRSU)
- Sensing and perception information (by LU)
- Metrics (by FHF)

Then, the coded program runs at real time which provides the following:

- Planned trajectory
- Robot commands

The installation of these components is to install the required numerical solver (e.g. C++ or MATLAB) suitable for the given use-case with the supporting optimization packages (e.g. ipopt or fmincon). Then the given inputs are fed into the components. Finally, the expected outputs are sent out the corresponding component. In this case, the robot commands are sent to the corresponding MRS member's actuators.

6.4 EXECUTABLE SCENARIOS WORKBENCH

6.4.1 Bdd-dsl

1. Clone the repository: https://github.com/hbrs-sesame/bdd-dsl
2. Install the Python dependencies:

```
pip install -r requirements.txt
```

3. Install the Python package:

```
pip3 install -e .
```

6.4.2 Floor-Plan-DSL

The complete and up-to-date installation guide is available on the Github repository. A short summary is as follows:

Installation through Docker

1. Install docker
2. In the root directory of the repo

```
docker build . --tag floorplan:latest
```

3. Run the container

```
docker run -v $<local output folder>:/usr/src/app/output  \
  -v $<local input folder>:/usr/src/app/models -it floorplan:latest
bash
```

Native Installation

4. Install Blender v2.82a (preferably with apt-get)
5. Clone the repository: https://github.com/sesame-project/FloorPlan-DSL
6. Install the Python dependencies:

```
pip3 install -r requirements.txt
```

Usage

To generate the example model, from the root folder of the Floor Plan DSL, run the command:

```
blender --background --python exsce_floorplan/exsce_floorplan.py --
python-use-system-env -- models/hospital.floorplan
```

6.4.3 kindyngen

The repositoryhttps://github.com/comp-rob2b/modelling-tools contains the concrete instructions to install and execute the tools[30]. Here we provide a short summary:

1. Clone the repository: https://github.com/comp-rob2b/modelling-toolshttps://github.com/comp-rob2b/kindynsynInstall the requirements: rdflib, pySHACL, numpy
2. Install the Python package:

```
pip3 install -e .
```

[30] https://github.com/hbrs-sesame/kindyngen/blob/main/docs/installation.md

6.5 EDDI-BASED SAFETY ANALYSIS TOOLS

6.5.1 Installation

6.5.1.1 HiP-HOPS

HiP-HOPS requires Microsoft Windows (Win 7 or higher; 64bit required) and Matlab Simulink.

1. Obtain and install Matlab Simulink from the Mathworks website: https://www.mathworks.com/products/simulink.html

2. Download HiP-HOPS from the Downloads section of the HiP-HOPS website: https://hip-hops.co.uk/post/ Note that the most recent release is at the top. For use with Simulink, either the Evaluation or Commercial versions are required. The Evaluation version is free but limited to 10 components in the model; the Commercial version requires a licence key.

3. Install HiP-HOPS using the downloaded installer (further instructions can be found in section 3 of the manual). Make a note of the installation location for step 4).

4. To run HiP-HOPS from within Matlab Simulink, add the HiP-HOPS failure editor folder to the path by entering the Set Path menu (File → Set Path).

5. With the path added, the HiP-HOPS interface can be launched by typing "`hiphops`" into the Matlab command prompt. Again, further details are in section 4 of the HiP-HOPS manual.

6.5.1.2 SafeTbox

Working with safeTbox requires using the Microsoft Windows operating system (Windows 7 or later, Windows 10 recommended), .NET Framework v4.5 or higher, **Enterprise Architect version 13 to 15.1**, and the latest Java Runtime Environment. It is also required that the user has administrator rights for installation.

To start working with safeTbox, the user must:

1. Download and install Enterprise Architect; see their official website for e.g. version 15.1 (registration required). A 30-day trial license can be used with safeTbox.

2. Navigate to www.safetbox.de/registration and register.

3. Once registered, navigate to www.safetbox.de/downloads.

    a. Download the current release.

    b. Download a trial license.

4. Execute the release installer, follow instructions and install safeTbox.

5. Launch Enterprise Architect and use the downloaded licence file to activate safeTbox.

6. Further guidance is provided in the user manual; this is accessible both from the tool itself and on the website at https://safetbox.de/docu-samples (an example model is also available there).

6.5.1.3 BayesFusion GeNIe Modeler

To use GeNIe Modeler, the user must:

1. Install the respective version of GeNIe Modeler as described on the official website[31]. Notice that you must decide between the business[32] and the academic[33] version. So, navigate to https://www.bayesfusion.com/downloads/.

    a. If an academic version is desired, navigate to https://download.bayesfusion.com/files.html?category=Academia and download Genie academic version (compatible with Windows, can be used in MacOS and Linux with emulation software e.g. Wine)

    b. The business version requires purchasing a license by contacting Bayesfusion. 30-day trial versions are available at https://download.bayesfusion.com/files.html?category=Business

2. Execute the installer (in the emulated environment, if applicable) and follow the instructions to install Genie Modeller.

3. Launch Genie Modeller for the first time.

4. Get a license for your GeNIe Modeler version after launching the application for the first time. For that, you just need to follow the dialog box that opened automatically.

    a. In case of a business license, you must have the individual license file provided by BayesFusion on hand.

5. Further guidance on creating BNs and using the Modeller can be found at https://support.bayesfusion.com/docs/.

6.5.1.4 SafeML

The SafeML package is written in Python and can be found on Github at https://github.com/ISorokos/SafeML. The following are the requirements for installing the SafeML library.

- Python (tested on versions > 3.5)
- Anaconda package installer
- (optional) Hardware accelerators (such as CUDA[34]-enabled GPUs)

[31] https://www.bayesfusion.com/downloads/
[32] https://download.bayesfusion.com/files.html?category=Business
[33] https://download.bayesfusion.com/files.html?category=Academia

The other necessary libraries can be downloaded and installed using the environment specification file. The library package is provided along with the *.yml* file that describes the anaconda/python environment with all the libraries required to run all the functions. Simply use

```
conda env create -f <environment file name>
conda activate <environment name>
```

Additionally, a distribution package can also be provided. For installing from distribution package, simply use the pip tool of python:

```
pip install safeml
```

The public release of the package is currently planned. Once released, the package can be directly installed from the pip library in python. In the meantime, the package can be made available on request from Fraunhofer IESE and/or the University of Hull. An older public version is available on Github[35].

6.5.1.5 SafeDrones

The SafeDrones package is built in Python and can be found on Github at https://github.com/Dependable-Intelligent-Systems-Lab/SafeDrones. The following are the requirements for installing SafeDrones library.

- Python (tested on versions > 3.5)
- Anaconda package installer

The other necessary libraries can be downloaded and installed using the environment specification file. The SafeDrone library is provided along with the *.yml* file that describes the anaconda/python environment with all the libraries required to run all the functions. Simply use

```
conda env create -f <environment file name>
conda activate <environment name>
```

6.5.2 Configuration

6.5.2.1 SafeTbox

To begin modelling a system, the user must:

1. Launch Enterprise Architect (EA).
2. See the safeTbox welcome screen (if not, try relaunching EA).

[34] GPU Accelerated Computing with Python | NVIDIA Developer
[35] https://github.com/ISorokos/SafeML/

3. On the left side of the application window, there should be a ‘Browse’ panel entry. Hover over it and once it opens, select ‘click to open new project’.

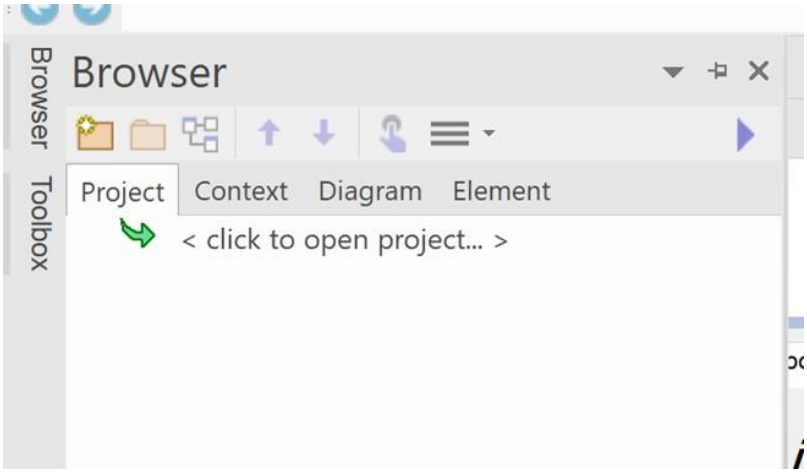


Figure 46: Open a new project menu

4. In the dialog, select the dropdown Local File => New Project…

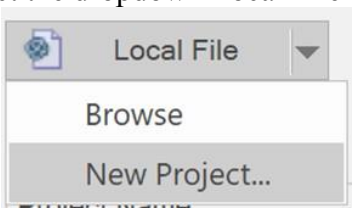


Figure 47: Select a new project menu

5. Select a file location and name for the new model file using the file dialog.

6. When ready, the Browser should feature a Model element.

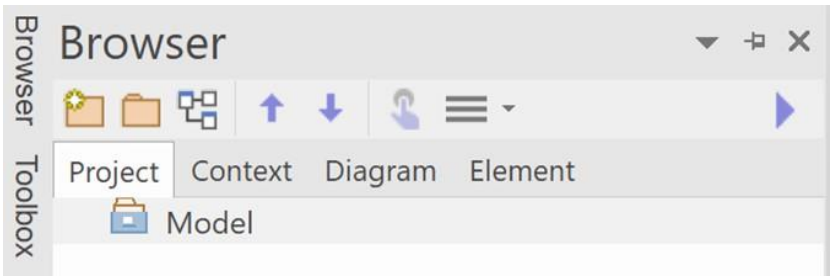


Figure 48: Model in the browser

7. Selecting Model and pressing Ctrl + Space on the keyboard should activate the context-sensitive safeTbox Smart Menu.

8. To create an architecture model, from the Smart Menu => Create => New Architecture Model.

    a. In the new diagram, use the toolbox to drag and drop elements or the Smart Menu => Create (after clicking on the diagram) to create new elements on the diagram. This logic applies to all other diagrams as well.

b. Selected element properties can be modified using Smart Menu => safeTbox Properties, or double-clicking on the element. This brings up a dialog with context-relevant properties. This applies to elements in other diagrams as well.

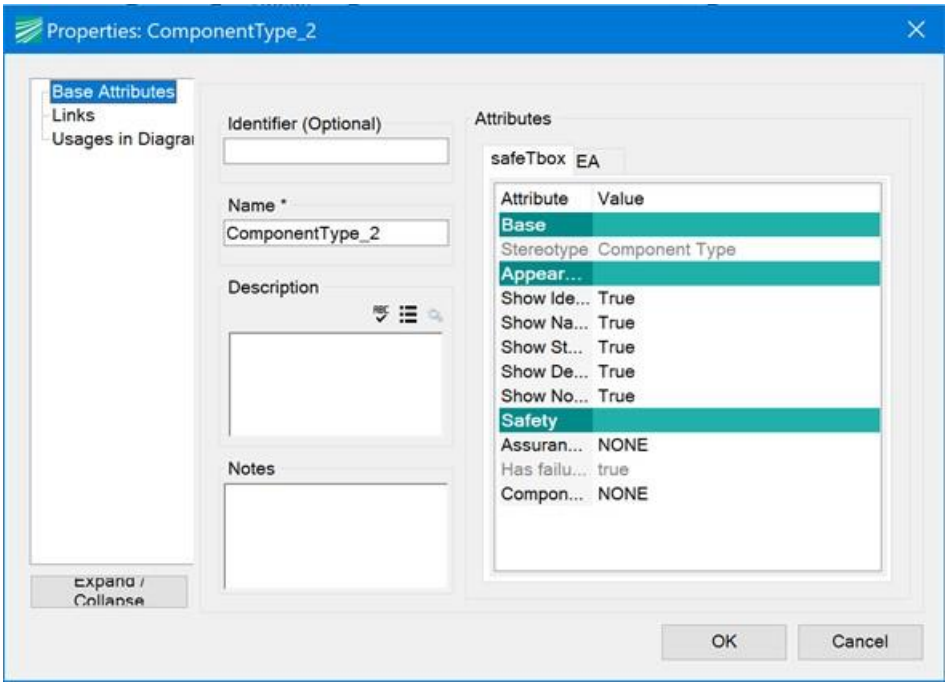


Figure 49: Context relevant properties dialog

c. Use subcomponents to model lower-level system/components. Subcomponents can be created by either making brand new components, or re-using previously defined components. Note that recursive hierarchies are not allowed (i.e. a chain of contained components cannot contain the same component twice).

d. Use Input/Output Ports to model corresponding connections between your component interface boundary and your component internal subcomponents.

e. Use connectors to link Input Ports to Output Ports. It is possible to select a port, and use the Quicklinker (the arrow icon) to immediately drag and drop a connector from the selected port to the target port.

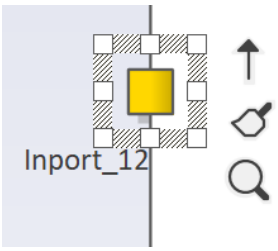


Figure 50: Using connectors to link Input Ports to Output Ports

9. To create a HARA model, select the Model in the (project) Browser, and use the Smart Menu => Create => New HARA Model. A spreadsheet editor should appear

in the main view. You can access the editor again by double-clicking on the HARA element in the project browser.

a. Fill in the sheets of the spreadsheet sequentially, starting with the Functions sheet. The previous sheets can be edited for documentation and information purposes. Add Functions by right-clicking on the row margin of the spreadsheet editor, or use the tab menu => safeTbox => Load existing component types.

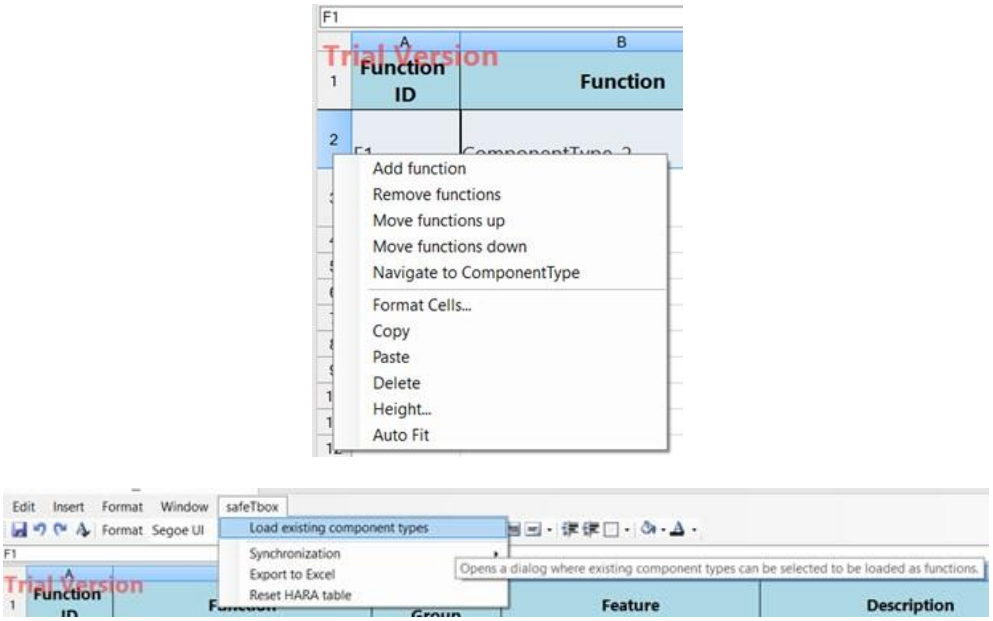


Figure 51: safeTbox spreadsheet editor (above) and tab menu (below)

b. In the FHA sheet, add entries by using the tab menu => safeTbox => Permutate functions with failure modes, selecting which keywords to consider in the dialog that appears. Alternatively, right-clicking on the row margin can also add entries manually. Hazards can also be added by right-clicking on the column margin of the rightmost columns.

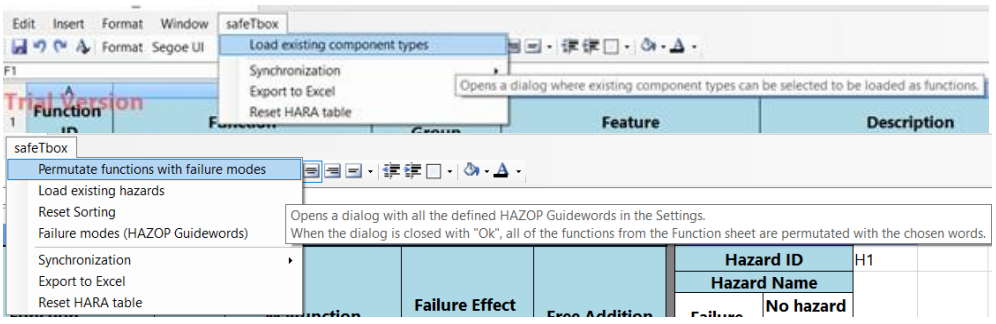


Figure 52: safeTbox menu tab

c. Right-click on the row margin or use the tab menu => safeTbox => Update situations risk to automatically add new Hazardous Events in either Situations sheet, based on the Hazards specified previously.

d. In the Risk Assessment sheet, use the tab menu => safeTbox => Update risk assessment sheet to automatically collect the Hazardous Events specified in the Situations sheets previously. Right-click under the Safety Goal columns to add new Safety Goals for specific Hazardous Event entries. This also applies for associating Assumptions.

10. To create a Component Fault Tree (CFT) for an existing component, select that component => Smart Menu => Add Failure Model.

    a. You can add Input/Output Failure modes to denote incoming/outgoing failure propagation from the current component's CFT.

    b. Logical Gates e.g. AND/OR represent how failure propagates.

    c. Basic Events represent component-internal sources of failure.

    d. CFTs of internal components can also be added, following the same principles and restrictions as subcomponents listed above for architecture models.

    e. To analyse a CFT, select a relevant element within it (e.g. an Output Failure Mode) => Smart Menu => Perform Analysis. Follow the dialog instructions and reach the analysis dialog. Analysis results can be saved for review at the bottom of the analysis dialog.

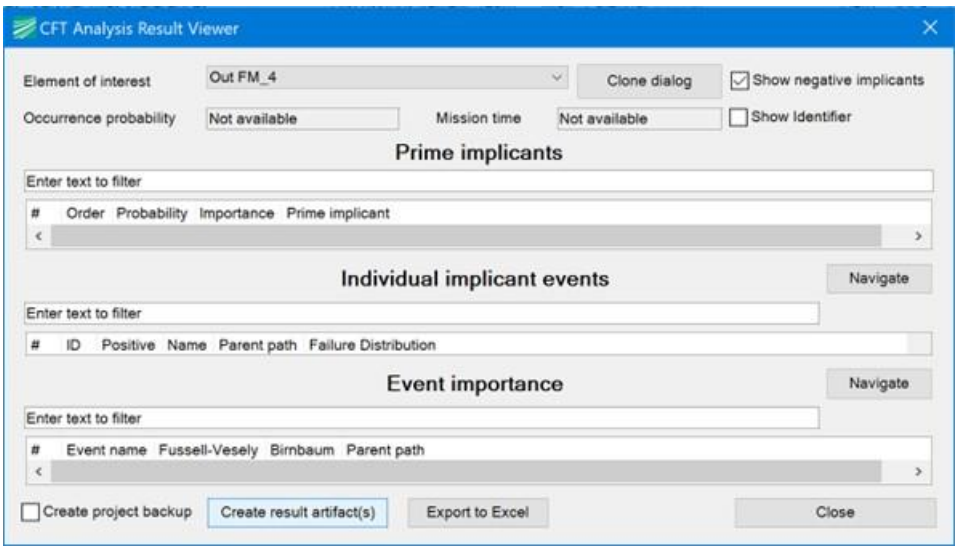


Figure 53: Analysis dialog

11. To create a GSN model, from the (Project) Browse panel => Smart Menu => Create => New Argumentation Module.

    a. Goals, Strategies, Solutions, and other GSN elements can be created per the GSN community standard[36].

[36] https://scsc.uk/r141C:1

b. Elements from foreign argument modules can be referenced as 'Away' elements e.g. Away Goals can reference Goals in other Modules.

12. To create a ConSert model, both a Collaborative System Group, and a Collaborative System must be created, also using the corresponding Smart Menu => Create options. Further guidance on how to model ConSerts is found in D4.4.

    a. Use Collaborative System Group => Smart Menu => Specify Service Types to open a set of forms that allow specification of the domain-level services possible.

    b. Use Collaborative System => Smart Menu => Specify Configurations and Services to open a set of forms that allow specification of a given system's configurations and services.

    c. Once the services have been specified for a Configuration, you can edit the configuration's ConSert logic by double-clicking on the .cst element in the (project) Browser, as seen in the example below (the bottom element seen in the Figure 54).

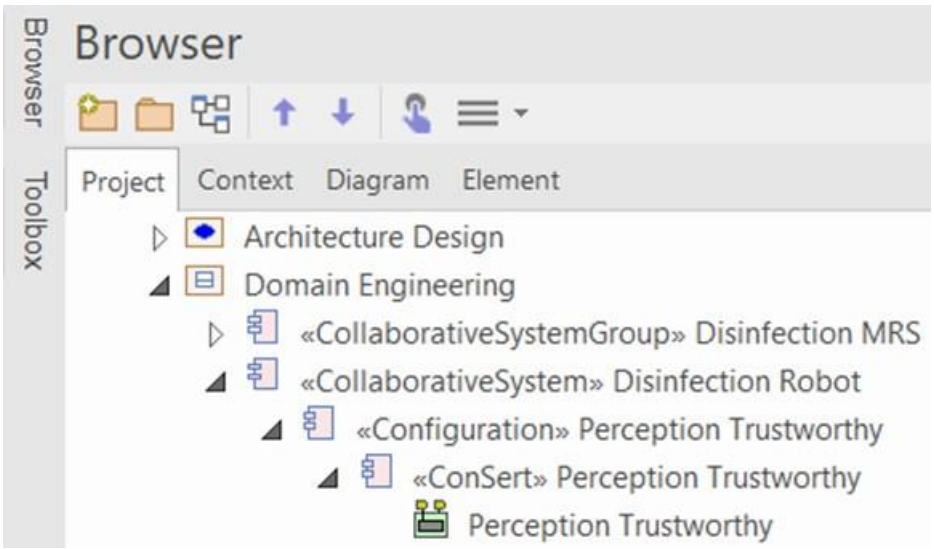


Figure 54: project browser example

    d. Use the Toolbox pane to drag and drop elements onto the ConSert model. When dropping something other than an AND/OR gate, select the corresponding services and service types you would like to map the dropped element to, using the dialogs that appear.

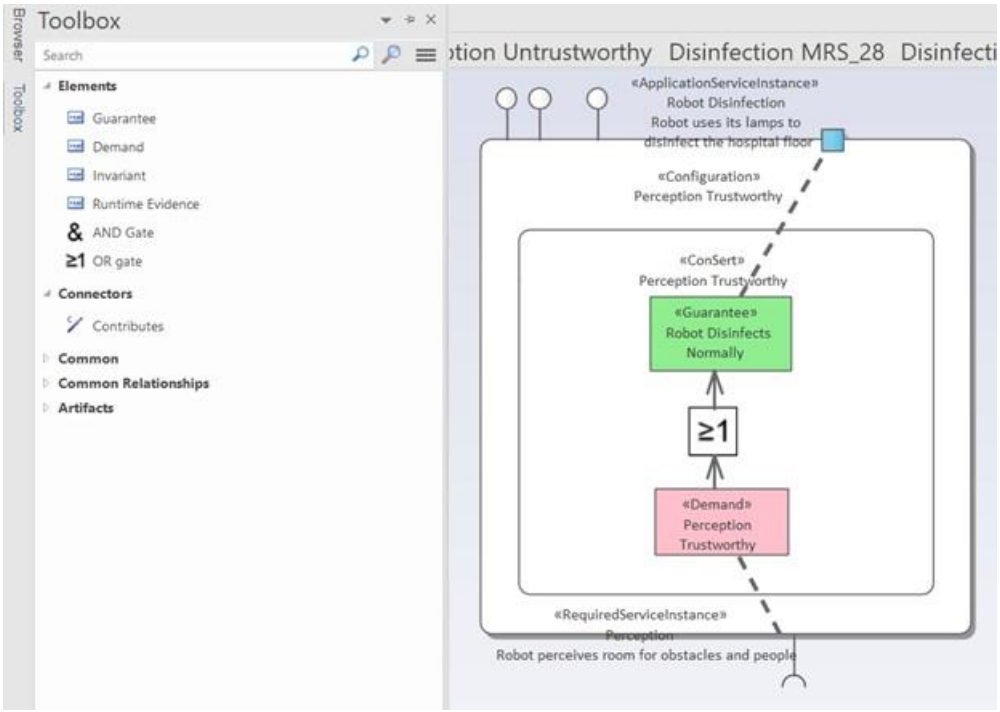


Figure 55: Toolbox pane to drag and drop elements onto the ConSert model

6.5.2.2 BayesFusion GeNIe Modeler

1. Install GeNIe Modeler as explained in Section 6.5.1.3. and launch GeNIe Modeler.

2. Now, GeNIe Modeler starts with an empty network like shown in Figure 56.

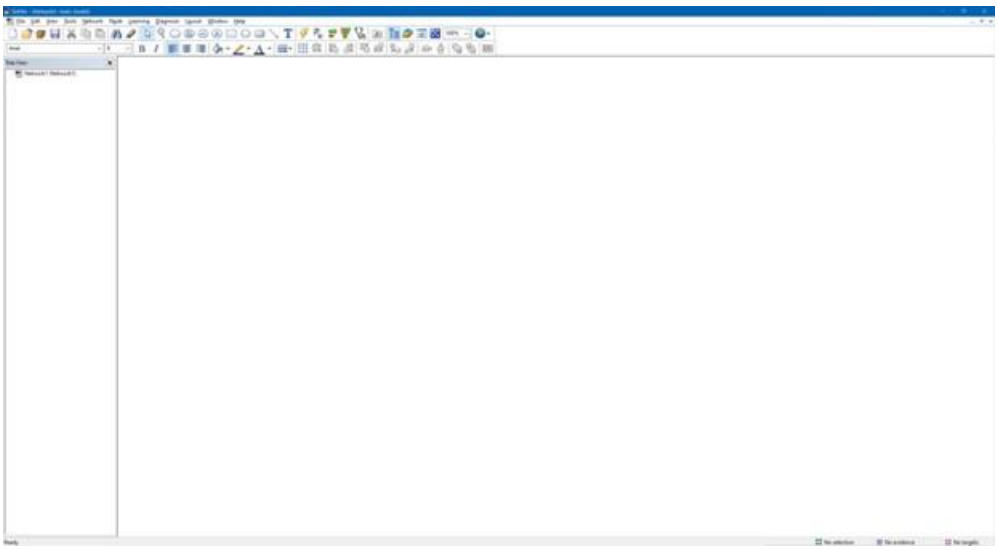


Figure 56: Initial screen of GeNIe Modeler

3. Now, you can start modeling a Bayesian network by adding nodes via the button in the topbar that symbolizes a network node (as shown in Figure 57). Hint: You have to click on the canvas after selecting the node button to create new nodes.

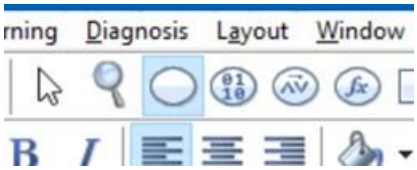


Figure 57: Button to select for creating new Bayesian network nodes

4. Similar, with the arc button (as shown in Figure 58), you can connect two nodes in the canvas to model a causal relationship.

Figure 58: Button to select for creating new causal relationships between two nodes

5. In the "Node properties" you can specify each node. You can open the "Node properties" dialog window by double clicking on a node on the canvas.
    a. In the "Definition" tab, you can define the concrete states of that node. You can rename them by double clicking on the node's states (left part in Figure 60). Via the buttons in the bar above (as shown in Figure 59), you can add or remove states.
    b. In the same "Definition" tab, you can define the conditional probability distribution in form of a table for that specific node. This is the concrete parameterization for this specific node, thus, it decides the outcomes when the Bayesian network is inferred given evidence. This table gives the probabilities for each state of the node given a state permutation over all parents of this node (as shown in Figure 60).

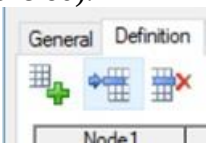


Figure 59: Buttons for adapting the number of states of a node

Node1	State0		State1	
Node3	State0	State1	State0	State1
State0	0.5	0.5	0.5	0.5
State1	0.5	0.5	0.5	0.5

Figure 60: Conditional probability table for a node that has two parents in an example Bayesian network

6. To test your Bayesian network, you can run inferences in GeNIe Modeler. The lightning button in the top bar updates all nodes in the network at once (as shown in Figure 61).
    a. To easier observe the outcomes of the nodes, you can change the view option on the canvas to include the states with their probability distribution. For this, mark all nodes and right click on them. Then, select the "View As" – "Bar Chart" option.
    b. To set specific evidence for one node, you can simply double click on the respective states when the "Bar Chart" view option is active.

Figure 61: Update button to run an inference over the network

7. For more details and concrete questions, we refer to the official user handbook[37] for GeNIe Modeler.

6.5.2.3 *SafeML*

For using SafeML in existing codebases, simply import the distance metrics from the library. An example is also available in the package provided. For importing the functions, use:

```
from safeml.core.ecdf_distance_measures import KuiperDistance, WassersteinDistance, KolmogorovSmirnovDistance, CramerVonMisesDistance, AndersonDarlingDistance, DTSDistance
```

The distance metrics can be initialised using

```
metric = WassersteinDistance()
```

Each type of distance metric, also supports several ways of computation. If the hardware accelerator is available, the corresponding function can be used.

```
pval, Dist = metric.measure_metric_p_value_gpu(X, Y)
```

Further information can be obtained from the `docs` folder provided with the package.

6.5.2.4 *SafeDrones*

The SafeDrones functions can be easily used by importing them from python. An example is also available in the package provided. For importing SafeDrones, use:

```
from SafeDrones.core.SafeDrones import SafeDrones
```

After importing, the SafeDrone evaluator can be initialised using

```
eval = SafeDrones()
```

For evaluating different types of risk and predicting failure, the corresponding functions can be used. For example, for motor failure risk prediction, the function:

```
Motor_Failure_Risk_Calc
```

can be used. The details of other functions can be obtained from the `docs` provided with the package.

6.5.2.5 *EDDI Editor*

The EDDI Editor is a standalone executable; to launch, simply run the executable. Windows 10 or above and an up-to-date .NET framework installation are required.

The main interface is displayed in Figure 62.

[37] https://support.bayesfusion.com/docs/GeNIe/

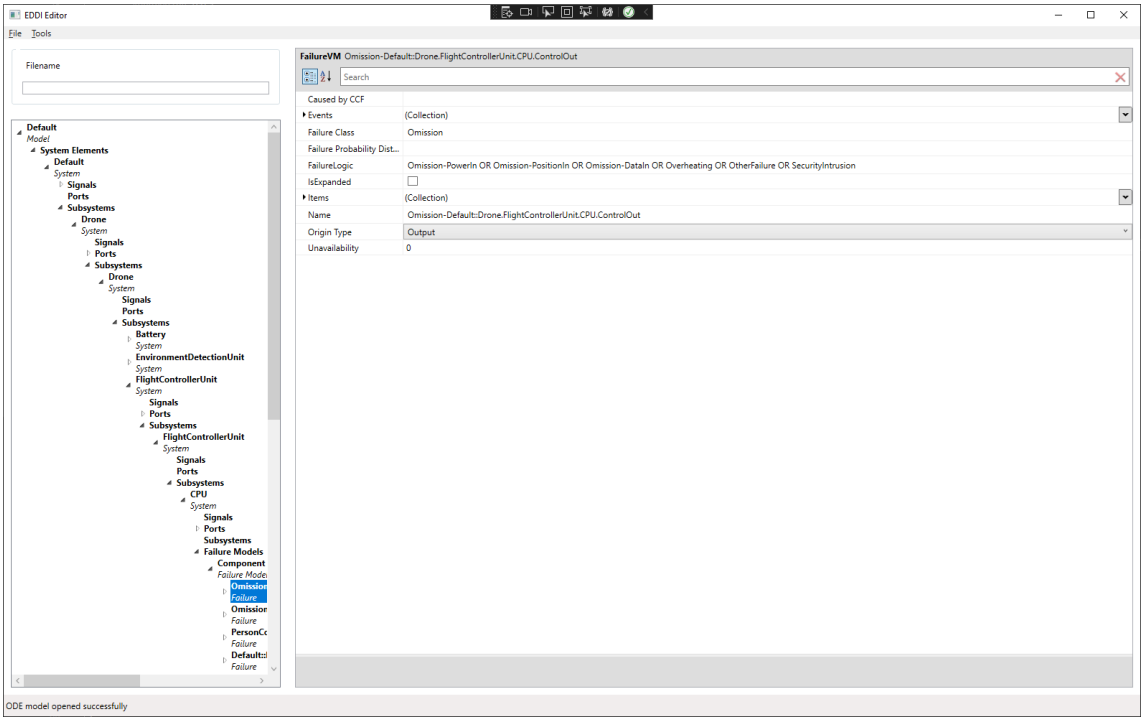


Figure 62: EDDI Editor

Functionality includes:

- Import of HiP-HOPS models (either separate system architecture or analysis models, or both combined) and conversion of them to ODE models;
- Import of Dymodia state machines and conversion of them to ODE models;
- Import of safeTbox EDDI models;
- Loading ODE models directly (e.g. system architectures, fault trees, FMEAs, state machines);
- Simple editing of entity properties in each model (both before and after conversion);
- Merging/combining of ODE models, e.g. by importing a model/subsystem hierarchy to replace an existing (empty) placeholder system, or adding new failure models etc;

Note that the EDDI Editor is not a modelling or analysis tool in itself: models must still be created and/or analysed in an appropriate tool first, like HiP-HOPS or safeTbox. Similarly, some degree of further post-processing (e.g. code generation) is necessary to produce an actual runtime EDDI.

Further information on how to use the EDDI Editor can be found in D4.6 [4] or the EDDI Editor user guide.

6.6 EDDI-BASED SECURITY ANALYSIS TOOLS

For a successful integration of the EDDI-based security analysis tools to the end user target system two separate tools must be installed and configured, OpenVAS and security EDDI tools (Snort, security EDDI). These tools need to be installed into the target system to offer the functionality of definition of the system vulnerabilities, detection of malicious packets transferred thought he network, and identification of the attacker's goal.

6.6.1 Installation

- **OpenVAS**: The installation process for OpenVAS is carried out in accordance with the instructions provided on the Greenbone website. The prerequisites for the installation include having Oracle VirtualBox 6.1 or a higher version, a minimum of 2 CPUs, 5GB of RAM, and direct access to the Internet.

  The installation involves using VirtualBox to create a new Virtual Machine (VM) based on an OVA file provided by Greenbone Enterprise. This VM comes preloaded with an installation of OpenVAS, which can be managed and used to conduct scans on target networks through a web interface. To configure the basic settings of the Greenbone OS, users can follow a Setup Wizard. This wizard facilitates the creation of a web user, enabling the use of a web interface. Following the completion of the wizard, a web browser can be employed to access and interact with this web interface, providing a user-friendly way to manage and operate OpenVAS.

- **security EDDI**: All the components that are part of the security EDDI are encapsulated in a containerized format. A Docker Compose YAML file, a tool for specifying and managing multi-container Docker applications, has been generated (Figure 63). In this setup, each component is presented as a service (ids_db, ids, mqtt-broker, snort-publisher, and security-eddi).

```yaml
version: '3.2'
services:
  ids_db:
    image: mariadb:10.3.10
    container_name: ids_db
    environment: ···
    volumes: ···
    ports: ···
    restart: always
  ids:
    #image: fabriziogaliano/docker-snort-ids:latest
    image: manmix/sesameids:1.11_desktop
    #image: manmix/sesameids:1.11_arm
    container_name: ids
    network_mode: host
    volumes: ···
    privileged: false
    environment: ···
    depends_on: ···
    restart: always
  mqtt-broker:
    image: eclipse-mosquitto:latest
    container_name: mqtt-broker
    ports: ···
    volumes:
    - type: bind ···
    restart: always
  snort-publisher:
    build: ···
    container_name: snort-publisher
    environment: ···
    depends_on: ···
    restart: always
  security-eddi:
    build: ···
    container_name: security-eddi
    network_mode: host
    environment: ···
    depends_on:
      - mqtt-broker
    restart: always
```

Figure 63: Docker-compose YAML file for the deployment of security part of EDDI

The YAML file outlines attributes for each service, including the image used to generate the corresponding container, the name of the container to be instantiated, necessary environmental variables, volumes to be mounted from the host, ports to be utilized, and more. Upon execution of the YAML file, a container is created for each component, and various volumes are mounted. This includes essential elements such as the Snort rules employed for each use case and the list of whitelisted IPs, ensuring the seamless deployment and operation of the entire system.

6.6.2 Configuration

- OpenVAS: The configuration process involves creating a scan target and defining a scan. To set up a scan target, users need to specify a list of hosts to be scanned,

along with a list of ports. For more advanced scans that require authentication, credentials can also be provided. When creating a scan, users are prompted to fill out a form that includes various attributes. These attributes encompass the minimum Quality of Detection (QoD) percentage and the Scan config. The QoD percentage reflects the reliability of the executed vulnerability detection. The Scan config attribute provides options such as:

- Base
- Discovery
- Full and fast
- Host Discovery
- Log4Shell
- System Discovery

Additionally, the duration of a scan is influenced by the chosen scan configuration. These configurations allow users to tailor the scanning process based on their specific needs and the desired depth of analysis.

- security EDDI: The configuration process involves the creation of a file that defines environmental variables and includes Docker Compose commands for building and starting the service images outlined in the YAML file. Six environmental variables are specified. The first three, namely IP (HOST_IP), network (HOST_NETWORK), and network interface (HOST_INTERFACE), are crucial for Snort, the intrusion detection system, to determine the network to monitor for malicious activity. ROS_MASTER and ROS_SECURITY variables are utilized by the ROS node generated when the security EDDI container is initiated. This node requires information on how the ROS master communicates its messages and the name of the topic to which the messages are published. Lastly, the EDDI variable defines the Python code for the security EDDI that will be executed. The docker-compose build command that follows searches for build instructions in the Docker Compose YAML file and executes the build process for each service with a build configuration. The build configuration typically includes details such as the build context, build arguments, and other relevant settings. Finally, a docker-compose up command is employed to start and initialize the services outlined in the Docker Compose YAML file. This command creates and launches containers based on the specified configurations, ensuring the seamless operation of the defined services.

6.7 SIMULATION-BASED TESTING OF EDDI TOOLS

6.7.1 Simulation-Based Testing Platform Tool

The SESAME Simulation-Based Testing infrastructure is implemented as a set of Java projects and tooling integrated with Eclipse[38], building upon open-source model-driven engineering tools such as the Eclipse Modelling Framework[39], Eclipse Epsilon[40] and Emfatic[41]. Additional standard Java technologies such as the Maven build tool[42] are used to recompile code components dynamically generated during the execution of the experiments. Apache Kafka[43] and Flink[44] are used for message communications and stream processing, in order to interconnect the MRS simulator, the individual test runners, and the fuzzing engine that manages the experiments.

Dependencies for Linux - Ubuntu 18.04 and 20.04 tested

- Apache Kafka / Zookeeper
- ROS installation (either ROS Melodic or Noetic) - if using ROS use cases - with rosbridge
- Eclipse and EMF modelling tools

When using the platform on Linux, it is possible to run all components natively, with no additional operating system or container support required.

Additional dependencies for Windows - Windows 10 tested

- Eclipse and EMF modelling tools
- Docker Desktop - since Kafka uses Docker on Windows
- Apache Kafka / Zookeeper, OpenJDK8 and JDK11
- Administrator access
- Virtualisation enabled in the BIOS (may be defined as ``Hyper-V")

When using the platform on Windows, Linux containers are required via Docker (using the Windows Subsystem for Linux internally) to provide Kafka/Zookeeper, which the simulation-testing platform uses for its message communication. Since the simulation-testing platform requires these and a native Windows installation of Kafka/Zookeeper does not have the necessary features, this additional operating system support (via containers) is required. Using the platform on Windows is necessary when using external applications such as Simit, for the KUKA use case.

6.7.1.1 Installation Instructions

The installation process for the platform in Linux is documented here.[45]. Separate installation instructions for Windows are available[46]. A user guide is also provided.[47]

[38] https://www.eclipse.org/
[39] https://www.eclipse.org/modeling/emf
[40] https://www.eclipse.org/epsilon
[41] https://www.eclipse.org/emfatic
[42] https://maven.apache.org
[43] https://kafka.apache.org
[44] https://flink.apache.org
[45] https://github.com/sesame-project/simulationBasedTesting/blob/main/documentation/index.md

6.7.2 Methodology Execution Example

In this section, we present an example of the simulation-based testing methodology of Figure 17, for testing a simplified model for the KUKA/TTS use case. This example incorporates a combination of three robots with sliding conveyors that interact to transport the gearboxes between assembly robots, together with human workers surrounding the assembly cell who assist with specific tasks requiring manual intervention. The arrangement of the cell is depicted in Figure 52. The primary safety requirement is to ensure that robots do not collide with regions that could cause a hazard to the human operator or other workers (the green cuboids outside of the cell frame in Figure 52). Under normal circumstances, motion trajectories are pre-planned to ensure conformity to this safety requirement. However, runtime faults in communication, sensing or motor/interlock activation may lead to robots malfunctioning and entering these forbidden areas.

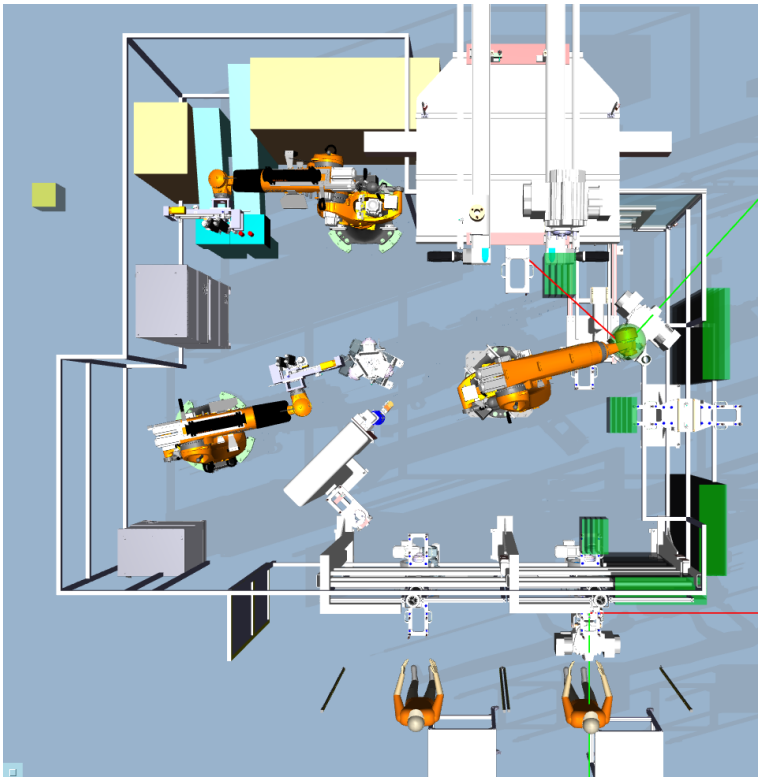


Figure 52: an example use case for testing the TTS cell

In order to use the simulation-based testing platform on this use case, firstly, the user should load Eclipse and then invoke a new Eclipse Application, by right-clicking upon the newly imported project *uk.ac.york.sesame.testing.generator* and selecting ``Run As'' / ``Eclipse Application''. This will launch a fresh Eclipse instance under which the

[46] https://github.com/sesame-project/simulationBasedTesting/blob/windows/README-windows.md
[47] https://github.com/sesame-project/simulationBasedTesting/blob/main/documentation/userguide.md

SESAME automated code generation plugins are available). Every project tested during simulation-based testing should be instantiated in a new Java project - here, we TTSTestProject. Create a folder ``models'' in it to store the models for the DSL. [48]

Step 3.1: Having analysed the case study requirements, participating robots and the necessary fuzzing operations, users will have specified the performance metrics and the fuzzing operations for testing. Users should codify this in an instantiation of the SESAME Testing DSL, specifying the structure of the testing space of possible operations and performance metrics. The DSL metamodel is specified in our deliverable D6.6 [14]. The Exceed editor provides a convenient visual editor to aid configuration of the testing process.

An example for testing the KUKA/TTS example case study is presented in Figure 53 for the case study. This model includes two performance metrics for quantifying the length of fuzzing used, and collisions with static safety zones. It also includes condition metrics for defining the triggers for condition-based fuzzing and the selection of potential fuzzing operations upon different robots. In this case, the fuzzing operations introduce distortion upon particular joints of robots R3200 and R270073. Further details on this case study testing are presented in the case study in D6.6 [14].

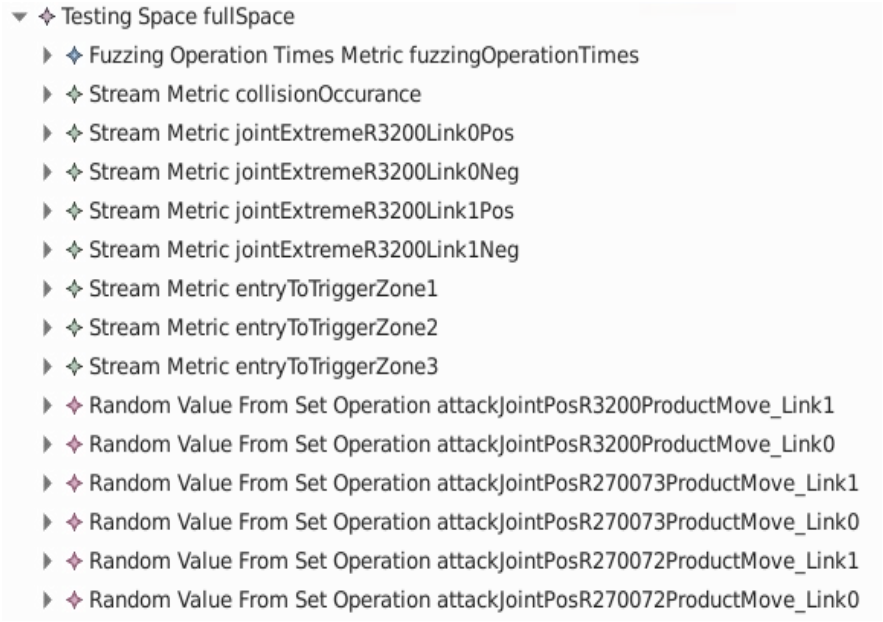


Figure 53: example of the testing DSL model for the KUKA/TTS use case

Step 3.2: Code generation can be used to generate metric templates automatically, within the newly generated project under the child Eclipse instance. The testing platform provides a plugin consisting of a wizard with a single page, which can be accessed by right-clicking on the user's newly generated project and selecting "Generate SESAME Code" (Figure 54). The plugin provides an interface option to select the file containing the user's populated model and associated settings (Figure 55). The annotations in red upon the screenshot show the values selected for the text boxes. Here we choose the model file and the locations of other items for the project.

[48] In order to generate a model for the first time in a newly created project, it is necessary to register the metamodels. This can be done by activating the early stage of our wizard, by right-clicking on ``SESAME'' / ``Generate SESAME Code'', as shown in Figure 53. Then, click Cancel on the dialog box that appears.

When this button is clicked, metric templates will be generated in the package *metrics.generated*. Experiment runners will also be generated based upon the experiment name defined for the TestCampaigns in the model, e.g., **ExptRunner_(name).java**.

Figure 54: Plugin menu

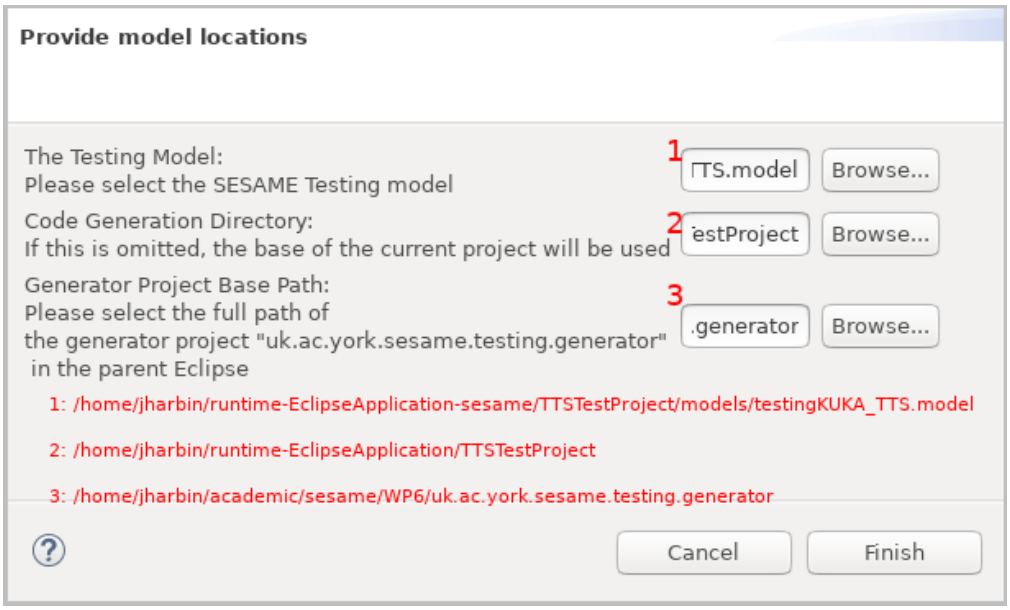


Figure 55: Code generation wizard dialog

This step can also be performed using the integrated platform. The model file can be uploaded to the online platform, and using its code generation features, the experiment runners and metric templates will be automatically generated and available for download to implement (see section 4 for more details).

Step 3.3: The next step involves the user specifying scenario-specific performance metrics for the metrics generated in the model, in order to quantify violations of mission requirements. In order to implement these metrics, the user first needs to copy these classes from package *metrics.generated* into a new package *metrics.custom*. Then, it is necessary to implement the needed platform-specific metrics as Java code.

Figure *56* presents a fragment of the implementation of the *collisionOccurrence* metric, used to quantify violations by tracking the number of intervals of collisions of the gripper safety zone with any exterior safety zone. The metric operates as follows:

If the region surrounding the robot gripper (green sphere) collides with these regions, the collision detection logic in DDD Simulator will trigger a safety zone message, which will be sent to the testing platform via the TTSSimulator custom API over gRPC. As an inbound simulator event, these will then trigger execution of the processElement1 method of the metric. If the message inbound topic is a safety zone message of sufficient depth and the timing is ready to trigger its logging, then the *violationCount* will be incremented. This value is emitted as the output value. The final *violationCount* in the course of the experiment will be logged as to the model file the output of this metric for that test configuration.

```
if (topic.contains(completionTopicName) && topicMatches(topic)) {
    if (msg.getValue() instanceof String) {
        String s = (String) msg.getValue();
        Optional<ROSMessage> rosmsg_o = ROSMessageConversion.fromJsonString(s);
        if (rosmsg_o.isPresent()) {
            ROSMessage rosmsg = rosmsg_o.get();
            SafetyZone sv = rosmsg.getSafetyZone();
            float level = sv.getLevel();

            if (violationCount.value() == null) {
                violationCount.update(0L);
            }

            if (level < getLevelThreshold() && isReadyToLogNow()) {
                violationCount.update(violationCount.value() + 1);
                out.collect(Double.valueOf(violationCount.value()));
            }
        }
```

Figure 56: Test Collision Metric code for detecting collisions during simulation

Step 3.4: The user should create an Eclipse Run Configuration for the **ExptRunner_(name).java** for the experiment they would like to execute, and invoke this Run Configuration in order to run the experiment. This runner will be configured with the parameters chosen from the Testing DSL. If new experiments are specified in the testing DSL, or if any details of the experiment configuration have been changed, it will be necessary to again select the "Generate SESAME Code" plugin, and repeat this process of code generation to regenerate the experiment runners.

The experiment runner will generate tests according to the strategy specified for the experiment, for example, with NSGA. Its TestGenerationApproach selection allows the user to specify the parameters for an experiment by setting one of several subclasses. For example, including NSGAEvolutionaryAlgorithm allows an evolutionary experiment with the NSGA-II algorithm, and contains specific parameters relevant to this approach, e.g., the number of iterations and the population size. We also provide a new coverage-aware GA NSGACoverageWithCells, which seeks to improve coverage of the space of potential fuzzing tests. Further, RepeatedRunner provides support for repeated execution of a particular selected test a number of times. The utility of this is to allow an interesting test with a high reality gap or other performance issues to be repeated and the reasons for its behaviour investigated in depth.

Regardless of the test generation strategy selected, the performedTests attribute is populated during the execution of experiments, containing the particular Tests generated and

executed for that campaign. Each test is evaluated to quantify the impact of the fuzzing test in terms of the performance metrics defined in Step 3.3. The *resultSets* attribute is also populated as the experiments proceed and finalised upon their completion, containing references to the population of results upon a Pareto front. This enables keeping track of the history of evolved tests during simulation-based testing.

6.7.3 Implementing a SimlogAPI interface

Depending on the underlying simulation engine, the implementation of the SimlogAPI interface could require different levels of development efforts. Nevertheless, the following guidelines provide a quick introduction to:

- the generation of the server and client stubs from the SimlogAPI.proto definition
- the main concepts related to the implementation of a testing client that would like to exploit the interface once implemented at simulation side

6.7.3.1 Generating the stubs

A convenience maven project has been provided for the quick setup of Java stubs. Artifacts for other programming languages can be easily generated modifying the pom.xml configuration file.

1. Download the maven project ZIP[49]
2. Unpack the file
3. Use the favourite target IDE (IntelliJ IDEA, Netbeans, Eclipse) to import the project
4. Build the project, the default compiling process will generate a shaded jar in the target directory.
5. If stubs should be generated for a different target language than Java, refer to the protobuf-maven-plugin documentation[50] to correctly configure it in the pom.xml

6.7.3.2 Implementing a Java client

The concepts related to the development of a sample client for Java apply to other languages. In order to implement a client, it is important to complete the previous step, generating the stubs. The following snippets of code provide examples on the basic operations for subscribing to a simulation topic and receive notifications.

Opening a communication channel with simulation engine and subscribing:

```
String target = "localhost:8089";
String topicID = "R3200.Link1.R";

// 1-Open the channel on the desired URL/port
```

[49] https://github.com/sesame-project/SimulationTestingFramework/blob/main/SimlogAPI.zip
[50] https://www.xolstice.org/protobuf-maven-plugin/index.html

```
ManagedChannel channel = ManagedChannelBuild-
er.forTarget(target).usePlaintext().build();
try {

  // 2-Instantiate the client stub
  blockingStub = SimlogAPIGrpc.newBlockingStub(channel);
  asyncStub = SimlogAPIGrpc.newStub(channel);

  TopicDescriptor request =
  TopicDescriptor.newBuilder().setPath(path).build();

  // 3-Call the desired service function
  ROSObserver ro = new ROSObserver(topicID);
  asyncStub.subscribe(request, ro);

} catch (StatusRuntimeException e) {
   warning("RPC failed: {0}", e.getStatus());
} finally {
   channel.shutdownNow().awaitTermination(5, TimeUnit.SECONDS);
}
```

Implementing a stream observer to receive messages:

```
private static class ROSObserver implements StreamObserver<ROSMessage>
{

        private String path;

        public ROSObserver(String p) {
            this.path = p;
        }

        @Override
        public void onNext(ROSMessage m) {
            // Data notification method
            System.out.println(path + ":message received =" +
              m.getValue());
        }

        @Override
        public void onError(Throwable t) {
            System.err.println(path + ":failed: " +
              Status.fromThrowable(t));
        }

        @Override
        public void onCompleted() {
            System.out.println(path + ":finished");
        }
}
```

6.8 TESTING OF ML COMPONENTS TOOLS

6.8.1 DeepKnowledge configuration and usage

We developed **DeepKnowledge** as a Python package that facilitates effortless installation and is a pivotal endeavor to deploy our tool within different industrial partners' de-

velopment environments. This allow DeepKnowledge to be reproducible at very low effective cost.

DeepKnowledge has been updated since the last version described in deliverable D6.1 [16]. The new iteration has been deployed with an expanded scope, encompassing not solely image recognition and classification (DKOX use case) but also extending its functionality to object detection, i.e., YOLOv8 (KIOS and PAL use cases).

The package offer several advantages, including: **simplified deployment,** such that our SESAME partners can effortlessly install and access our package without needing to navigate complex dependencies; **reusability**; i**ntegration with dependency management tools** like pip, conda, and virtual environments, enhancing the overall development and deployment experience.

To allow for reproducibility of our implementation, package, documentation and metadata are available on Github at https://github.com/sesame-project/ML_Testing

6.8.1.1 Installation and configuration

The development of a DeepKnowledge Python package for testing deep learning models necessitates the incorporation of numerous libraries for several compelling reasons:

- Covering multiple Computer vision tasks including image classification and object detection required different libraries for diverse deep learning models support, such as ultralytics for YoloV8.
- Specialized Functionality: different Libraries are designed to provide specialized functionality and tools tailored for specific tasks such as mathematical computations or pre-processing and augmentation functions, loss functions, and evaluation metrics. We also needed different libraries for different coverage metrics.

After cloning the github repository, users should run the commands below to install the libraries required by DeepKnowledge.

```
$ cd DIRECTORY/DeepKnw_pack/

$ pip install -r DeepKnowledgeRequirements.txt

$ python setup.py install
```

6.8.1.2 Running DeepKnowledge

Users can run DeepKnowledge using the example command below.

```
$ python
```

```
####Example usage for DesignTime Deployment:

>> import DeepKnw_run as knw

>> COV=knw.DeepKnw(PATH_To_CONFIG_FILE)

>> test path=PATH To Test Data Images
size=5000 (← EXPL)

>> test_loader = COV.getTestloader(test_path,size)

>> Deep-
Knowledge_Coverage=COV.estimate_coverage(test_loader)


####Example usage for RunTime Deployment:


>> import DeepKnw_run as knw

>> COV=knw.DeepKnw(PATH_To_CONFIG_FILE)

>>YOLOloader, model_features=COV.DesignDataAnalyzer()

>> Frames_path=PATH_To_RunTime_Data

>> Batch=5 (← EXPL)

>> Frame_Loader =
COV.getTestloader(Frames_path,Batch)

>> Deep-
knowledge_Uncertainty=COV.Runtime_Estimate(Frame_Load
er,YOLOloader)
```

6.8.2 GenRepair Tool Configuration and Usage

The GenRepair tool is developed as a set of Python scripts enabling the use and integration of the hardening and repairing tools easily and in a cost effective way.

The provided Python scripts can be run into Jupyter Notebook, or in an IDE. In addition, GenRepair scripts can be run in the command line using the command prompt or PowerShell in Windows. In macOS or Linux, the test engineers can execute the tool also using the terminal or xTerm.

Code, documentation and metadat are available on the Github repository \https://github.com/sesame-project/MLTesting.

6.8.2.1 Installation and configuration

Similar to Deepknowledge, GenRepair requires a set of libraries that covers different functionality including the StableDiffusion Library for semantically augmenting the dataset.

These libraries can be installed by running the following commands:

```
$ pip install -r GenRepair_requirements.txt
####We need to install the generative AI models Stable Diffusion and Transformer from the huggingface platform as fellow:
$ pip install https://github.com/huggingface/diffusers/archive/main.zip -qUU --ignore-installed

$ pip install transformers -q -UU ftfy gradio
```

6.8.2.2 Running GenRepair

The users needs to run the tool following the precise steps using the following command lines.

First a Coverage-Guided Fuzz Testing can be performed separately to reinforce the safety assurance of the DNN component by running:

```
$ cd path/to/the/project/folder

$ Python3.8 fuzzing.py -method[0 or 1] -fuzzer[type_of_fuzzing]  -repair[continuous_learning_paradigm] -it[nbre_of_iteration] -model [path_to_keras_model_file] -dataset [dataset_name] -approach [coverage_criteria] -logfile [path_to_log_file]
```

Then, based on the results and on the testing engineers judgement a continuous model repairing can be performed by running *repairing.py* script using the shell commands:

```
$ cd path/to/the/project/folder

$ Python3.8 repairing.py -method[0 or 1] -
fuzzer[type_of_fuzzing]  -
repair[continuous_learning_paradigm] -
it[nbre_of_iteration] -model
[path_to_keras_model_file] -dataset [dataset_name] -
approach [coverage_criteria] -logfile
[path_to_log_file]
```

6.8.2.3 GenRepair Parameters

```
-method : an integer took 0 for fuzzing operation and 1
for repairing.

-fuzzer : the name of the selected fuzzer. Our platform
provides 5 strategies, with the possibility of choosing a
combination of different testing criteria as guidance.
These fuzzers are:

  Random Noise testing (RN). This combines random sam-
pling as seed selection strategy and Gaussian Noise for
data augmentation
  Random Inpainting (RInp). This fuzzer uses random sam-
pling strategy and test-guided Stable Diffusion Inpaint-
ing as data augmentation.
  Random Semantic Occlusion (SemOcc). This fuzzer uses
```

```
random sampling strategy with  Semantic occlusion (both
random erasing and   synthetic occlusion similarly) to
augment each input seed.
  DeepKnowledge Inpainting (KnwInp). Different from RInp,
this strategy guides testing using DeepKnowledge coverage
criteria as feedback. An input seed is put to the seed
queue if it improves the Deepknowledge coverage.

-repair : this the selected paradigm for continuous
learning. We can select :
CLTask : Task incremental learning
CLClass : Class incremental learning
CL : Continuous learning of known classes

-it : number of iterations for data augmentation within
the fuzzing process. We advise to select an integer be-
tween 2 an 10.

-app: for approach. The approach for coverage estimation.
The selected coverage is used as guidance in each itera-
tion to pick the augmented seed as new test. Our current
implementation supports DeepKnowledge (Knw), DeepIm-
portance (idc),(nc),(kmnc),(nbc),(snac),(tknc),(ssc),
(lsa), and (dsa).

-model : the name of the Keras model file. The trained
keras model is saved as .hdf5 file or the architecture
can be saved as JSON and the weights saved separately as
an .h5 file. All the trained model are saved under the
folder `Networks'. Our implementation provides three
trained DNN models including Allconvnet.h5, LeNet5.h5,
and Vgg19.h5.

-dataset : name of the dataset to be used. Current imple-
```

```
mentation supports Cifar-10 (cifar) COCO (coco) and grape
leaves (grape). Our platform is extensible and other da-
taset can be added by modifying `Dataprocessing.py' and
`Data_Augment.py' scripts.

-layer : The subject layer's index for approaches includ-
ing 'idc','tknc', 'lsa'. Note that only trainable layers
can be selected.

- logfile : The name of the file that the results are to
be saved.
```

6.9 RUNTIME EDDI GENERATION TOOLS

6.9.1 Runtime EDDI Generator Tools

The ROS configuration file is described in YAML format. It consists of four main sections:

Model: In the **Model** section of the YAML file, basic information is provided. The standard fields are **id, type, frequency** and **parameters.** These correspond to the model name, the model type (ConSert, Bayesian Network, SafeML) and the frequency of the ROS main loop. The **parameters** field describes model specific information, as explained in the corresponding sections below.

SimulatorOutputs: In this section, values from an existing ROS network are described. These values might be used to create input values for the EDDI monitor. Each of these values have a name (**id**), a data type (**type**) and the topic (**topic**) within the ROS network. The ROS node will subscribe to the topic and propagate the received message for further processing

EDDIInputs: Here, a list of all input parameters of the EDDI monitor are described. Each of them has a name (**id**), a data type (**type**), a default value (**default**) and a field **requires.** The **requires** field describes a list of **id**s which are matched to the **id**s of the **SimulatorOutputs** values. An EDDI input parameter might depend on one or more existing messages.

EDDIOutputs: In this section, the output of the EDDI monitor is specified. The ROS node will publish the message under the specified **topic.** Additionally, a name (**id**) field is set.

The main purpose of the configuration file is to link ROS messages within an existing ROS network to input parameters of the respective model. Therefore, the **SimulatorOutputs** are defined and linked within the **EDDIInput** section to the related

input parameters. Also, it is specified how the results of the EDDI monitor are published within the ROS network. Further, the configuration file is used to specify the general properties of the ROS node and package such as the name, frequency and type.

This file is required as an input to the ROS wrapper generator explained in the upcoming sections.

A generic Python script consumes the ROS configuration file mentioned above and allows each of the EDDI runtime components described below to be generated.

6.9.2 ConSerts

The ROS configuration file specifies the ConSert guarantees within the parameter section. The name of demands, runtime evidence and guarantees must match between the config file and the EDDI ConSert description.

In order to generate a ConSert monitor ROS node, the first step derives the ConSert described in the XML EDDI model and creates a ConSert description as a YAML file.

```
java -jar egl.jar -e <egl_script> -m <ecore model_file> -x <xml_file>
```

The next step generates the ConSert monitor in Rust. Therefore, the consert-rs tool is used. It transforms the ConSert defined in a YAML into rust code. The "--py" argument results in annotated rust code which is required in the next step.

```
conserts.exe compile --py -I <consert_name.yml>
```

Afterwards, the wheel files are created using maturin[51]. Therefore, ensure maturin is installed. Otherwise,

```
pip install maturin
```

In the same directory as the generated rust file are located, the wheel files are generated with:

```
maturin build
```

The resulting .whls files are then installed with

```
pip install consert_<consert_name>.whl
```

Now the corresponding dependencies can be imported in any python script. The next step requires an already existing catkin_ws directory. If no such directory exists, create at least

```
catkin_ws/src
```

The workspace can then also be initiated with

```
catkin init
```

[51] https://pypi.org/project/maturin/

Now with

```
python generator.py -w <catkin_ws> -c <config.yml> -m <consert.yml>
```

where the catkin_ws path, the ROS configuration file path and the path of the ConSert specification YAML are given as arguments.

This will create the ROS packages: eddi_monitor, eddi_messages and eddi_monitor_launcher if those do not already exist. Additionally, a ROS package is created for the specified ConSert which contains the script to run the ConSert as a ROS node. Before the ROS node can be executed, some small manual coding is required. Within the EDDIInput.py script, for each input specified in the configuration file, a processing method was created which must be implemented. The goal here is to process the required ROS messages to a proper EDDI input value. In Figure 64 the function "process_RtE_A_E0(..)" must be implemented by assigning the field "RtE_A_E0" depending on the input parameter "RtE_E0".

```
1   class EDDIInput:
2       def __init__(self):
3           self.RtE_A_E0 = False
4       def process_RtE_A_E0 (self, RtE_E0):
5           # TODO
6           self.RtE_A_E0 = RtE_E0.data
7
8       def processing (self, simulator_outputs):
9           if 'RtE_E0' in simulator_outputs:
10              self.process_RtE_A_E0(simulator_outputs['RtE_E0'])
11
```

Figure 64 Generated example EDDI Input script

In order to run the ROS node now use

```
chmod +x src/eddi_monitor/<consert_name>/scripts/RosNode.py
dos2unix src/eddi_monitor/<consert_name>/scripts/RosNode.py % (if
generated on Windows and executed now on Linux)
source catkin_ws/devel/setup.bash
catkin_make
roscore & % (if not already started)
roslaunch eddi_monitor_launcher eddi.launch &
```

All EDDI ROS nodes created accordingly with the generator script are then started.

6.9.3 Bayesian Networks

For generating a Bayesian network monitor ROS node, several parts are required:

1. A ROS configuration is needed that is tailored to the concrete Bayesian network model and ROS runtime environment.

2. An EDDI representing the Bayesian network is needed.

3. The generator and transformation scripts must be available.

The ROS configuration file (YAML format) for the Bayesian network monitor comprises just the generic parts, like the concrete inputs, as explained in the previous sections.

Second, an EDDI model (XML-based format) representing the Bayesian network is a prerequisite. This model is then translated into a specific runtime representation (.xdsl / .xmlbif format) of the Bayesian network using the Epsilon Generation Language (EGL) and Python.

1. Translate the EDDI model to a .xdsl representation of the Bayesian network using (EGL):

```
java -jar egl.jar -e <egl_script> -m <ecore model_file> -x <xml_file>
```

   a. With the "egl_script" being the concrete "ddi_bn_to_xdsl.egl" script.

   b. With "ecore_model_file" being the concrete metamodel representation of the Bayesian network: "bayesianNetwork.ecore"

   c. With "xml_file" being the concrete input DDI model file.

2. Then, the .xdsl Bayesian network representation is automatically translated into the .xmlbif Bayesian network format for runtime inference. This is done in Python and happens automatically when generating the Python Bayesian network monitor, respectively, the ROS Bayesian network monitor, using the provided Python scripts as explained below.

Now, when having the runtime representation of the Bayesian network as a .xdsl file, there are two options. Either a Python Bayesian network monitor or a ROS Bayesian network monitor can be generated. For both a Python script for the code generation is provided.

1. The Python Bayesian network monitor can be generated with the "bn_monitor_generator.py" script by providing the previously generated .xdsl runtime representation of the Bayesian network as an input:

```
python3 -m BayesianNetwork.bn_monitor_generator -bn <bayesianNetwork>
```

   a. With "bayesianNetwork" being the name of your .xdsl file which must be located in the same directory as the "bn_monitor_generator.py" script.

   b. The generated Python package will be located in the "out/bn_monitor/" directory that will be generated automatically.

2. The ROS Bayesian network monitor can be generated with the "generator.py" script by providing the aforementioned ROS configuration YAML file:

```
python3 generator.py -w <catkin_ws> -c <config.yml> -m
<bayesianNetwork.xdsl>
```

a. With “catkin_ws” being the path to your catkin workspace for whose src directory the ROS Bayesian network monitor shall be generated and stored at.

b. With “config.yml” being the aforementioned ROS configuration file.

c. With “bayesianNetwork.xdsl” being the runtime Bayesian network model file as generated before with the EGL script based on the EDDI model.

In both cases, you must manually adapt the Bayesian network configuration to your needs (output nodes and concrete discretization for the Bayesian network). This configuration Python script “bayesian_network_config.py” is located in the generated code under “files/bayesianNetworkName/”, respectively, under “catkin_ws/src/bayesianNetworkName/scripts/files/bayesianNetworkName/”.

Also, in case of the ROS Bayesian network monitor you must adapt / implement the concrete data translations, i.e., the mapping between the ROS topic data inputs and the concrete inputs for the Bayesian network monitor. This shall be done in the generated “EDDIInput.py” Python script that is located in “catkin_ws/src/bayesianNetworkName/scripts/”.

Finally, you can run the generated Bayesian network monitor:

1. The Python Bayesian network monitor runs by the generated “bn_monitor.py” script that comprises the interface for the monitor.

2. The ROS Bayesian network monitor can run in the ROS environment that is described in the ROS YAML configuration. For details on the required steps, the ROS package structure, and so on, we refer to the ConSerts section above (see Section 6.9.2 ConSerts) in which the ROS steps are explained in depth (the steps for the Bayesian network monitor are similar to the ones from the ConSert monitor).

Important: For running the Bayesian network monitor, you must install the pgmpy library[52] using Python’s pip in your runtime Python environment:

```
python3 -m pip install pgmpy
```

6.9.4 SafeML as ROS Monitoring Component

The SafeML monitor requires:

1. Input model, used for detection

2. Feature extraction method

3. Model weights used

[52] https://pgmpy.org/

4. Design-time SafeML measures

Hence, this requires a tailored solution for specific use case. The SafeML monitor tool, and the tailored files for the use-cases it was involved in can be found on the SESAME GitHub (runtime_eddis/SafeML at master · sesame-project/runtime_eddis (github.com)). For generating SafeML ROS monitoring component, preferred method is by using dockers. For SafeML, the base image used is 'opendr/opendr-toolkit:cpu_nightly_300823'. The base image used is Open Deep Robotics (OpenDR) image, that uses YOLO for object detection, classification and tracking for robotics. It also has ROS integration. For this project, we only considered detection.

The SafeML monitor assets can be compiled first, by using the various data required. The model agnostic nature of SafeML allows use of any input model. However, as it increases the complexity, it works best with the model that allow the intermediate feature extraction, enabling dimensionality reduction. The feature extraction techniques can be model specific. And should be consistent during design time and runtime usage of SafeML. During design time, these techniques allow users to extract the important relevant information (such as accuracy, statistical distance, features, etc). These extracted features can then be used at runtime to obtain a SafeML monitor node. A sample Dockerfile to generate SafeML monitor node is as follows.

```
FROM opendr/opendr-toolkit:cpu_nightly_300823
RUN rm /bin/sh && ln -s /bin/bash /bin/sh

WORKDIR /opendr/
#RUN /bin/bash -c 'source /opendr/bin/activate.sh'
RUN source bin/activate.sh

WORKDIR /opendr/projects/
RUN mkdir WS
COPY safeml_scue/ /opendr/projects/WS/safeml_scue/

RUN apt-get update
RUN python3 -m pip install --upgrade pip

RUN cp /opendr/projects/WS/safeml_scue/YOLOv8_node.py
/opendr/projects/opendr_ws/src/opendr_perception/scripts/
RUN cp -r /opendr/projects/WS/safeml_scue/yolov8_mod/
/opendr/projects/opendr_ws/src/opendr_perception/scripts/
RUN cp -r /opendr/projects/WS/safeml_scue/scue/
/opendr/projects/opendr_ws/src/opendr_perception/scripts/
RUN cp /opendr/projects/WS/safeml_scue/safeml.py
/opendr/projects/opendr_ws/src/opendr_perception/scripts/

RUN /opendr/venv/bin/pip install -r
/opendr/projects/WS/safeml_scue/requirements.txt
RUN source /opt/ros/noetic/setup.bash

WORKDIR /opendr/projects/opendr_ws
RUN /bin/bash -c '. /opt/ros/noetic/setup.bash; cd
/opendr/projects/opendr_ws; catkin_make'
RUN echo "source /opendr/projects/opendr_ws/devel/setup.bash" >>
~/.bashrc

RUN chmod +x src/opendr_perception/scripts/YOLOv8_node.py
```

With the help of above dockerfile, the SafeML monitor docker can be run using

```
docker build -t safeml:0.1.1 .
```

The monitor can then be ran using

```
docker run --gpus all --net=host --rm -it safeml:0.1.1 bash
```

This will start a docker, which will wait and listen for the ROS master to output the image topics. Activate the appropriate environment

The SafeML monitor can then be started, by giving the topic name it should listen to for the image, and the topic name it should send the output of SafeML to. A sample run is as follows:

```
rosrun opendr_perception YOLOv8_node.py -i /topic/name/for/input/image
-m /path/to/model/weights
```

6.10 MULTI-AGENT SYSTEM FOR SECURITY AND SAFETY MANAGEMENT

For ROS systems, EDDIs can be deployed using our generator.py script. Therefore, a catkin workspace must exist. Alternatively, a *catkin_ws/src* folder must be created. For each EDDI model file e.g., a ConSert model, a configuration file must be created that links the relevant messages of the ROS network to the inputs of the EDDI and on the other hand defines the topics and format of the EDDI outputs. These configuration files are mandatory to deploy the EDDIs. Then, the ROS nodes can be generated by executing:

```
generator.py -w <catkin_workspace> -m <model_file> -c <config_file>
```

The catkin workspace now contains the ROS nodes for each EDDI. Next, each RosNode.py within the *catkin_ws/src/eddi_monitor/<node>/scripts/* must be made executable:

```
chmod +x RosNode.py
```

In order to run the ROS nodes, the EDDIs (ConSerts) must be build first. The ConSerts are built with *conserts-rs*. Choose the binary that supports the target system and call:

```
Conserts compile -i <model_file> --py
```

Within the target folder, navigate within the generated ConSert package and call:

```
maturin build
```

A new target folder is created that contains a wheel file within the wheels directory. The wheel must be installed with:

```
pip3 install <wheel>
```

After the new ROS files are added to the directory, the workspace must be rebuild.

In non-ROS systems, EDDIs can be deployed using Docker[53] and Docker Compose[54] technologies. This approach is used in the KUKA use case (see D8.14), where ROS is not available. The approach re-uses the generated EDDI code (specifically ConSerts). The code is used by a Python runner component which supports a multitude of web-based APIs (MQTT[55], RESTful HTML, ActiveMQ [56]and gRPC[57]). The user can deploy the runner component in standalone or server mode. Standalone mode runs a single EDDI (ConSert), whereas server mode starts a server which awaits RESTful HTTP requests. Upon receiving a request, the server initiates a ConSert (from the ones installed and available) using the specified configuration and API.

The requirements for building EDDI (ConSerts) code apply here as well. Python 3.8 is supported, but newer versions may also be compatible. Additional dependencies for the runner component can be installed using the following (from the runner component directory).

```
pip3 install –r ./requirements.txt
```

The runner is setup by copying the generated EDDI (ConSert) Python wheel files under the 'consert_wheels' directory in the runner component's directory. The user should then install the wheel files into the Python interpreter environment using

```
pip3 install <path to ConSert .whl file>
```

In standalone mode, the runner component is deployed using

```
python3 ./main.py [rest|mqtt|activemq|grpc]
```

In server mode, the runner component is deployed using

```
python3 ./main.py --offline=False
```

The –h flag can be used to view additional configuration options for both modes and for each API mode.

To build a reusable Docker image, the following command can be invoked from the runner component root directory

```
docker build --build-arg CONSERT_WHEEL_RPATH=<filename of ConSert .whl
file> -t <name of docker image> .
```

Docker Compose can be used to automatically raise and manage several runner services. This approach requires the definition of a .yml file for Docker Compose (see example in runner component root directory). Once that has been created, the user can use the following command to start the EDDI (ConSert) MAS

```
docker compose up –d
```

[53] https://docs.docker.com/get-docker/
[54] https://docs.docker.com/compose/
[55] https://mqtt.org/
[56] https://activemq.apache.org/
[57] https://grpc.io/

7. CONCLUSIONS

Following the plan for delivering the SESAME technologies, and the selected Adaptive Project Management approach, the Incremental Integration Strategy (IIS), and the planned features of the initial (M18) and final (M30) versions of the SESAME platform, this deliverable includes the integrated versions of the SESAME solutions, reports on features and installation/customization guidelines. The report itself documents the integration for the final versions of the SESAME tools and modules. A detailed description of all components in the final version of the SESAME solutions is provided, as well as the basic workflow of the SESAME tools and components (which is customisable for the different use cases), and the description of the SESAME integrated platform. The continuous integration and deployment process, which was followed throughout the project, is highlighted as well.

The report provides a guide for the installation and configuration of the SESAME components. The objective was to provide modelling and tooling that are robot operating system agnostic and therefore able to support multiple industrial robotics platform. Aiming at high flexibility and easy adaptation to diverse robotic applications and starting from the five different use cases in the project, the solutions were designed to be loosely coupled.

Furthermore, the requirements upon the SESAME components were extracted from deliverable D1.1 SESAME Project Requirements [23] and analysed in detail.

The integrated platform and SESAME components have throughout the project addressed the issues that have risen during development and accommodated any new technical requirements or changes in the selected technologies. The feedback from the use case partners, obtained within the testing of the SESAME solution, as well as from the reviewers, was strongly taken into account in the updates of the integrated platform and the corresponding documentations/guidelines.

8. REFERENCES

[1] SESAME consortium, "D8.3 Integrated Platform - Initial Version," 2022.

[2] SESAME consortium, "D8.1 Architectural Guidelines," 2021.

[3] SESAME consortium, "D4.2/D5.2 Safety/Security-Targeted ODE and EDDI specification," 2022.

[4] SESAME consortium, "D4.6 Tools for Automated Safety Analysis of EDDIs (final version)," 2023.

[5] B. Kaiser, P. Liggesmeyer and O. Mäckel, "A new component concept for fault trees," *Proceedings of the 8th Australian workshop on Safety Critical Systems and Software,* vol. 33, pp. 37-46, 2003.

[6] T. Kelly, "A Systematic Approach to Safety Case Management," *SAE Transactions,* vol. 113, pp. 257-266, 2004.

[7] D. Schneider and M. Trapp, "Conditional safety certification of open adaptive systems.," *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* vol. 8, no. 2, pp. 1-20, 2013.

[8] Y. Papadopoulos, I. Sorokos, J. Reich and R. Wei, "Engineering tools for creation, integration and maintenance of DDIs V2," DEIS Project, 2019.

[9] SESAME consortium, "D7.1 Runtime Safety and Security Concept - EDDI Runtime Model Specification," 2022.

[10] SESAME consortium, "D7.2 Tools for Generation of Runtime EDDIs," 2022.

[11] SESAME consortium, "D5.6 Tools for Automated Security Analysis of MRS and for Production of EDDIs (Final Version)," 2023.

[12] SESAME consortium, "D6.6 Multi-Stage Quality Assurance Methodology for EDDI-Supported MRS," 2023.

[13] SESAME consortium, "D3.2 Executable Scenarios Workbench (Initial Version)," 2022.

[14] S. Urolagin, P. KV and N. Reddy, "Generalization capability of artificial neural network incorporated with pruning method," in *Advanced Computing, Networking and Security: International Conference, ADCONS*, Surathkal, India, , December 16-18 2011.

[15] S. Gerasimou, H. F. Eniser, A. Sen and A. Cakan, "Importance-Driven Deep Learn-ing System Testing.," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020.

[16] SESAME consortium, "D6.1 Assurance of Data Driven and Learning Components of EDDIs," 2022.

[17] SESAME consortium, "D6.4 Recommendations for EDDI Repair and Hardening," 2023.

[18] A. Schmidt, J. Reich and I. Sorokos, "Live In ConSerts: Model-Driven Runtime Safety Assurance on Microcontrollers, Edge, and Cloud," *17th European Dependable Computing Conference (EDCC),* pp. 61-66, 2021.

[19] S. Kabir and Y. Papadopoulos, "Applications of Bayesian Networks and Petri Nets in Safety, Reliability, and Risk Assessments: A Review," *Safety Science,* vol. 115, pp. 154-175, 2019.

[20] S. Kabir, I. Sorokos, K. Aslansefat, Y. Papadoulos, Y. Gheraibia, J. Reich, M. Saimler and R. Wei, "A runtime safety analysis concept for open adaptive systems," *International Symposium on Model-Based Safety and Assessment,* pp. 332-346, 2019.

[21] SESAME consortium, "D5.4 Tailorability of EDDIs," 2022.

[22] D. Bozhinoski, M. Oviedo, N. Garcia, H. Deshpande, G. van der Hoorn, J. Tjerngren, A. Wasowski and C. Corbato, "MROS: runtime adaptation for robot control architectures," *Advanced Robotics,* vol. 36, no. 11, pp. 502-518, 2022.

[23] SESAME consortium, "D1.1 Project Requirements," 2021.