SESAME
SECURE AND SAFE MULTI-ROBOT SYSTEMS

Project Number 101017258

D3.3 Executable Scenario Management

Version 1.0
30 June 2023
Final

Public Distribution

Bonn-Rhein-Sieg University

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

Project Partner Contact Information

Aero41 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
The Open Group Scott Hansen Rond Point Schuman 6, 5th 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 nicholas.matragkas@york.ac.uk	

Document Control

Version	Status	Date
0.1	Version ready for internal review	26 June 2023
0.2	Fixed links to models and public repositories	27 June 2023
0.3	Updates from internal feedback	28 June 2023
0.4	Updates from feedback from FHF	29 June 2023
1.0	Final version	30 June 2023

Table of Contents

List of Figures

Executive Summary

In this deliverable, we describe how the ExSce Management enables the storage and querying of provenance data for scenario executions. In the first section, we will give an overview of what provenance is, and the concepts and relations available in the W3C PROV specification used in the ExSce Management. In addition, we will briefly introduce metamorphic testing which is used to generate new scenarios and as a test oracle, validating whether a run "passes or fails" the test.

A case study based on the PAL use case is used as a running example throughout the deliverable to demonstrate how the ExSce management is used to model and manage the scenarios of a multi-robot system performing navigation tasks. We use the publicly available software of the PAL robots, which is based on ROS1, and uses the default navigation stack. Extending our specification to include currently unavailable components (e.g. task planner and task allocation) or ROS2 only requires that the relevant concepts can be transformed into (and conform to) the PROV specification.

First, we describe how scenario specifications are modelled and conform to the PROV specification. Execution data is obtained from transforming bag files into PROV documents, which can then be queried as new scenarios are generated. This includes scenario meta-information such as derivations from other scenarios, simulation timestamps and various metrics for the entire run and individual path segments. Oracles are modelled as relations between a so-called "base" scenario, resulting in two main types: baseline and metamorphic oracles. On one hand, baseline oracles compare new runs of a scenario against its past runs to account for non-determinism, noise and unexpected behaviours. On the other hand, metamorphic oracles are based on a different scenario from which the scenario in question was derived.

Secondly, we describe how input transformations can be used to generate new scenarios and the type of output relations that we consider as part of the metamorphic relations. This is enabled by the storage of the PROV data on a property graph using Neo4j, which is allows us to query and compose both input transformations and output relations. The PROV data of these transformations is also stored in the database, enabling stakeholders to trace how scenarios were derived, i.e., which operations generated which artefacts (e.g., task specifications) and which was the resulting scenario.

Finally, we present a tutorial to summarize how to employ the ExSce Management approach and tools using the PAL case study as an example and we discuss how the ExSce Management approach generalizes in the context of other ExSce tools, SESAME deliverables and use cases.

Please note that the contents of this deliverable were automatically converted from the ExSce Management documentation[1]. For the latest version, please visit the website.

[1] https://hbrs-sesame.github.io/exsce_management/

1 Introduction

1.1 Provenance

At its core, *provenance* describes the historical ownership of an object. In computer science, provenance refers to the *lineage of data*, that is, how data is created and used. In the ExSce workbench, provenance allows us to answer questions such as "Which scenario was used as a basis to generate *scenario X*?", "Which data was used to compute this metric?" "How many runs of scenario X *pass* the acceptance criteria?", "What would be the acceptance criteria for a new scenario with this specification?". Provenance describes agents, such as robots, and activities, such as executions or "runs", that produce, influence or deliver an entity, such as an object or data. In short, the ExSce workbench conforms to the W3C PROV specification[2] as a means to describe scenarios, their executions and their corresponding acceptance criteria; describe what data was collected and how it was used for the acceptance criteria on each run; and as part of the process to verify and validate that a system meets its requirements.

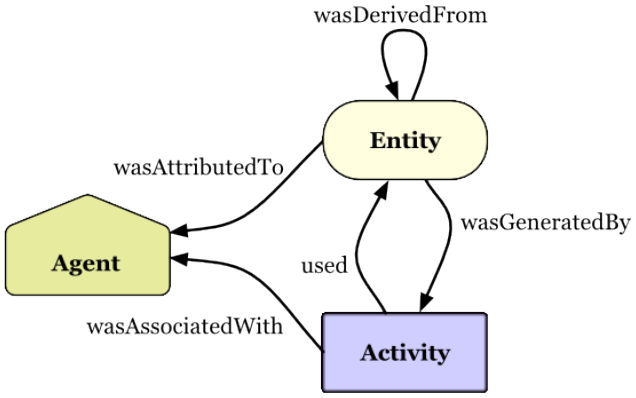

Figure 1: Key PROV concepts and relationships (from [1])

Here, we provide a small overview of the PROV models (cf. Figure 1). There are three main concepts: `Entities` represent physical, digital or conceptual things; `Activities` are used to represent actions or processes that *generate* or change entities; and `Agents` are responsible for or play a role in activities. The basic relationships between these concepts describe how activities *use* entities or *generate* new ones, and which agent was responsible (**`wasAssociatedWith`**) for an activity. **`wasAttributedTo`** describes which agent played a role in the activity that generated that entity and **`wasDerivedFrom`** describes that the existence or properties of an entity are due to another entity. A `Plan` is a special type of entity that describes the steps an agent followed while performing an activity. Figure 2 below shows a simple example of PROV models in the context of the ExSce Management.

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

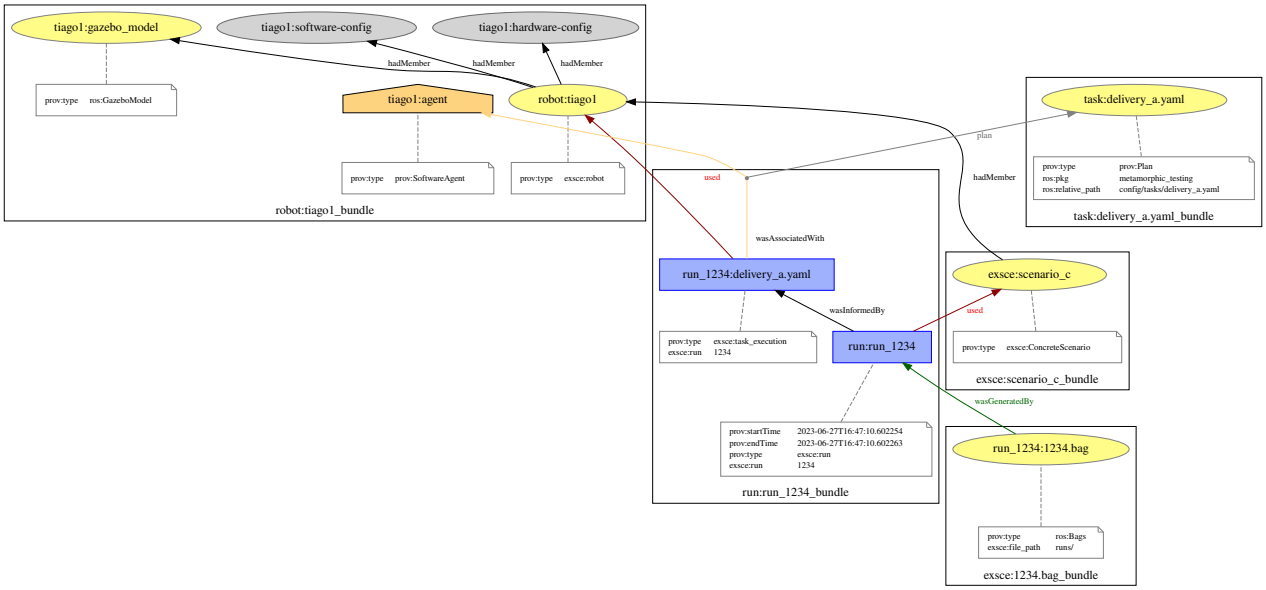

Figure 2: A simple example of the PROV concepts and relations modelling an executable scenario and execution data

1.2 Metamorphic testing

One of the challenges of testing (autonomous) robotic systems lies in the difficulty of defining whether the result of a test is correct or not[2]. A *test oracle* is the procedure by which determines whether the behaviour of the System Under Test (SUT) is the correct one, in principle comparing an "expected outcome" with the observed one. However, for complex software, it is challenging to predict what the *expected outcome* for a given input should be [3], e.g. how long a robot should take to complete a task depends on many factors. Furthermore, changes to the test inputs, even slight variations, can greatly influence the outcome.

Metamorphic testing [4] is a testing technique that leverages the relationships between the outputs of a system, instead of fully formalising its input-output behaviour. This is particularly well-suited for robotic systems, which often consider the robot(s) as a blackbox when it comes to testing. As a simple example, consider a robot and a task that consists of navigating from waypoint A to waypoint B. A *metamorphic relation*, illustrated in Figure 3, consists of an input transformation and an output relation. An input transformation, such as reverting the order of the waypoints, generates a new test case. An output relationship compares the outputs of this pair of test cases, e.g. the robot travels (roughly) the same distance when it navigates from A to B than from B to A. In metamorphic testing, a test oracle validates that the output relationship holds for a pair of test cases, i.e. rather than specifying the robot should travel exactly 5m from A to B, which might also depend on and vary because of robot parameters or properties (e.g. its kinematics), the test oracle checks that travelling from A to B and viceversa is (roughly) the same distance. Consider also that due to non-determinism and noise, robotic systems can exploit metamorphic relations even without user-defined input transformations, since the each run will have slightly different "inputs" (e.g. due to sensor noise); assuming we want the robot to behave the same regardless of those variations, we can define the output relation as invariant, e.g. distance to obstacles should always remain above some limit.

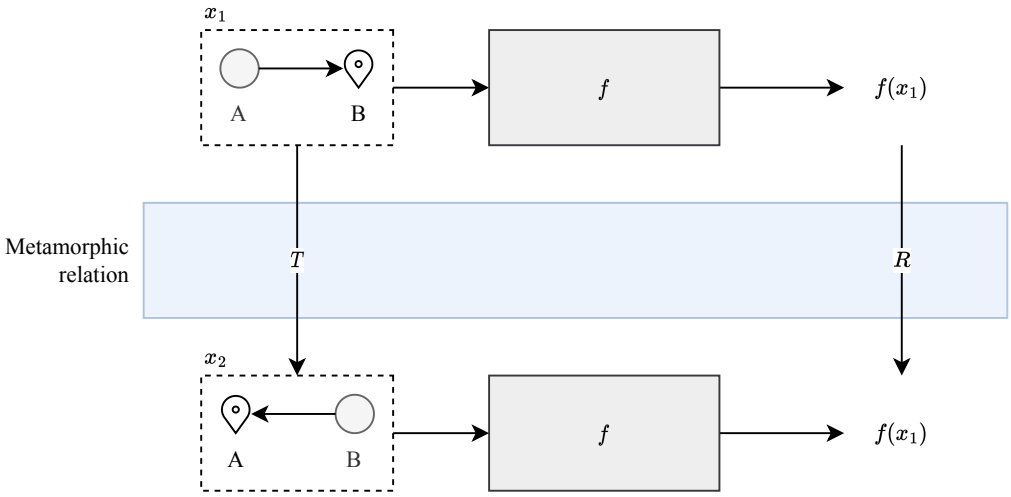

Figure 3: Example of a metamorphic relation for a simple navigation task. T is the input transformation, and R the output relation.

2 Case Study: Navigation tasks for a multi-robot system

In this section, we present a case study of a multi-robot system performing navigation tasks to be used as a running example in the rest of this document.

2.1 System Under Test

Table 1 summarizes the mobile robots in the multi-robot system considered for this case study:

Table 1: Caption: Robots considered in this case study

Robot ID	Model	Base
tiago1	tiago	pmb2
tiago2	tiago	pmb2
tiago3	tiago	omni_base
tiago4	tiago	omni_base
pmb1	pmb2	-
pmb2	pmb2	-
omni1	omni_base	-
omni2	omni_base	-

Their software is based on the Robot Operating System[3] (specifically, ROS1), and the examples shown in this document use their publicly available packages and simulation[4]. For a given test, a pre-defined mission consists of multiple tasks (i.e. there is no task planner). In turn, each task consists of a list of actions, and each action has a list of waypoints to be visited. The task allocation is done manually (testers choose which robots perform which tasks) and off-line (tasks are pre-assigned before the test starts). The robots use move_base and amcl for their navigation stack.

2.2 Environment

These robots are tasked with navigating between various locations in the ground floor of the Hochschule Bonn-Rhein-Sieg, as shown in Figure 4 below. The occupancy grid and gazebo models have been generated by the Floorplan DSL[5].

[3] https://www.ros.org/
[4] http://wiki.ros.org/Robots/TIAGo/Tutorials
[5] https://github.com/sesame-project/FloorPlan-DSL/

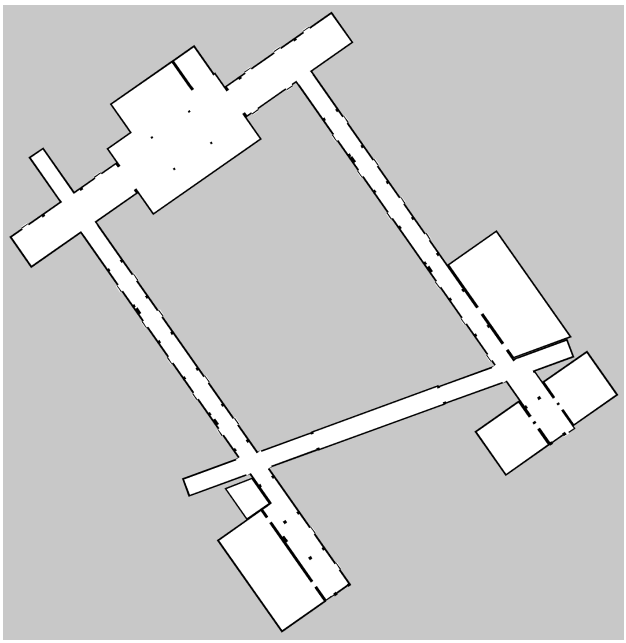

Figure 4: Environment occupancy grid for the case study

2.3 Tasks

In addition to some user-defined tasks, we make use of the tasks generated for the environment by the Floorplan DSL. Figure 5 below shows all available routes:

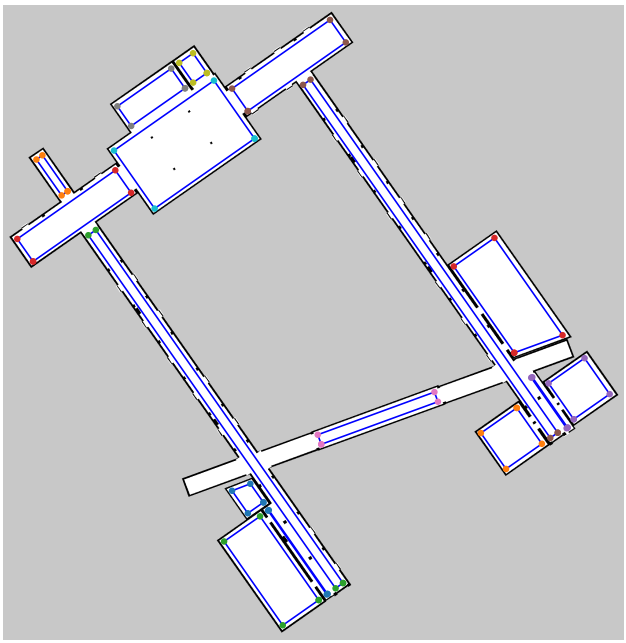

Figure 5: Tasks generated by the Floorplan DSL

2.4 Metrics

The testers are interested in validating the following aspects.

Performance:

- Mission duration - The overall time it takes for the selected robots to successfully complete all their tasks.
- Number of waypoints visited - How many waypoints each robot completed successfully.
- Distance travelled - How much did each robot travel.
- Task duration - How much time did it take a robot to complete all its tasks.

Safety:

- Robots must not exceed a maximum speed of 1.0 m/s
- Robots will maintain a distance of at least 10cm to all other obstacles.
- The distance between robots will be of 10cm at minimum.

3 Executable Scenarios

3.1 Specification

We consider the scenario specification as the description of the test input data, meaning we model the scenario components to the level of granularity that is relevant for our tests and *what* inputs we wish to vary. A scenario specification for our case study consists of the following basic ingredients:

- A unique ID,
- a mission that includes the tasks to be executed, their ordering, and their (static) task allocation,
- the robots which have been assigned to this mission, and
- the environment in which the scenario is to be executed, including the starting position of each robot and any obstacles.

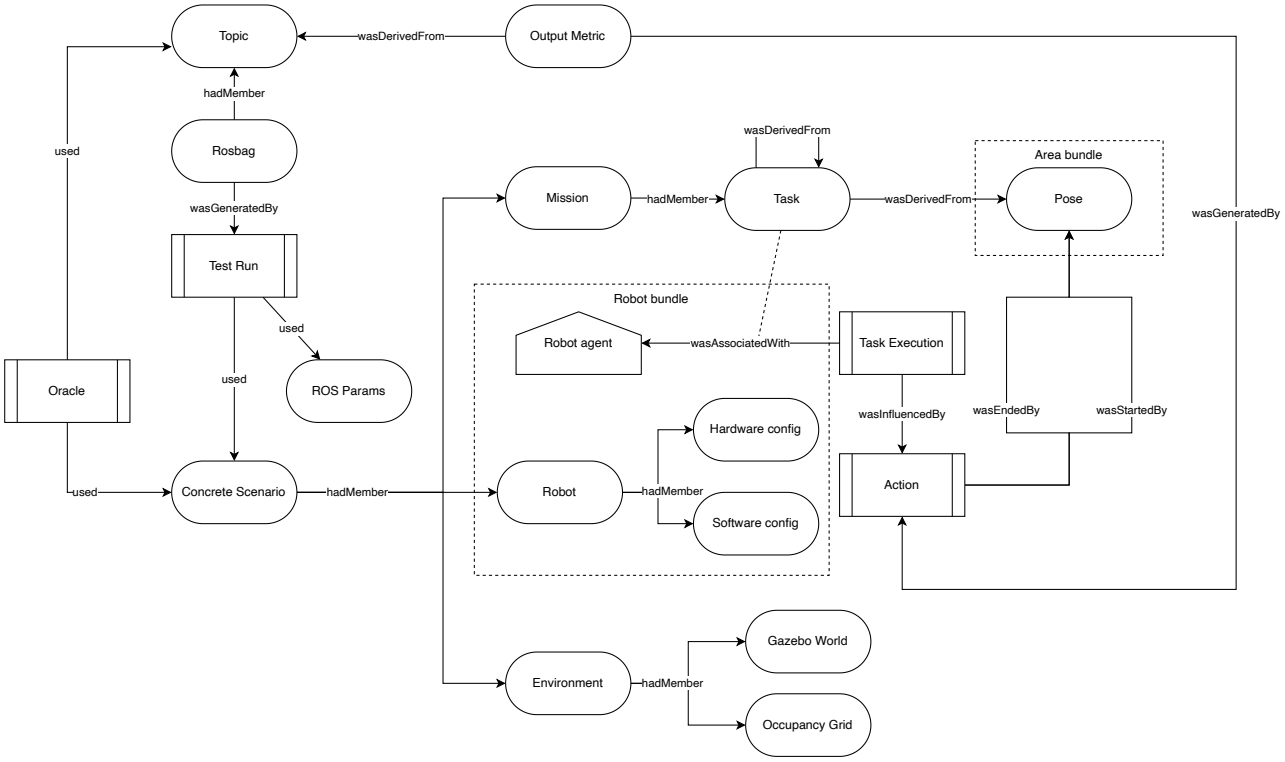

Figure 6: Overview of the PROV concepts and relations used in the ExSce Management

The specifications used in this example are available here[6]. Note that although we currently only support YAML[7] for the specification in our case study, the way in which a scenario is specified does not matter as long as a corresponding model-to-model transformation between the specification and the PROV models is available. Figure 6 shows an overview of the PROV concepts and relations described in this section. In this section, we show both the YAML-based specification, and the resulting transformation (conforming to the PROV meta-models described in this section) to PROV-N (a notation of PROV aimed at human consumption, e.g. Listing 1). In practice, we use the prov[8] library, which can serialize PROV documents to JSON.

[6] https://github.com/hbrs-sesame/models/blob/main/scenarios/scenario_c.yaml
[7] https://yaml.org/
[8] https://prov.readthedocs.io/en/latest/prov.html#

```
bundle exsce:scenario_c_bundle
  entity(exsce:scenario_c, [prov:type="exsce:ConcreteScenario"])

  hadMember(exsce:scenario_c, mission:mission_c)
  hadMember(exsce:scenario_c, env:brsu_building_c_with_doors)
  hadMember(exsce:scenario_c, robot:tiago1)
  hadMember(exsce:scenario_c, robot:tiago2)
endBundle
```

Listing 1: A scenario is modelled as a PROV collection, which has a mission, the environment and the robots as members.

3.1.1 Mission

For our case study we consider two types of missions: those that can be executed in parallel (each robot executes one task at a time) or sequential (only one robot is executing a task at any given time).

```
mission:
  id: mission_c
  type: parallel
  allocation:
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_a.yaml
    - robot: tiago2
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_b.yaml
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/navigate_home_1.yaml
    - robot: tiago2
      pkg: metamorphic_testing
      file_path: config/tasks/navigate_home_2.yaml
```

Listing 2: Mission specification in YAML format

The mission is a collection that models the specific instance of the tasks chosen and to which robot they were allocated to. The tasks are modelled separately to enable their reuse (they can be assigned multiple times to the same or another robot), and as we mentioned before, to abstract their concrete, domain or application-specific representation. Similarly, this allows us to use, for example, the tasks generated by the Floorplan DSL[9].

```
bundle exsce:scenario_c_bundle
  entity(mission:mission_c, [prov:type="exsce:mission", exsce:missionType="parallel"])
  entity(mission_c:tiago1-task_01, [prov:type="prov:Plan"])
  entity(mission_c:tiago2-task_01, [prov:type="prov:Plan"])
  entity(mission_c:tiago1-task_02, [prov:type="prov:Plan"])
  entity(mission_c:tiago2-task_02, [prov:type="prov:Plan"])

  hadMember(mission:mission_c, mission_c:tiago1-task_01)
  hadMember(mission:mission_c, mission_c:tiago2-task_01)
  hadMember(mission:mission_c, mission_c:tiago1-task_02)
  hadMember(mission:mission_c, mission_c:tiago2-task_02)

  wasDerivedFrom(mission_c:tiago1-task_01, task:delivery_a.yaml, -, -, -, [prov:type="prov:
      PrimarySource"])
  wasDerivedFrom(mission_c:tiago2-task_01, task:delivery_b.yaml, -, -, -, [prov:type="prov:
      PrimarySource"])
  wasDerivedFrom(mission_c:tiago1-task_02, task:navigate_home_1.yaml, -, -, -, [prov:type="prov:
      PrimarySource"])
  wasDerivedFrom(mission_c:tiago2-task_02, task:navigate_home_2.yaml, -, -, -, [prov:type="prov:
      PrimarySource"])
endBundle
```

Listing 3: PROV models to represent a mission.

[9] https://github.com/sesame-project/floorplan-dsl-replication-package

Tasks In this example, the tasks have been implicitly included as part of the allocation. Note also that there are different levels of abstraction, the scenario specification does not constrain *how* tasks should be specified, allowing us to reuse the scenario specification, and likely any task-agnostic transformations, for other types of systems with different task specification formats (given that there is a model to model transformation of the task to PROV).

A task is an entity which is derived from its subtask; each subtask in turn, is derived from the poses it contains as waypoints.

```
bundle task:delivery_a.yaml_bundle

  entity(task:delivery_a.yaml, [prov:type="prov:Plan", ros:pkg="metamorphic_testing", ros:
      relative_path="config/tasks/delivery_a.yaml"])

  entity(task:delivery_a.yaml-subtask_001, [prov:type="prov:Plan"])

  wasDerivedFrom(task:delivery_a.yaml, task:delivery_a.yaml-subtask_001, -, -, -, [prov:role="
      subtask_001"])
  wasDerivedFrom(delivery_a.yaml-subtask_001:w_001; task:delivery_a.yaml-subtask_001, pose:c022-w001
      -R0.00_P0.00_Y2.27, -, -, -, [prov:role="waypoint"])
  wasDerivedFrom(delivery_a.yaml-subtask_001:w_002; task:delivery_a.yaml-subtask_001, pose:c018-w001
      -R0.00_P0.00_Y-0.64, -, -, -, [prov:role="waypoint"])
endBundle
```

Listing 4: PROV models for a navigation task

Allocation In this case study, we have considered pre-defined assignments in the scenario specification. Note, however, that the allocation does not appear in the mission PROV shown above. To account for the cases where an algorithm allocates tasks at runtime, we collect the allocation as part of the runtime information. For this use case, this is done by obtaining the allocation from the scenario parameters before sending the tasks to the robots. We foresee that using an online allocator will require changes to capture the allocations depending on their implementation and how they are made available, perhaps as part of the model-to-model transformation discussed in Section 3.2.

As an example, Listing 5 shows the PROV models for the allocated tasks of `tiago1`. Each task is modelled as an activity assigned to a robot using the **`wasAssociatedWith`** relationship, with each task modelled in the mission added as the plan. Each Action is also an activity that uses the **`wasStartedBy`** to model the starting pose of the robot, and **`wasEndedBy`** its goal. The sequence of tasks is modelled via the **`wasInformedBy`** relationship, because, in this case study, activities are dependent on the successful completion of their previous activity.

```
bundle run:run_12345_bundle

  activity(run_12345:tiago1-task_01, -, -, [prov:type="exsce:task_execution", exsce:run="12345"])
  used(run_12345:tiago1-task_01, robot:tiago1, -)
  wasAssociatedWith(run_12345:tiago1-task_01, tiago1:agent, mission_c:tiago1-task_01)

  activity(run_12345:tiago1-task_01-subtask_001-w_001, -, -, [prov:type="exsce:action", exsce:run="
      12345"])
  wasEndedBy(run_12345:tiago1-task_01-subtask_001-w_001, pose:c022-w001-R0.00_P0.00_Y2.27, -, -, [
      prov:role="goal", exsce:runs="12345"])
  wasStartedBy(run_12345:tiago1-task_01-subtask_001-w_001, pose:c025-w001-R0.00_P0.00_Y0.00, -, -)
  wasAssociatedWith(run_12345:tiago1-task_01-subtask_001-w_001, tiago1:agent, -)

  activity(run_12345:tiago1-task_01-subtask_001-w_002, -, -, [prov:type="exsce:action", exsce:run="
      12345"])
  wasInformedBy(run_12345:tiago1-task_01-subtask_001-w_002, run_12345:tiago1-task_01-subtask_001-
      w_001)
  wasEndedBy(run_12345:tiago1-task_01-subtask_001-w_002, pose:c018-w001-R0.00_P0.00_Y-0.64, -, -, [
      prov:role="goal", exsce:runs="12345"])
  wasStartedBy(run_12345:tiago1-task_01-subtask_001-w_002, pose:c022-w001-R0.00_P0.00_Y2.27, -, -)
  wasAssociatedWith(run_12345:tiago1-task_01-subtask_001-w_002, tiago1:agent, -)

  wasInformedBy(run_12345:tiago1-task_01, run_12345:tiago1-task_01-subtask_001-w_001)
  wasInformedBy(run_12345:tiago1-task_01, run_12345:tiago1-task_01-subtask_001-w_002)
```

```
  activity(run_12345:tiago1-task_02, -, -, [prov:type="exsce:task_execution", exsce:run="12345"])
  used(run_12345:tiago1-task_02, robot:tiago1, -)
  wasAssociatedWith(run_12345:tiago1-task_02, tiago1:agent, mission_c:tiago1-task_02)

  activity(run_12345:tiago1-task_02-subtask_001-w_001, -, -, [prov:type="exsce:action", exsce:run="
      12345"])
  wasInformedBy(run_12345:tiago1-task_02-subtask_001-w_001, run_12345:tiago1-task_01-subtask_001-
      w_002)
  wasEndedBy(run_12345:tiago1-task_02-subtask_001-w_001, pose:c025-w001-R0.00_P0.00_Y0.93, -, -, [
      prov:role="goal", exsce:runs="12345"])
  wasStartedBy(run_12345:tiago1-task_02-subtask_001-w_001, pose:c018-w001-R0.00_P0.00_Y-0.64, -, -)
  wasAssociatedWith(run_12345:tiago1-task_02-subtask_001-w_001, tiago1:agent, -)
  wasInformedBy(run_12345:tiago1-task_02, run_12345:tiago1-task_02-subtask_001-w_001)

endBundle
```

Listing 5: PROV models for the (run-time) task allocation for tiago1.

3.1.2 Robots

Because we treat the robots as a black box, we have kept their specification very minimal, as shown in Listing 6. Each robot needs a unique ID, its type and, optionally for ROS, the namespace of its topics.

```
robots:
  - id: tiago1
    robot_namespace: tiago1
    robot_type: tiago
  - id: tiago2
    robot_namespace: tiago2
    robot_type: tiago
```

Listing 6: Fleet specification in YAML

A robot bundle considers the software agent that represents the robot, its hardware and software configuration, and the simulation model used, if any. Listing 7 shows the PROV-N version for `tiago1`.

```
bundle robot:tiago1_bundle
  entity(robot:tiago1, [prov:type="exsce:robot"])

  agent(tiago1:agent, [prov:type="prov:SoftwareAgent"])
  entity(tiago1:software-config, [prov:type="software-config"])
  entity(tiago1:hardware-config, [prov:type="hardware-config"])
  entity(tiago1:gazebo_model, [prov:type='ros:GazeboModel'])

  hadMember(robot:tiago1, tiago1:gazebo_model)
  hadMember(robot:tiago1, tiago1:software-config)
  hadMember(robot:tiago1, tiago1:hardware-config)
endBundle
```

Listing 7: PROV models to represent a robot.

3.1.3 Environment

The environment specification in Listing 8 consists of three main parts:

- The models that describe the environment, in this case the occupancy grid and the gazebo world for simulation,
- the models describing objects/obstacles not included in the gazebo world (useful to vary their locations) and their starting positions (in the example below no additional obstacles were added)
- the starting position of robots.

```
environment:
  id: brsu_building_c_with_doors
  models:
    map:
      map_name: brsu_building_c_with_doors
      pkg: floorplan-DSL-environments
      relative_path: maps/
    gazebo_world:
      model_name: brsu_building_c_with_doors
      pkg: floorplan-DSL-environments
      relative_path: worlds/
  robots:
    tiago1:
      start_pose:
        id: c025-w001
        x: 44.80387496948242
        y: 37.15502166748047
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
    tiago2:
      start_pose:
        id: c025-w002
        x: 43.432926177978516
        y: 38.493873596191406
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
```

Listing 8: Environment specification in YAML

The environment PROV (cf. Listing 9) is a collection of records for the simulation models and the environment's occupancy grid. For this case study, we use the environments generated by the Floorplan DSL[10].

```
bundle env:brsu_building_c_with_doors_bundle
  entity(env:brsu_building_c_with_doors, [prov:type="exsce:environment"])

  entity(brsu_building_c_with_doors:brsu_building_c_with_doors_map, [prov:type="exsce:map", ros:pkg=
      "floorplan-DSL-environments", ros:relative_path="maps/"])

  entity(brsu_building_c_with_doors:occupancy_grid, [prov:type="ros:OccupancyGrid"])
  hadMember(env:brsu_building_c_with_doors, brsu_building_c_with_doors:
      brsu_building_c_with_doors_map)
  entity(brsu_building_c_with_doors:metadata, [prov:type='ros:Parameter_Server'])
  hadMember(brsu_building_c_with_doors:brsu_building_c_with_doors_map, brsu_building_c_with_doors:
      metadata)
  hadMember(brsu_building_c_with_doors:brsu_building_c_with_doors_map, brsu_building_c_with_doors:
      occupancy_grid)

  entity(brsu_building_c_with_doors:gazebo_world, [prov:type='ros:GazeboModel', ros:pkg="floorplan-
      DSL-environments", ros:relative_path="worlds/"])
  hadMember(env:brsu_building_c_with_doors, brsu_building_c_with_doors:gazebo_world)
endBundle
```

Listing 9: PROV models to represent an environment

For simulation in particular, the starting positions of the robots are modeled as derivations as shown in Listing 10.

```
bundle exsce:scenario_c_bundle
  wasDerivedFrom(scenario_c:tiago1-start-pose; tiago1:gazebo_model, pose:c025-w001-R0.00_P0.00_Y0
      .00, -, -, -, [prov:role="start_pose", exsce:scenario="scenario_c"])
  wasDerivedFrom(scenario_c:tiago2-start-pose; tiago2:gazebo_model, pose:c025-w002-R0.00_P0.00_Y0
      .00, -, -, -, [prov:role="start_pose", exsce:scenario="scenario_c"])
endBundle
```

Listing 10: The PROV models for the starting positions of the robots in a scenario.

[10] https://github.com/sesame-project/FloorPlan-DSL

Finally, Listing 11 shows how poses in each environment are grouped in bundles that correspond to the room they are in.

```
bundle env:c022_bundle
  entity(pose:c022-w001-R0.00_P0.00_Y2.27, [prov:type="geom:Pose"])
  entity(pose:c022-w002-R0.00_P0.00_Y2.27, [prov:type="geom:Pose"])

  entity(point:c022-w001, [coord:x="52.8256" %% xsd:float, coord:y="32.5629" %% xsd:float, coord:z="
      0" %% xsd:float, prov:type="geom:Point"])
  entity(point:c022-w002, [coord:x="54.4976" %% xsd:float, coord:y="31.5202" %% xsd:float, coord:z="
      0" %% xsd:float, prov:type="geom:Point"])

  entity(orientation:R0.00_P0.00_Y2.27, [angle:roll="0" %% xsd:float, angle:pitch="0" %% xsd:float,
      angle:yaw="2.269" %% xsd:float, prov:type="geom:EulerAngles", geom:axes="sxyz"])

  hadMember(pose:c022-w001-R0.00_P0.00_Y2.27, point:c022-w001)
  hadMember(pose:c022-w002-R0.00_P0.00_Y2.27, point:c022-w002)

  hadMember(pose:c022-w001-R0.00_P0.00_Y2.27, orientation:R0.00_P0.00_Y2.27)
  hadMember(pose:c022-w002-R0.00_P0.00_Y2.27, orientation:R0.00_P0.00_Y2.27)
endBundle
```

Listing 11: PROV models to represent poses of an area in an environment.

3.2 Execution

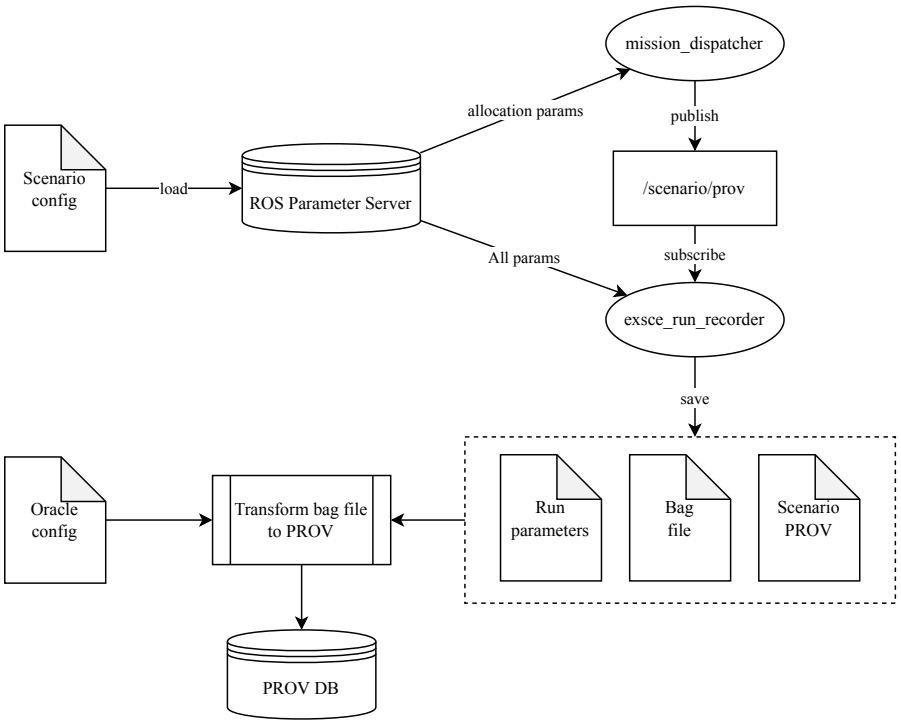

Figure 7: Scenario execution for ROS-based systems.

Figure 7 shows an overview of how scenario execution is integrated into a ROS-based system. For our implementation, the scenario specification is loaded to the ROS parameter server. A helper script that dispatches tasks to the robots also generates the run-time PROV for the task activities and their allocation as well as each action in the robots' tasks. This is published to a topic so it can be recorded in the rosbag file for the run. The `exsce_run_recorder` is used to generate the bag file and save the parameters from the server. These two activities are added to the run-time generated PROV, and stored in a JSON file.

The execution or test run is an activity that *uses* parameters from the ROS parameter server[11] and which *generates* a bag file as an artefact, as shown in Listing 12. The `run` activity is informed by the execution of the tasks described in Section 3.1.1.

```
bundle run:run_e30df0b6-09f0-11ee-ba45-13d500f34135_bundle

  activity(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, 2023-06-13T15:48:13.720444, 2023-06-13T15
      :56:32.315739, [prov:type="exsce:run", exsce:run="e30df0b6-09f0-11ee-ba45-13d500f34135"])

  entity(run_e30df0b6:parameter-server, [prov:type='ros:Parameter_Server', exsce:file_path="/home/
      argen/projects/tiago/tiago_ws/src/metamorphic_testing/runs/2023-06-13T15-48-07_scenario_c_run-
      e30df0b6.rosparams"])

  used(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, run_e30df0b6:parameter-server, -)
  used(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, exsce:scenario_c, -)

  wasGeneratedBy(run_e30df0b6:2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag, run:run_e30df0b6-09f0
      -11ee-ba45-13d500f34135, -)

  wasInformedBy(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, run_e30df0b6:tiago1-task_01)
  wasInformedBy(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, run_e30df0b6:tiago2-task_01)
  wasInformedBy(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, run_e30df0b6:tiago1-task_02)
  wasInformedBy(run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, run_e30df0b6:tiago2-task_02)

endBundle
```

Listing 12: PROV models to represent a test run in ROS and other run-time information

A bundle is created for each bag file, and the topics it contains are modelled as entities that are part of its collection, as shown in Listing 13:

```
bundle exsce:2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag_bundle
  entity(run_e30df0b6:2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag, [prov:type='ros:Bags', exsce:
      file_path="/home/argen/projects/tiago/tiago_ws/src/metamorphic_testing/runs/2023-06-13T15
      -48-07_scenario_c_run-e30df0b6.bag"])
  entity(topic:/tiago1/distance_travelled, [prov:type='ros:Topics'])
  hadMember(run_e30df0b6:2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag, topic:/tiago1/
      distance_travelled)
  entity(topic:/tiago1/nav_vel, [prov:type='ros:Topics'])
  hadMember(run_e30df0b6:2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag, topic:/tiago1/nav_vel)
endBundle
```

Listing 13: PROV models to represent a bag file and the topics it contains

[11] http://wiki.ros.org/Parameter_Server

3.2.1 Metrics and Monitors

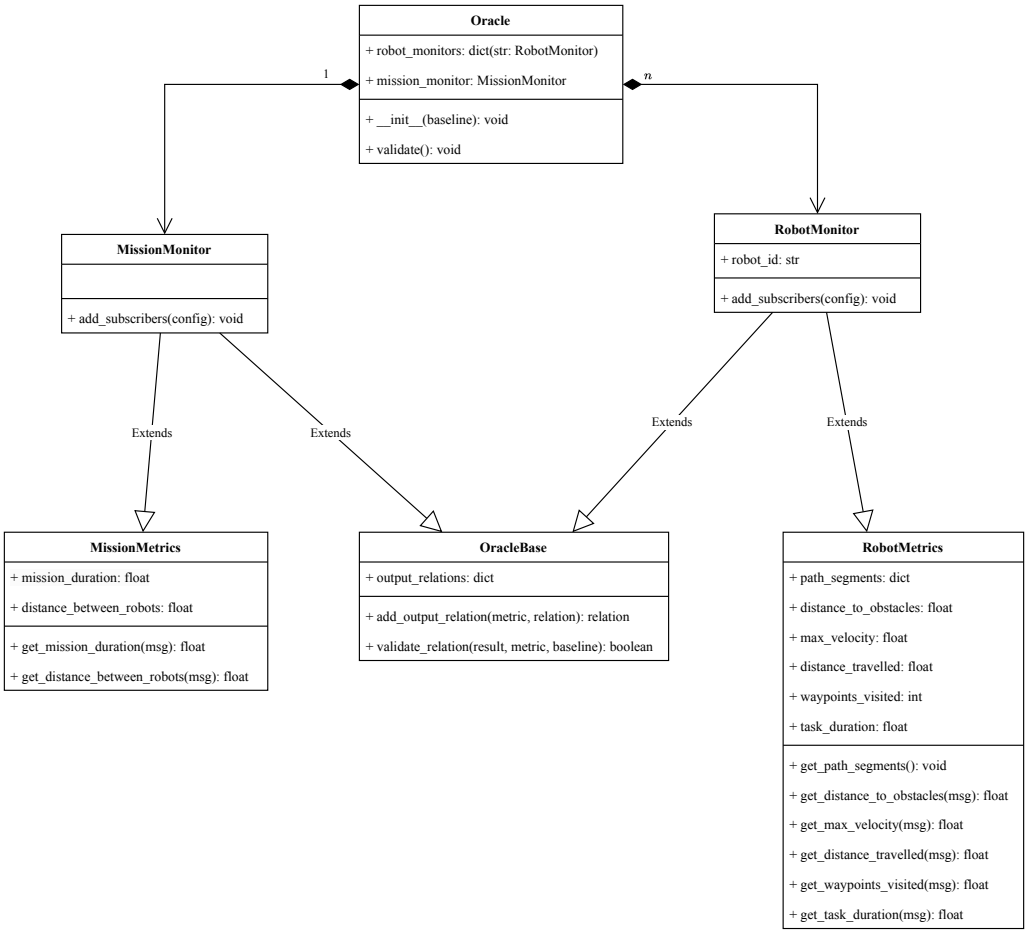

Figure 8: Oracles for a ROS-based multi-robot system are based on simple monitors that subscribe to the topics used as the data source.

Although we obtain the observed outputs off-line through bag files, the implementation to obtain the desired metrics follows the same logic and can be used as a run-time monitor that uses the publisher-subscriber pattern in ROS. These monitors can also be composed into the run-time oracle if desired, as shown in Figure 8. The relevant parts of the configuration for the bag transformation is shown in Listing 14 (`tiago2` is not included).

```
id: scenario_c
baseline:
  base_scenario: scenario_c
  mission:
    mission_duration:
      topics:
        - /tiago1/waypoint_dispatcher/feedback
        - /tiago2/waypoint_dispatcher/feedback
    distance_between_robots:
      topics:
        - /tiago1/robot_pose
        - /tiago2/robot_pose
  robots:
    tiago1:
      task_duration:
        topics:
        - /tiago1/waypoint_dispatcher/feedback
      waypoints_visited:
        topics:
        - /tiago1/waypoint_dispatcher/result
      distance_travelled:
        topics:
```

```
      - /tiago1/distance_travelled
    distance_to_obstacles:
      topics:
      - /tiago1/scan
    max_velocity:
      topics:
      - /tiago1/nav_vel
  tiago2:
    task_duration:
      topics:
      - /tiago2/waypoint_dispatcher/feedback
    waypoints_visited:
      topics:
      - /tiago2/waypoint_dispatcher/result
    distance_travelled:
      topics:
      - /tiago2/distance_travelled
    distance_to_obstacles:
      topics:
      - /tiago2/scan
    max_velocity:
      topics:
      - /tiago2/nav_vel
```

Listing 14: Configuration for the transformation of ROS1 bag files to ExSce PROV modelss

Metrics are computed from the topics published by the robots. A collection of topics models the data source that was used to compute the metric, and the relationship **wasGeneratedBy** links the output metric to the activity of the run that generated this result. Mission metrics take into account the entire run. To enable the output relation composition, when the bag file is being transformed, in addition to the overall mission metrics, these monitors compute the safety and robot performance metrics for each path segment, so that they can be individually queried and aggregated for other possible paths. In Listing 15 you see the PROV models for the `distance_travelled` metric of `tiago1`: One entity for the value of the entire run, but also the corresponding value for this metric on each path segment.

```
bundle run_e30df0b6:distance_travelled_bundle

  entity(run_e30df0b6:tiago1-distance_travelled-topics, [prov:type="exsce:topic"])
  hadMember(run_e30df0b6:tiago1-distance_travelled-topics, topic:/tiago1/distance_travelled)

  entity(tiago1:distance_travelled, [prov:type='exsce:output_metric', run:metricType="
      distance_travelled", exsce:run="e30df0b6-09f0-11ee-ba45-13d500f34135", prov:value="49.5234" %%
       xsd:float, exsce:robot="tiago1"])
  wasDerivedFrom(tiago1:distance_travelled, run_e30df0b6:tiago1-distance_travelled-topics, -, -, -,
      [prov:type="prov:PrimarySource"])
  wasGeneratedBy(tiago1:distance_travelled, run:run_e30df0b6-09f0-11ee-ba45-13d500f34135, -)

  entity(tiago1-task_01-subtask_001-w_001:distance_travelled, [prov:type='exsce:output_metric', run:
      metricType="distance_travelled", exsce:run="e30df0b6-09f0-11ee-ba45-13d500f34135", prov:value=
      "10.2956" %% xsd:float, exsce:robot="tiago1"])
  wasDerivedFrom(tiago1-task_01-subtask_001-w_001:distance_travelled, run_e30df0b6:tiago1-
      distance_travelled-topics, -, -, -, [prov:type="prov:PrimarySource"])
  wasGeneratedBy(tiago1-task_01-subtask_001-w_001:distance_travelled, run_e30df0b6:tiago1-task_01-
      subtask_001-w_001, -)

  entity(tiago1-task_01-subtask_001-w_002:distance_travelled, [prov:type='exsce:output_metric', run:
      metricType="distance_travelled", exsce:run="e30df0b6-09f0-11ee-ba45-13d500f34135", prov:value=
      "19.5078" %% xsd:float, exsce:robot="tiago1"])
  wasDerivedFrom(tiago1-task_01-subtask_001-w_002:distance_travelled, run_e30df0b6:tiago1-
      distance_travelled-topics, -, -, -, [prov:type="prov:PrimarySource"])
  wasGeneratedBy(tiago1-task_01-subtask_001-w_002:distance_travelled, run_e30df0b6:tiago1-task_01-
      subtask_001-w_002, -)

  entity(tiago1-task_02-subtask_001-w_001:distance_travelled, [prov:type='exsce:output_metric', run:
      metricType="distance_travelled", exsce:run="e30df0b6-09f0-11ee-ba45-13d500f34135", prov:value=
      "19.7156" %% xsd:float, exsce:robot="tiago1"])
  wasDerivedFrom(tiago1-task_02-subtask_001-w_001:distance_travelled, run_e30df0b6:tiago1-
      distance_travelled-topics, -, -, -, [prov:type="prov:PrimarySource"])
  wasGeneratedBy(tiago1-task_02-subtask_001-w_001:distance_travelled, run_e30df0b6:tiago1-task_02-
      subtask_001-w_001, -)
```

```
endBundle
```

Listing 15: PROV models for the distance_travelled metric of tiago1

3.3 Test Oracles

Oracles in PROV are modelled as activities that validate run results based on data from ROS topics. In Listing 16, the mission oracle and the oracle for one of the robots in one of our sample scenarios.

```
activity(scenario_c:mission_duration, -, -, [prov:type="exsce:oracle", oracle:metric="
    mission_duration", oracle:robot="None"])
used(scenario_c:mission_duration, topic:/tiago1/waypoint_dispatcher/feedback, -, [prov:role="topic"
    ])
used(scenario_c:mission_duration, topic:/tiago2/waypoint_dispatcher/feedback, -, [prov:role="topic"
    ])

activity(scenario_c:distance_between_robots, -, -, [prov:type="exsce:oracle", oracle:metric="
    distance_between_robots", oracle:robot="None"])
used(scenario_c:distance_between_robots, topic:/tiago1/robot_pose, -, [prov:role="topic"])
used(scenario_c:distance_between_robots, topic:/tiago2/robot_pose, -, [prov:role="topic"])


activity(scenario_c:tiago1-waypoints_visited, -, -, [prov:type="exsce:oracle", oracle:metric="
    waypoints_visited", oracle:robot="tiago1"])
used(scenario_c:tiago1-waypoints_visited, topic:/tiago1/waypoint_dispatcher/result, -, [prov:role="
    topic"])

activity(scenario_c:tiago1-distance_travelled, -, -, [prov:type="exsce:oracle", oracle:metric="
    distance_travelled", oracle:robot="tiago1"])
used(scenario_c:tiago1-distance_travelled, topic:/tiago1/distance_travelled, -, [prov:role="topic"])
```

Listing 16: PROV models for the mission and tiago1 test oracles

3.3.1 Relationships

The oracles above validate that *new* runs conform to the relationships between two tests. As such, they are modelled by `usage` relationships, where the activity uses the results of the base scenario for comparison. As arguments of the relationship we can add the type of output relation, its value, limit or delta, and its tolerance. In Listing 17 you see an example of a baseline oracle which compares new runs of `scenario_c` against its past runs.

```
bundle exsce:scenario_c_bundle

  entity(exsce:scenario_c, [prov:type="exsce:ConcreteScenario"])

  used(scenario_c:mission_duration, exsce:scenario_c, -, [oracle:relationship="invariant", oracle:
      package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value="97.606" %%
      xsd:float, oracle:tolerance="10" %% xsd:float])

  used(scenario_c:distance_between_robots, exsce:scenario_c, -, [oracle:relationship="above_min",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:limit="0.1"
      %% xsd:float])

  used(scenario_c:tiago1-task_duration, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value="97.606
      " %% xsd:float, oracle:tolerance="5" %% xsd:float])

  used(scenario_c:tiago1-waypoints_visited, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value=3])

  used(scenario_c:tiago1-distance_travelled, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value="
      50.7029" %% xsd:float, oracle:tolerance="2" %% xsd:float])
```

```
  used(scenario_c:tiago1-distance_to_obstacles, exsce:scenario_c, -, [oracle:relationship="above_min
      ", oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:limit="0.1
      " %% xsd:float])

  used(scenario_c:tiago1-max_velocity, exsce:scenario_c, -, [oracle:relationship="below_max", oracle
      :package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:limit="1" %% xsd:
      float])

  used(scenario_c:tiago2-task_duration, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value="89.804
      " %% xsd:float, oracle:tolerance="5" %% xsd:float])

  used(scenario_c:tiago2-waypoints_visited, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value=3])

  used(scenario_c:tiago2-distance_travelled, exsce:scenario_c, -, [oracle:relationship="invariant",
      oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:value="
      59.5331" %% xsd:float, oracle:tolerance="2" %% xsd:float])

  used(scenario_c:tiago2-distance_to_obstacles, exsce:scenario_c, -, [oracle:relationship="above_min
      ", oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:limit="0.1
      " %% xsd:float])

  used(scenario_c:tiago2-max_velocity, exsce:scenario_c, -, [oracle:relationship="below_max", oracle
      :package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:limit="1" %% xsd:
      float])
endBundle
```

Listing 17: Example of a baseline oracle for scenario_c.

Basic relationships We consider the following basic output relations:

- increasing(result, value, delta, tolerance=None)
- decreasing(result, value, delta, tolerance=None)
- invariant(result, value, tolerance=None)

These are useful for metrics which are straight forward to observe or those that are invariant around a constant. Two special types of invariant relationships are:

- below_max(result, limit)
- above_min(result, limit)

Complex relationships Complex relationships use or combine one or more of

- the basic output relationships,
- relationships composing input transformations, and
- relationships that derive or aggregate metrics based on inputs according to domain or application-specific logic

3.3.2 Oracles

Baseline oracles Safety-related oracles are usually invariant around a constant, i.e. regardless of the input transformation, the robot should behave the same:

Table 2: Output relations for safety metrics

Metric	Output relation	Limit
Max. velocity	below_max	1.0
Dist. to obstacles	above_min	0.1
Dist. between robots	above_min	0.1

Performance-related oracles where we compare runs of the same scenario are *invariant* around the average of the metric. We use the standard deviation for the tolerance.

Metamorphic oracles The output relations for the performance metrics require complex relationships that depend on specific scenarios being used. Consider the following example:

Table 3: Example of a complex metamorphic relation

Metric	Input transformation	Output relation
Dist. travelled	Add new waypoint	Increasing
Dist. travelled	Remove one waypoint	Decreasing
Dist. travelled	Invert waypoint order	Invariant

The value used as a baseline depends on the specific base scenario, and the delta depends on the specific path of the new task. Furthermore, output relationships are not straight forward to define when multiple input transformations are applied; however, using the provenance from existing runs we can also compose outputs to determine what is the output relation, e.g., query the results of previous runs to obtain the distance travelled between each pair of waypoints of the new path. Querying will be described in more detail in Section 5.

4 Metamorphic Relations

4.1 Input transformations

New scenarios are generated by applying input transformations to existing scenarios. The scenario genration is an activity that is informed by the individual transformations to the inputs (each modelled as an activity as well). The sequence of transformations are modelled with **wasInformedBy**, and the first and transformation are the activities associated with the `start` and `end`.

```
bundle scenario_780741116:scenario_generation
  activity(scenario_780741116:generation, -, -)
  wasGeneratedBy(exsce:scenario_780741116, scenario_780741116:generation, -)
  used(scenario_780741116:generation, exsce:scenario_c, -)

  activity(scenario_780741116:remove_robot-tiago2, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_robot", exsce:robot="tiago2"])
  wasStartedBy(scenario_780741116:generation, -, scenario_780741116:remove_robot-tiago2, -)

  ...

  activity(scenario_780741116:remove_start_pose-tiago2, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_start_pose"])
  wasInformedBy(scenario_780741116:remove_start_pose-tiago2, scenario_780741116:
      substitute_allocation-allocation_01-task_03)
  wasEndedBy(scenario_780741116:generation, -, scenario_780741116:remove_start_pose-tiago2, -)

endBundle
```

Listing 18: Excerpt of the PROV models for a scenario transformation.

Below we describe the PROV models for some of the transformations we considered for this case study. Note that transformations can be composed of other transformations (e.g. removing a robot).

4.1.1 Mission

Add allocated task To add a new allocated task, one needs to specify the task, the robot the task is allocated to, and the position in which to insert the new task. Listing 19 shows how this is modelled in PROV.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:add_task-config/tasks/delivery_a.yaml, -, -, [prov:type="exsce:transform",
      exsce:transform="add_task", exsce:pkg="metamorphic_testing", exsce:allocation_order=1, exsce:
      robot="tiago1"])
endBundle
```

Listing 19: PROV models for a scenario transformation where a new task is added and allocated

Remove task Listing 20 and Listing 21 show the PROV models for removing tasks that match a task ID and a robot ID, respectively.:

```
bundle new_scenario:scenario_generation
  activity(new_scenario:remove_task-config/tasks/navigate_home_1.yaml, -, -, [prov:type="exsce:
      transform", exsce:transform="remove_task"])
endBundle
```

Listing 20: Scenario transformation by removing a task that matches a task ID

```
bundle new_scenario:scenario_generation
  activity(new_scenario:remove_task-tiago2, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_task"])
endBundle
```

Listing 21: Scenario transformation by removing a task that matches a robot ID

Substitute task Substituting tasks in the mission adds one activity per task to be substituted. Listing 22 shows the PROV-N representation of this transformation.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:substitute_tasks-config/tasks/delivery_b.yaml, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_tasks", prov:value="config/tasks/delivery_c.yaml"])
endBundle
```

Listing 22: Scenario transformation: Substituting tasks

Substitute allocation The transformation in Listing 23 requires the specification of a new allocation as described in Section 3.1.1.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=1, exsce:robot="tiago1"])

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=2, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=3, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=4, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)
endBundle
```

Listing 23: Scenario transformation to substitute a scenario's task allocation

Swap assignments The transformation in Listing 24 takes the existing allocation and changes the assignment according to a set of key-value pairs, where the value is the robot ID that should substitute the robot ID in the key. The new assignments are then applied by using *substitute allocation*.

```
bundle new_scenario:scenario_generation

  activity(new_scenario:swap_assignment-tiago1, -, -, [prov:type="exsce:transform", exsce:transform=
      "swap_assignment", prov:value="tiago2", exsce:robot="tiago1"])

  activity(new_scenario:swap_assignment-tiago2, -, -, [prov:type="exsce:transform", exsce:transform=
      "swap_assignment", prov:value="tiago1", exsce:robot="tiago2"])
  wasInformedBy(new_scenario:swap_assignment-tiago2, new_scenario:swap_assignment-tiago1)

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=1, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:
      swap_assignment-tiago2)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=2, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)
```

```
  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=3, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=4, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)
endBundle
```

Listing 24: Scenario transformation: swap task assignments between robots in the scenario

Reverse task order This transformation uses *substitute allocation* to invert the order in which the tasks should the completed. Listing 25 shows this transformation's provenance.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:reverse_task_order-mission, -, -, [prov:type="exsce:transform", exsce:
      transform="reverse_task_order"])

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=1, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:
      reverse_task_order-mission)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=2, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=3, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=4, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)
endBundle
```

Listing 25: PROV models for a scenario transformation that reverses the task ordering

4.1.2 Task

A transformation to reverse the order in which waypoints of a task are visited generates a new task specification. Then it uses substitute task to replace the original task with the new specification that contains the inverted waypoints. This transformation is domain-specific, as the nature of the application determines whether the order of waypoints can be inverted or if it requires some preconditions or application logic.

```
bundle new_scenario:scenario_generation

  activity(new_scenario:reverse_waypoint_order-config/tasks/delivery_a.yaml, -, -, [prov:type="exsce
      :transform", exsce:transform="reverse_waypoint_order"])
  used(new_scenario:reverse_waypoint_order-config/tasks/delivery_a.yaml, task:delivery_a.yaml, -)
```

```
  wasGeneratedBy(task:delivery_a_reversed.yaml, new_scenario:reverse_waypoint_order-config/tasks/
      delivery_a.yaml, -)

  activity(new_scenario:substitute_tasks-config/tasks/delivery_a.yaml, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_tasks", prov:value="config/tasks/delivery_a_reversed.
      yaml"])
  wasInformedBy(new_scenario:substitute_tasks-config/tasks/delivery_a.yaml, new_scenario:
      reverse_waypoint_order-config/tasks/delivery_a.yaml)
endBundle
```

Listing 26: PROV models for the scenario transformation to reverse the waypoint sequence in a task

4.1.3 Environment

Swap start poses Similar to *swap assignment*, we can change robots' starting positions using key-value pairs the robot IDs involved in the swap, as shown in Listing 27.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:swap_start_poses-tiago1, -, -, [prov:type="exsce:transform", exsce:transform
      ="swap_start_poses", prov:value="tiago2"])

  activity(new_scenario:swap_start_poses-tiago2, -, -, [prov:type="exsce:transform", exsce:transform
      ="swap_start_poses", prov:value="tiago1"])
  wasInformedBy(new_scenario:swap_start_poses-tiago2, new_scenario:swap_start_poses-tiago1)
endBundle
```

Listing 27: PROV models to represent a scenaroi transformation by swapping robots' starting positions

4.1.4 Robots

Add robot Adding a new robot requires that we know its specification (as described in Section 3.1.2) and its starting position. The transformation uses then other transformations for the tasks related to this new robot, e.g., with *add task* (Listing 28) new tasks can be added to the mission. In addition, one can modify the allocation by using (1) *substitute allocation* (Listing 29) or by (2) using *swap assignments* to reassign tasks from one robot to the new one (as in Listing 30).

```
bundle new_scenario:scenario_generation
  activity(new_scenario:add_task-config/tasks/delivery_c.yaml, -, -, [prov:type="exsce:transform",
      exsce:transform="add_task", exsce:pkg="metamorphic_testing", exsce:allocation_order=4, exsce:
      robot="tiago3"])
  activity(new_scenario:add_start_pose-new-pose, -, -, [prov:type="exsce:transform", exsce:transform
      ="add_start_pose", exsce:robot="tiago3"])
  wasInformedBy(new_scenario:add_start_pose-new-pose, new_scenario:add_task-config/tasks/delivery_c.
      yaml)
  activity(new_scenario:add_robot-tiago3, -, -, [prov:type="exsce:transform", exsce:transform="
      add_robot"])
  wasInformedBy(new_scenario:add_robot-tiago3, new_scenario:add_start_pose-new-pose)
endBundle
```

Listing 28: PROV models for adding a new robot with new tasks in a scenario transform

```
bundle new_scenario:scenario_generation
  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_c.yaml", exsce:allocation_order=1, exsce:robot="tiago1"])

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=2, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)
```

```
  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=3, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=4, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)

  activity(new_scenario:substitute_allocation-allocation_01-task_05, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=5, exsce:robot="tiago3"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_05, new_scenario:
      substitute_allocation-allocation_01-task_04)

  activity(new_scenario:add_start_pose-new-pose, -, -, [prov:type="exsce:transform", exsce:transform
      ="add_start_pose", exsce:robot="tiago3"])
  wasInformedBy(new_scenario:add_start_pose-new-pose, new_scenario:substitute_allocation-
      allocation_01-task_05)

  activity(new_scenario:add_robot-tiago3, -, -, [prov:type="exsce:transform", exsce:transform="
      add_robot"])
  wasInformedBy(new_scenario:add_robot-tiago3, new_scenario:add_start_pose-new-pose)
endBundle
```

Listing 29: PROV models for adding a new robot with a new allocation in a scenario transform

```
bundle new_scenario:scenario_generation
  activity(new_scenario:swap_assignment-tiago3, -, -, [prov:type="exsce:transform", exsce:transform=
      "swap_assignment", prov:value="tiago1", exsce:robot="tiago3"])

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=1, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:
      swap_assignment-tiago3)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=2, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=3, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=4, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)

  activity(new_scenario:add_task-config/tasks/delivery_c.yaml, -, -, [prov:type="exsce:transform",
      exsce:transform="add_task", exsce:pkg="metamorphic_testing", exsce:allocation_order=4, exsce:
      robot="tiago1"])
  wasInformedBy(new_scenario:add_task-config/tasks/delivery_c.yaml, new_scenario:
      substitute_allocation-allocation_01-task_04)

  activity(new_scenario:add_start_pose-new-pose, -, -, [prov:type="exsce:transform", exsce:transform
      ="add_start_pose", exsce:robot="tiago3"])
  wasInformedBy(new_scenario:add_start_pose-new-pose, new_scenario:add_task-config/tasks/delivery_c.
      yaml)
```

```
  activity(new_scenario:add_robot-tiago3, -, -, [prov:type="exsce:transform", exsce:transform="
      add_robot"])
  wasInformedBy(new_scenario:add_robot-tiago3, new_scenario:add_start_pose-new-pose)
endBundle
```

Listing 30: PROV models for adding a new robot and reassigning existing tasks to it in a scenario transform

Remove robot Similarly, removing a robot uses additional transformations to handle the tasks of the robot being removed. The simplest case is to remove the tasks of the robot in question (Listing 31). Alternatively, one can specify a new allocation that updates the assignment (Listing 32) or one can reassign the tasks of the removed robot to another robot (Listing 33).

```
bundle new_scenario:scenario_generation
  activity(new_scenario:remove_robot-tiago2, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_robot", exsce:robot="tiago2"])

  activity(new_scenario:remove_task-tiago2, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_task"])
  wasInformedBy(new_scenario:remove_task-tiago2, new_scenario:remove_robot-tiago2)

  activity(new_scenario:remove_start_pose-tiago2, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_start_pose"])
  wasInformedBy(new_scenario:remove_start_pose-tiago2, new_scenario:remove_task-tiago2)
endBundle
```

Listing 31: Scenario transformation: Remove a robot and its tasks

```
bundle new_scenario:scenario_generation
  activity(new_scenario:remove_robot-tiago2, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_robot", exsce:robot="tiago2"])

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_c.yaml", exsce:allocation_order=1, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:remove_robot-
      tiago2)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=2, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=3, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:remove_start_pose-tiago2, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_start_pose"])
  wasInformedBy(new_scenario:remove_start_pose-tiago2, new_scenario:substitute_allocation-
      allocation_01-task_03)
endBundle
```

Listing 32: Scenario transformation: Remove a robot and add a replace the task allocation

```
bundle new_scenario:scenario_generation
  activity(new_scenario:remove_robot-tiago2, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_robot", exsce:robot="tiago2"])

  activity(new_scenario:swap_assignment-tiago2, -, -, [prov:type="exsce:transform", exsce:transform=
      "swap_assignment", prov:value="tiago1", exsce:robot="tiago2"])
  wasInformedBy(new_scenario:swap_assignment-tiago2, new_scenario:remove_robot-tiago2)

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=1, exsce:robot="tiago1"])
```

```
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:
      swap_assignment-tiago2)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=2, exsce:robot="tiago1"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=3, exsce:robot="tiago1"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:remove_start_pose-tiago2, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_start_pose"])
  wasInformedBy(new_scenario:remove_start_pose-tiago2, new_scenario:substitute_allocation-
      allocation_01-task_03)
endBundle
```

Listing 33: Scenario transformation: Removing a robot and reassigning its tasks

Replace robot Replacing a robot is the combination of *add robot* and *remove robot*, and swapping the assignments and start position of the new robot with the old one. Listing 34 shows the provenance of this transformation.

```
bundle new_scenario:scenario_generation
  activity(new_scenario:replace_robot-tiago1, -, -, [prov:type="exsce:transform", exsce:transform="
      replace_robot"])

  activity(new_scenario:add_start_pose-c025-w001, -, -, [prov:type="exsce:transform", exsce:
      transform="add_start_pose", exsce:robot="tiago3"])
  wasInformedBy(new_scenario:add_start_pose-c025-w001, new_scenario:replace_robot-tiago1)

  activity(new_scenario:add_robot-tiago3, -, -, [prov:type="exsce:transform", exsce:transform="
      add_robot"])
  wasInformedBy(new_scenario:add_robot-tiago3, new_scenario:add_start_pose-c025-w001)

  activity(new_scenario:remove_robot-tiago1, -, -, [prov:type="exsce:transform", exsce:transform="
      remove_robot", exsce:robot="tiago1"])
  wasInformedBy(new_scenario:remove_robot-tiago1, new_scenario:add_robot-tiago3)

  activity(new_scenario:swap_assignment-tiago1, -, -, [prov:type="exsce:transform", exsce:transform=
      "swap_assignment", prov:value="tiago3", exsce:robot="tiago1"])
  wasInformedBy(new_scenario:swap_assignment-tiago1, new_scenario:remove_robot-tiago1)

  activity(new_scenario:substitute_allocation-allocation_01-task_01, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_a.yaml", exsce:allocation_order=1, exsce:robot="tiago3"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_01, new_scenario:
      swap_assignment-tiago1)

  activity(new_scenario:substitute_allocation-allocation_01-task_02, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/delivery_b.yaml", exsce:allocation_order=2, exsce:robot="tiago2"])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_02, new_scenario:
      substitute_allocation-allocation_01-task_01)

  activity(new_scenario:substitute_allocation-allocation_01-task_03, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
      file_path="config/tasks/navigate_home_1.yaml", exsce:allocation_order=3, exsce:robot="tiago3"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_03, new_scenario:
      substitute_allocation-allocation_01-task_02)

  activity(new_scenario:substitute_allocation-allocation_01-task_04, -, -, [prov:type="exsce:
      transform", exsce:transform="substitute_allocation", exsce:pkg="metamorphic_testing", exsce:
```

```
      file_path="config/tasks/navigate_home_2.yaml", exsce:allocation_order=4, exsce:robot="tiago2"
      ])
  wasInformedBy(new_scenario:substitute_allocation-allocation_01-task_04, new_scenario:
      substitute_allocation-allocation_01-task_03)

  activity(new_scenario:remove_start_pose-tiago1, -, -, [prov:type="exsce:transform", exsce:
      transform="remove_start_pose"])
  wasInformedBy(new_scenario:remove_start_pose-tiago1, new_scenario:substitute_allocation-
      allocation_01-task_04)
endBundle
```

Listing 34: PROV models for a transformation to replace a robot with a new one

4.2 Output relations

The output relations take advantage of the information stored in the property graph. First, we query the baseline relationships of base scenario. To get the oracle config for the new scenario, we copy the safety relations from the base scenario; we assume these are invariant relations, which don't change. Next we get the new expected outputs by computing basic output relations (e.g., counting the number of waypoints in free space of the tasks in the new scenario) or querying existing path segments for complex relations. For example, we query the average time it takes a robot to travel between each pair of waypoints in the path that results from the new task, and aggregate them. If no data has been recorded for the metric in question for that particular path segment, we use an estimate. Finally, we compare the new expected outcomes against the base scenario and identify which basic relation applies.

Listing 35 shows an example of the generated oracle configuration, containing the baseline and metamorphic oracles for a transformation where we removed `tiago2`. Note that the baseline is with respect to itself, but the metamorphic config uses the results of the scenario (`scenario_c`) it was derived from. This has the advantage of being able to validate its results once enough data has been collected from runs of this new scenario, but also being able to use any new data collected in `scenario_c` for its validation.

```
id: new_scenario
baseline:
  base_scenario: new_scenario
  mission:
    mission_duration:
      baseline:
        tolerance: 0.0
        value: 141.19514473468308
      relationship:
        name: invariant
        package: exsce.metamorphic.relationships
      topics:
        - /tiago1/waypoint_dispatcher/feedback
  robots:
    tiago1:
      distance_travelled:
        baseline:
          tolerance: 0.0
          value: 98.24204030780109
        relationship:
          name: invariant
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/distance_travelled
      task_duration:
        baseline:
          tolerance: 0.0
          value: 141.19514473468308
        relationship:
          name: invariant
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/waypoint_dispatcher/feedback
```

```
      waypoints_visited:
        baseline:
          tolerance: null
          value: 5
        relationship:
          name: invariant
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/waypoint_dispatcher/result
metamorphic:
  base_scenario: scenario_c
  mission:
    mission_duration:
      baseline:
        delta: 43.58914473468309
        tolerance: 0.0
        value: 141.19514473468308
      relationship:
        name: increasing
        package: exsce.metamorphic.relationships
      topics:
        - /tiago1/waypoint_dispatcher/feedback
  robots:
    tiago1:
      distance_travelled:
        baseline:
          delta: 47.539097638514015
          tolerance: 0.0
          value: 50.702942669287076
        relationship:
          name: increasing
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/distance_travelled
      task_duration:
        baseline:
          delta: 43.58914473468309
          tolerance: 0.0
          value: 97.606
        relationship:
          name: increasing
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/waypoint_dispatcher/feedback
      waypoints_visited:
        baseline:
          delta: 2
          tolerance: null
          value: 3
        relationship:
          name: increasing
          package: exsce.metamorphic.relationships
        topics:
          - /tiago1/waypoint_dispatcher/result
```

Listing 35: Adding output relations to oracle configuration file

As shown in Listing 36, the oracle's PROV is the same activity using the same topics as data sources, but with different *usage* relationships for the "baseline" and "metamorphic" oracles, each corresponding to the scenario that provides the runs to use as source data for the metric in question.

```
bundle exsce:scenario_780741116_bundle
  entity(exsce:scenario_780741116, [prov:type="exsce:ConcreteScenario"])

  activity(scenario_780741116:mission_duration, -, -, [prov:type="exsce:oracle", oracle:metric="
      mission_duration", oracle:robot="None"])
  used(scenario_780741116:mission_duration, exsce:scenario_780741116, -, [oracle:relationship="
      invariant", oracle:package="exsce.metamorphic.relationships", oracle:type="baseline", oracle:
      value="141.195" %% xsd:float, oracle:tolerance="0" %% xsd:float])
  used(scenario_780741116:mission_duration, exsce:scenario_c, -, [oracle:relationship="increasing",
      oracle:package="exsce.metamorphic.relationships", oracle:type="metamorphic", oracle:value="
```

```
      141.195" %% xsd:float, oracle:tolerance="0" %% xsd:float, oracle:delta="43.5891" %% xsd:float
      ])

  used(scenario_780741116:mission_duration, topic:/tiago1/waypoint_dispatcher/feedback, -, [prov:
      role="topic"])
  used(scenario_780741116:mission_duration, topic:/tiago1/waypoint_dispatcher/feedback, -, [prov:
      role="topic"])

endBundle
```

Listing 36: PROV models for a baseline oracle

4.2.1 Estimating metrics

As an example, we present a simplistic approach to estimate the `task_duration` and `distance_travelled` metrics. The exact way of estimating missing data should consider the context-specific requirements that take into account robot, application, and domain specific knowledge.

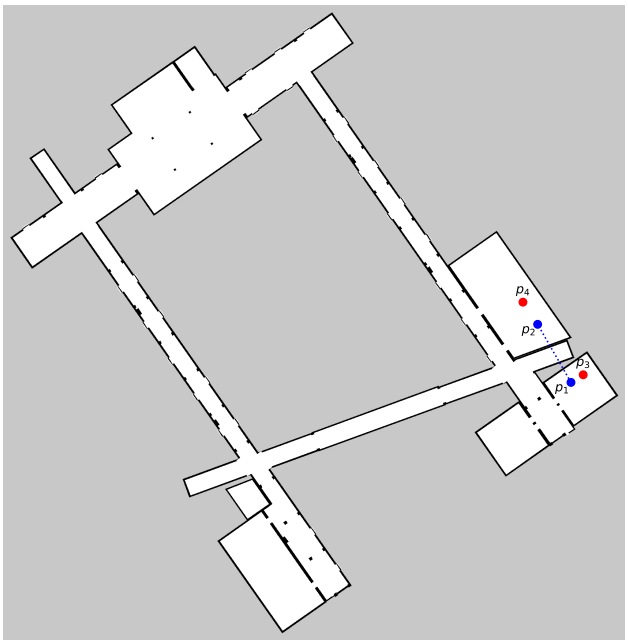

Figure 9: Points of reference used to estimate the `task_duration` and `distance_travelled` metrics when no available data exists

Consider the path segment between two waypoints p_1 and p_2 (cf. Figure 9) which does not have data available for the metric $m_{1\to2}$ that we are interested in, neither in the direction of travel of the task $p_1 \to p_2$ nor in the opposite direction $p_1 \leftarrow p_2$, i.e., $m_{2\to1}$. To estimate the metric, we first query the PROV database for all the poses in the same room as p_1 and p_2, and sort them by the shortest distance, respectively. Let us say that p_3 and p_4 are the closest poses to p_1 and p_2, respectively. We query the metrics for the path segments between (p_3, p_2), (p_1, p_4) and between (p_3, p_4) (in either direction, but preferring the same traveling direction as the task):

$$m_{3\leftrightarrow2} = query\_metric(metric, p_3, p_2)$$

$$m_{1\leftrightarrow4} = query\_metric(metric, p_1, p_4)$$

$$m_{3\leftrightarrow4} = query\_metric(metric, p_3, p_4)$$

We then compute the average of the proportional estimates based on euclidean distance, where n is the number of path segments $m_{i \leftrightarrow j}$ with available data:

$$m_{1 \rightarrow 2} = \frac{1}{n} \sum_{k=1}^{n} m_{i \leftrightarrow j} \frac{dist(p_i, p_j)}{dist(p_1, p_2)}$$

Note that $query\_metric$ returns data from recorded metrics from existing runs, and the euclidean distance propotion is used to scale existing metrics taking into account the difference between two points in the same room, e.g. the path segments (p_1, p_2) and (p_3, p_2).

5 Queriable Scenario Execution

By storing the provenance data in a property graph, we can easily query scenarios and their execution. In this section, we describe the relevant queries for the observed outputs, getting and generating oracles, and clustering. Other auxiliary queries are not included here. The queries are written in Cypher[12], the query language used by Neo4j. Note that the serialization of the PROV data into Neo4j is determined by prov-db-connector[13], the library we use in our implementation. Therefore the syntax shown in the queries in this section reflects some of the design choices of the library (e.g. the property `meta:identifier_original`). In future work, we plan to write our own Neo4j adapter to simplify the syntax for our intended use case.

5.1 Observed outputs

The first step is to get the execution data (`$runs`) of a particular scenario. Listing 37 shows the Cypher query used to do this.

```
MATCH (s)
WHERE s.`prov:type` = "exsce:ConcreteScenario" AND s.`meta:identifier_original`= $scenario_id
MATCH (a:Activity)
WHERE a.`prov:type` = "exsce:run"
MATCH (s)<-[:used]-(a)
RETURN a.`meta:identifier_original` AS run_id
```

Listing 37: Given a $scenario_id, return all the runs associated with this scenario.

Remember that we store execution data for the overall run and for each path segment. Listing **??** gets the result for a particular robot for the overall run, while Listing 39 does so for a particular path segment. The former queries a single `$run_id`, while the latter can also query metrics from a list of runs(for aggregation).

```
MATCH (n)
WHERE n.`prov:type` = "exsce:output_metric" AND n.`run:metricType` = $metric_type
WHERE $run_id CONTAINS n.`exsce:run` AND n.`meta:identifier_original` CONTAINS $robot
MATCH (n)-[g:wasGeneratedBy]->(a:Activity)-[:used]->(s)
WHERE s.`prov:type` = "exsce:ConcreteScenario"
RETURN n.`prov:value` AS value
```

Listing 38: Get the output metrics of type $metric_type for a $run_id and a particular $robot.

To get the observed output of a path segment (Listing 39), and given a list of `$runs`, get all the output metrics of type `$metric_type`. For each of the runs, find the start and end pose for the `exsce:action` that generated the output metric. Return the average and standard deviation of all the output metrics for each pair of poses.

```
UNWIND $runs AS run_id
MATCH (n WHERE n.`prov:type` = "exsce:output_metric" AND n.`run:metricType` = $metric_type)
WHERE run_id CONTAINS n.`exsce:run`
MATCH (n)-[g:wasGeneratedBy]->(a:Activity)
WHERE a.`prov:type` = "exsce:action"
MATCH (a) -[s:wasStartedBy]-> (p1)
WHERE p1.`prov:type` = "geom:Pose" AND p1.`meta:identifier_original` = $pose_id_a
MATCH (a) -[e:wasEndedBy]-> (p2)
WHERE p2.`prov:type` = "geom:Pose" AND p2.`meta:identifier_original` = $pose_id_b
WITH n.`prov:value` AS value
RETURN avg(value) AS average, stDevP(value) AS std_dev
```

Listing 39: Query a $metric_type for a path segment in a list of $runs

A special case for aggregation is the `waypoints_visited` metric, where we want to be able to query a list of runs, but which is not stored in individual path segments. Following a similar pattern as the queries above, this query (Listing 40) checks for the `waypoints_visited` output metric that matches a specific `$scenario_id`. The

[12] https://neo4j.com/docs/getting-started/cypher-intro/
[13] https://prov-db-connector.readthedocs.io/

`$metric_id` variable matches the robot ID, e.g. `tiago1:waypoints_visited`. The query returns the average of waypoints visited by a single robot.

```
UNWIND $runs AS run_id
MATCH (n:Entity)
WHERE n.`prov:type` = "exsce:output_metric" AND n.`run:metricType` = "waypoints_visited"
WHERE run_id CONTAINS n.`exsce:run`
MATCH (n)-[:wasGeneratedBy*]->(a:Activity)-[:used]->(s)
WHERE s.`prov:type` = "exsce:ConcreteScenario"
MATCH (s WHERE s.`meta:identifier_original` = $scenario_id)
MATCH (n WHERE n.`meta:identifier_original`= $metric_id)
WITH n.`prov:value` AS value
RETURN avg(value) AS average
```

Listing 40: Query waypoints_visited by a $robotfor a list of $runs

Finally, Listing 41 shows how to query the `mission_duration` metric. Given a list of run IDs `$runs`, get all the output metrics of type `mission_duration` that match each run ID, and return the average value and standard deviation of all the matching output metrics.

```
UNWIND $runs AS run_id
MATCH (n WHERE n.`prov:type` = "exsce:output_metric" AND n.`run:metricType` = "mission_duration")
WHERE run_id CONTAINS n.`exsce:run`
MATCH (n)-[g:wasGeneratedBy]->(a:Activity)-[:used]->(s)
WHERE s.`prov:type` = "exsce:ConcreteScenario" AND s.`meta:identifier_original` CONTAINS
    $scenario_id
WITH n.`prov:value` AS value
RETURN avg(value) AS average, stDevP(value) AS std_dev
```

Listing 41: Query mission_duration for a list of $runs

5.2 Oracles

5.2.1 Get oracle config

Listing 42 shows how to get the oracles for a `$scenario_id` which is a *usage* relationship containing the baseline, type of output relation, the metric that it's for, as well as the `ros:Topics` used as data sources.

```
MATCH (n WHERE n.`prov:type` = "exsce:ConcreteScenario")
MATCH (n)<-[u:used]-(o)
WHERE o.`prov:type`="exsce:oracle" AND o.`meta:identifier_original` CONTAINS $scenario_id
MATCH (o)-[v:used]->(t)
WHERE t.`prov:type`="ros:Topics"
WITH o.`oracle:robot` AS robot, u.`oracle:limit` AS oracle_limit, u.`oracle:value` AS value, u.`
    oracle:package` AS package, u.`oracle:relationship` AS rel_name, u.`oracle:tolerance` AS
    tolerance, o.`meta:identifier_original` AS oracle_id,t.`meta:identifier_original` AS topic, o.`
    oracle:metric` AS metric, n.`meta:identifier_original` AS scenario, u.`oracle:delta` AS delta
RETURN  scenario, oracle_id, oracle_limit, value, package, rel_name, tolerance, topic, robot, metric
    , delta
```

Listing 42: Cypher query to get the baseline oracle

5.2.2 Update baseline

For a baseline oracle of `$scenario_id` for a specific `$metric` and `$robot`, set the baseline to `$value` and `$tolerance`.

```
MATCH (n WHERE n.`prov:type` = "exsce:ConcreteScenario")
MATCH (n)<-[u:used]-(o)
WHERE o.`prov:type`="exsce:oracle" AND o.`meta:identifier_original` CONTAINS $scenario_id
WHERE u.`oracle:type` = "baseline" AND o.`oracle:metric` = $metric AND o.`oracle:robot` = $robot
SET u += $baseline
```

Listing 43: Cypher query to update the base values of the baseline oracle

5.3 Clustering

The queries described here enable us to create datasets to apply clustering algorithms, e.g., using Scipy's hierarchical clustering[14] or the sklearn[15] libraries.

In addition to the observed outputs, we consider the following structural properties of the scenarios to compute the distances between scenarios required for linkage. The metrics discussed here are not an exhaustive list, but rather exemplify how to use existing data in the PROV database. The evaluation of these similarity metrics for the scenario distance in the clusters will be studied as part of WP8.

5.3.1 Robot similarity

To measure how similar the robots in two scenarios are, we look at the properties of the robots in the scenario. The similarity metric takes into account the difference in the number of robots, and, for this particular case study, two differences in hardware relevant for the navigation tasks, namely the base type and wether a robot has a torso or if it just has is a mobile base. We normalize these metrics by the maximum number of robots in all the scenarios stored in the PROV database.

Let us assume r_1 and r_2 are the sets of robots for the two scenarios we want to compare, s_1 and s_2.

First, we compute the difference between the number of robots of the two scenarios as the absolute value of the difference of their cardinality:

$$d_1 = |\#(r_1) - \#(r_2)|$$

Next, we compute the sum of the difference of the number of robots with the same mobile base. Using set notation, this is the cardinality of the symmetric difference for each subset of robots with each base type b:

$$d_2 = \sum_{b=0}^{b} \#(r_{1_b} \triangle r_{2_b})$$

where

$$r_{i_b} = \{r | r \text{ is a robot with base type } b \text{ in scenario } s_i\}$$

Finally, we compute the sum of the difference of the number of robots with and without a torso t. Similar to the equation above, this is the sum of the cardinalities of the symmetric difference of the subsets with and without torso:

$$d_3 = \sum_{j=0}^{j} \#(r_{1_{t=j}} \triangle r_{2_{t=j}})$$

where $r_{i_{t=0}} \subseteq r_i$ for scenario s_i and $r_{i_{t=0}}$ indicates that $r \in r_{t=0}$ where $r_{t=0}$ is the subset of robots without a torso, and $r_{t=1}$ is the subset of robots with one.

Listing 44 shows how to query the robots of a scenario, together with the necessary information about its hardware to compute the similarity based on the base type and the presence of a torso in the PAL robots.

[14] https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html
[15] https://scikit-learn.org/stable/modules/clustering.html

```
MATCH (n WHERE n.`prov:type` = "exsce:ConcreteScenario" AND n.`meta:identifier_original`=
    $scenario_id)
MATCH (n) -[:hadMember]-> (r:Entity)-[:hadMember]->(hw:Entity)
WHERE r.`prov:type` = "exsce:robot" AND hw.`prov:type` = "hardware-config"
RETURN r.`meta:identifier_original` AS robot_id, r.`robot:type` AS robot_type, hw.`pal:base_type` as
     base_type
```

Listing 44: Query the PAL robots used in a $scenario_id, the types of mobile base they use and whether they have a torso

5.3.2 Path segments

Next, we determine how similar are the navigation tasks in a scenario by looking at the path segments. We group the path segments in three groups: those that are present in both scenarios, those that begin and end in the same area, and the rest. We normalize this metric by dividing the result by two times the largest number of segments in a scenario, which represents the worst case scenario where no path segments are common to both scenarios.

$$d_4 = \sum f(x)$$

where

$$f(x) = \begin{cases} 0, & \text{if path segment x is present in } s_1 \text{ and } s_2 \\ 0.5 & \text{if path segment x starts and ends in the same area of a path segment in the other scenario} \\ 1, & \text{otherwise} \end{cases}$$

For each scenario, we use Listing 45 to get the path segments.

```
MATCH (n WHERE n.`prov:type` = "exsce:ConcreteScenario" AND n.`meta:identifier_original`=
    $scenario_id)
MATCH (n)<-[:used]-(r:Activity)
WHERE r.`prov:type` = "exsce:run"
MATCH (a) -[s:wasStartedBy]-> (p1 where p1.`prov:type` = "geom:Pose")
MATCH (a) -[e:wasEndedBy]-> (p2 where p2.`prov:type` = "geom:Pose")
WHERE a.`exsce:run` = r.`exsce:run`
RETURN p1.`meta:identifier_original` AS pose_2, p2.`meta:identifier_original` AS pose_2
```

Listing 45: Querying the pair of points for each path segment in a scenario

5.3.3 Metrics

For the metric similarity distances, we can use the queries in Section 5.1 to obtain the results for each run. Exscept for the `waypoints_visited` metric, we normalize each metric by dividing it by its largest measurement on all runs in the PROV database. The number of visited waypoints is expressed as a percentage for each robot, and averaged for each run.

For the safety metric violations we consider two additional metrics: the duration of the safety violations (e.g. how much time the robot exceeded the `max_velocity` limit) and the number of "hotspots" or places where these violations occurr in the environment.

Listing 46 shows the query required to obtain the total time a metric `$metric_id` was violated in a `$run_id`.

```
UNWIND $runs AS run_id
MATCH (n where n.`prov:type` = "exsce:output_metric" AND n.`run:metricType` = "violation_duration"
    AND n.`exsce:metric` = $metric_id)
WHERE run_id CONTAINS n.`exsce:run`
MATCH (n)-[g:wasGeneratedBy]->(a:Activity)-[:used]->(s where s.`prov:type` = "exsce:ConcreteScenario
    " AND s.`meta:identifier_original` CONTAINS $scenario_id)
```

```
WITH n.`prov:value` as value
RETURN avg(value) AS average, stDevP(value) AS std_dev
```

Listing 46: Querying the total amount of time a safety metric was violated

The similarity metric is the difference of the total duration of the safety violation for metric m, where t_{m_1} and t_{m_2} are the durations for s_1 and s_2, respectively:

$$d_5 = |t_{m_1} - t_{m_2}|$$

We can also use spatial clustering for the violations to identify "hotspots" where violations occurr in the environment. As a similarity metric, we use the difference of the number of violation hotspots between scenarios, assuming h_1 and h_2 are the sets of hotspots for s_1 and s_2, respectively:

$$d_6 = |\#(h_1) - \#(h_2)|$$

The query to obtain the number of clusters is shown in Listing 47, where `$hotspot_type` matches the metric we are interested in, e.g., `max_velocity`.

```
UNWIND $runs AS run_id
MATCH (n:Entity where n.`prov:type` = "exsce:hotspot" AND n.`hotspot:type` = $hotspot_type )
WHERE run_id CONTAINS n.`exsce:run`
MATCH (n)-[:wasGeneratedBy*]->(a:Activity)-[:used]->(s where s.`prov:type` = "exsce:ConcreteScenario
    ")
WHERE s.`meta:identifier_original` = $scenario_id
RETURN run_id, count(n) AS hotspot_qty
```

Listing 47: Querying the number of hotspots where metrics are violated

6 ExSce Management Tutorials

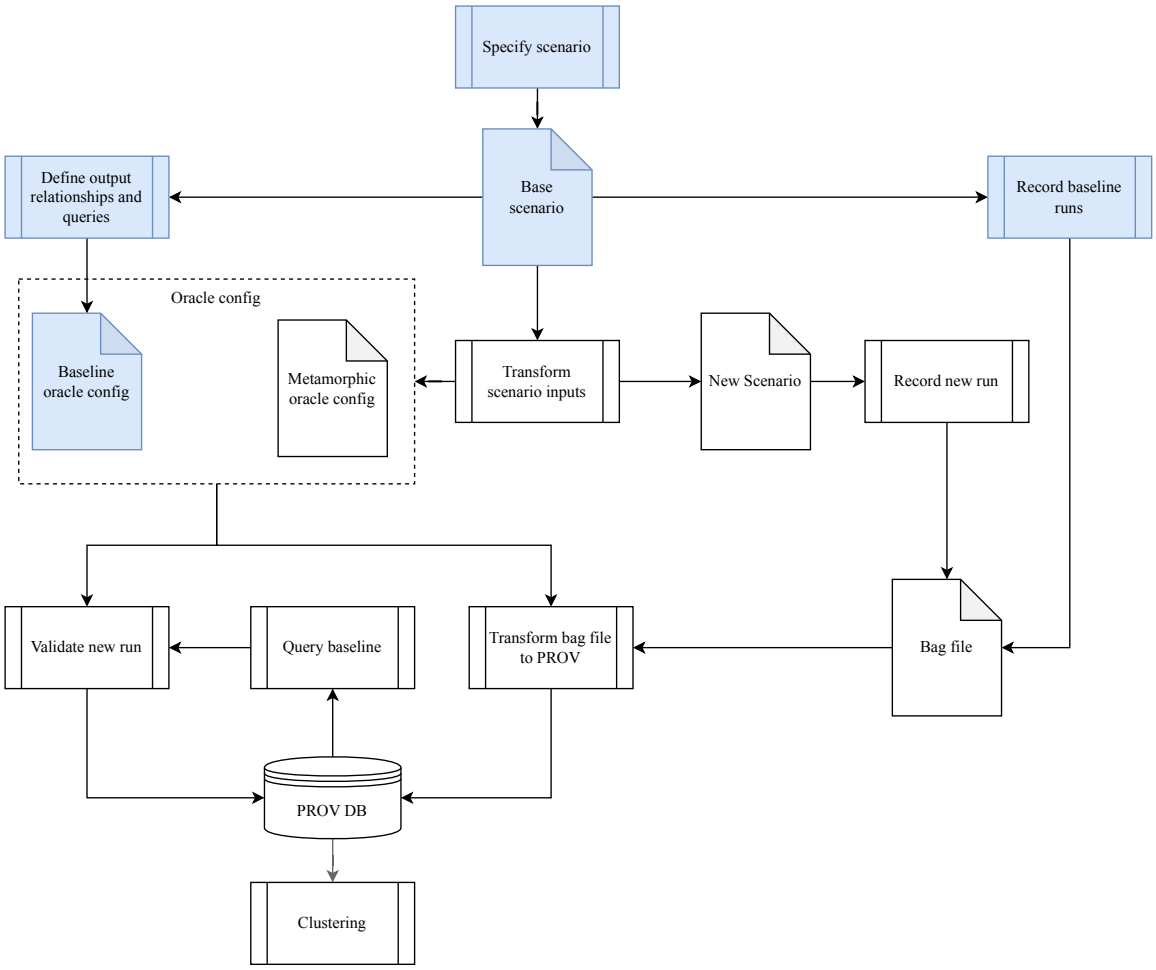

Figure 10: The Executable Scenario Management process. In blue, activities and artefacts for user-defined scenarios. In white, the activities and artefacts that are part of the metamorphic testing.

6.1 Modelling and executing a scenario to record provenance data

6.1.1 Defining a base scenario

The process, shown in Figure 10, starts by having specified a scenario to use as a base, here we use the YAML specification of scenario_c[16] for our example:

```
id: scenario_c
mission:
  id: mission_c
  type: parallel
  allocation:
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_a.yaml
    - robot: tiago2
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_b.yaml
    - robot: tiago1
```

[16] https://github.com/hbrs-sesame/models/blob/main/scenarios/scenario_c.yaml

```
      pkg: metamorphic_testing
      file_path: config/tasks/navigate_home_1.yaml
    - robot: tiago2
      pkg: metamorphic_testing
      file_path: config/tasks/navigate_home_2.yaml
environment:
  id: brsu_building_c_with_doors
  models:
    map:
      map_name: brsu_building_c_with_doors
      pkg: floorplan-DSL-environments
      relative_path: maps/
    gazebo_world:
      model_name: brsu_building_c_with_doors
      pkg: floorplan-DSL-environments
      relative_path: worlds/
  robots:
    tiago1:
      start_pose:
        id: classroom_c025-w001
        x: 44.80387496948242
        y: 37.15502166748047
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
    tiago2:
      start_pose:
        id: classroom_c025-w002
        x: 43.432926177978516
        y: 38.493873596191406
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
robots:
  - robot_id: tiago1
    robot_namespace: tiago1
    robot_type: tiago
  - robot_id: tiago2
    robot_namespace: tiago2
    robot_type: tiago
```

Listing 48: Scenario speficication for scenario_c

6.1.2 Execute a scenario and collect baseline data

Next, we execute the scenario and record n number of runs where the SUT performs their behaviour *correctly*. This requires three terminals.

1. First launching the scenario:

   ```
   roslaunch metamorphic_testing scenario_c.launch
   ```

2. Start the `record_exsce_run` script:

   ```
   rosrun metamorphic_testing record_exsce_run
   ```

3. Start the mission dispatcher:

   ```
   rosrun metamorphic_testing mission_dispatcher
   ```

The robots will proceed with the scenario execution. When the run ends, the `record_exsce_run` script will save three files, following the format `<ISO Date>_<scenario ID>_<run ID>.<ext>`, for example:

```
2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag
2023-06-13T15-48-07_scenario_c_run-e30df0b6.prov.json
2023-06-13T15-48-07_scenario_c_run-e30df0b6.rosparams
```

6.1.3 Transformation to PROV

The results of these runs and their provenance are obtained from the run artefacts and added to the Neo4j[17] database. We define the first part of the oracle config to obtain the run results and store their PROV in the database in Listing 51.

```
id: scenario_c # The ID of the scenario

# The baseline are the metrics are collected among runs of the same scenario
baseline:
  base_scenario: scenario_c # This should always be the same as the scenario ID

  # Metrics that apply to the overall mission
  mission:
    mission_duration: # Metric ID
      topics: # Data sources from which we derive the metrics
        - /tiago1/waypoint_dispatcher/feedback
        - /tiago2/waypoint_dispatcher/feedback
    distance_between_robots:
      topics:
        - /tiago1/robot_pose
        - /tiago2/robot_pose

  # Metrics that apply to individual robots
  robots:
    tiago1: # Robot ID from the scenario specification
      task_duration: # Metric name
        topics: # Data sources from which we derive the metric
        - /tiago1/waypoint_dispatcher/feedback
      waypoints_visited:
        topics:
        - /tiago1/waypoint_dispatcher/result
      distance_travelled:
        topics:
        - /tiago1/distance_travelled
      distance_to_obstacles:
        topics:
        - /tiago1/scan
      max_velocity:
        topics:
        - /tiago1/nav_vel
    tiago2:
      task_duration:
        topics:
        - /tiago2/waypoint_dispatcher/feedback
      waypoints_visited:
        topics:
        - /tiago2/waypoint_dispatcher/result
      distance_travelled:
        topics:
        - /tiago2/distance_travelled
      distance_to_obstacles:
        topics:
        - /tiago2/scan
      max_velocity:
        topics:
        - /tiago2/nav_vel
```

Listing 49: Configuration for the bag-to-PROV transformation of scenario_c

[17] https://neo4j.com/

Then we write the monitors that can process the data from the bag file and return the metric. An excerpt for the `max_velocity` metric is shown in Listing 50.

```
class RobotMetrics:
    def __init__(self, robot_id) -> None:
        self.robot_id = robot_id

        self.max_velocity = 0.0 # This must match the metric name in the oracle config

        self.goals = dict()
        self.path = ["start-pose"]
        self.path_segments = dict()

    # This must match the pattern `get_<metric name>`
    def get_max_velocity(self, msg):
        vel = math.sqrt(
            math.pow(msg.linear.x, 2)
            + math.pow(msg.linear.y, 2)
            + math.pow(msg.linear.z, 2)
        )
        if vel > self.max_velocity:
            self.max_velocity = vel
```

Listing 50: Excerpt of the robot monitors that computes the max_velocity metric

Note that the class attribute matches the metric name used in the oracle config, and the method used to compute the metric is a getter for said attribute, as the oracle and rosbag transform implementation leverages Python's dynamic imports and instantiation.

To transform the runs and store them in the PROV database, make sure you have the Neo4j database running, and use the following command:

```
cd scripts
./transform_rosbag ../runs/2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag --config ../config/oracle
    /scenario_c.yaml
```

6.2 Defining a baseline oracle and validating new executions

After we have added a sufficient number of baseline runs to our PROV database, we now update the baseline oracle config file by adding the invariant relations for the safety metrics (which are constant, as described in Section 3.3), as shown in Listing 51.

```
id: scenario_c
baseline:
  base_scenario: scenario_c
  mission:
    mission_duration:
      topics:
        - /tiago1/waypoint_dispatcher/feedback
        - /tiago2/waypoint_dispatcher/feedback
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    distance_between_robots:
      topics:
        - /tiago1/robot_pose
        - /tiago2/robot_pose
      baseline:
        limit: 0.1
      relationship:
        package: exsce.metamorphic.relationships
        name: above_min
  robots:
    tiago1:
      task_duration:
        topics:
        - /tiago1/waypoint_dispatcher/feedback
```

```
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    waypoints_visited:
      topics:
      - /tiago1/waypoint_dispatcher/result
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    distance_travelled:
      topics:
      - /tiago1/distance_travelled
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    distance_to_obstacles:
      topics:
      - /tiago1/scan
      baseline:
        limit: 0.1
      relationship:
        package: exsce.metamorphic.relationships
        name: above_min
    max_velocity:
      topics:
      - /tiago1/nav_vel
      baseline:
        limit: 1.0
      relationship:
        package: exsce.metamorphic.relationships
        name: below_max
  tiago2:
    task_duration:
      topics:
      - /tiago2/waypoint_dispatcher/feedback
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    waypoints_visited:
      topics:
      - /tiago2/waypoint_dispatcher/result
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    distance_travelled:
      topics:
      - /tiago2/distance_travelled
      relationship:
        package: exsce.metamorphic.relationships
        name: invariant
    distance_to_obstacles:
      topics:
      - /tiago2/scan
      baseline:
        limit: 0.1
      relationship:
        package: exsce.metamorphic.relationships
        name: above_min
    max_velocity:
      topics:
      - /tiago2/nav_vel
      baseline:
        limit: 1.0
      relationship:
        package: exsce.metamorphic.relationships
        name: below_max
```

Listing 51: Baseline oracle configuration for scenario_c

Finally, we can use the baseline test oracle to automatically validate new baseline runs based on their output relation:

```
cd scripts
./transform_rosbag ../runs/2023-06-13T15-48-07_scenario_c_run-e30df0b6.bag --config ../config/oracle
    /scenario_c.yaml --validate
```

This classifies the results based on whether the baseline relationship holds or not.

6.3 Generating new scenarios

6.3.1 Manually defining a new scenario

Let us manually define one new scenario based on `scenario_c.yaml`. In this minimal example, we'll be removing `tiago2` and reassigning its tasks to `tiago1`.

First we update the mission, as shown in Listing 52, by changing the allocation of `delivery_b.yaml` to `tiago1` and removing the `navigate_home_2.yaml` task for `tiago2`.

```
mission:
  id: mission_c
  type: parallel
  allocation:
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_a.yaml
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/delivery_b.yaml
    - robot: tiago1
      pkg: metamorphic_testing
      file_path: config/tasks/navigate_home_1.yaml
```

Listing 52: A new mission specification after removing tiago2 manually

We also remove `tiago2` from the list of `robots`, and its starting position from the `environment.robots` (as shown in Listing 53):

```
environment:
  robots:
    tiago1:
      start_pose:
        id: classroom_c025-w001
        x: 44.80387496948242
        y: 37.15502166748047
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
robots:
  - robot_id: tiago1
    robot_namespace: tiago1
    robot_type: tiago
```

Listing 53: Environment specification after removing tiago2 in a scenario speciication

Finally, the oracle config is updated similarly to the above, by removing topics and metrics related to `tiago2`.

6.3.2 Generating new scenarios by applying input transformations

As mentioned in Section 4, we have pre-defined some transformations. We can write a script to use these functions and recreate the example from the previous section, for example:

```
    # scenario_config is the scenario specification.
    # limits is a list of metrics that we consider constant,
    # and which should not change between the original and derived scenario
    scenario = ScenarioTransformations(scenario_config, prov_api, limits)
```

```
# Input transformation
scenario.remove_robot("tiago2", assignment={"tiago2": "tiago1"})

# Calling `generate() creates and updates the PROV documents,
# and generates a new oracle config file for the new scenario
scenario.generate()
```

Listing 54: Example of a scenario transformation script that removes tiago2 and reassigns its tasks to tiago1

The script above will create the new scenario, its oracle configuration, and save both of them in the PROV database.

7 Generalization of the ExSce Management

7.1 ExSce Workbench

The scenario specification and acceptance criteria using the Behaviour Driven Development (BDD) tools in the workbench are compatible with the ExSce Management approach and tools. As mentioned in Section 3, the only thing that is required is a transformation from the specification format to PROV, which in the case of the BDD tools, are the transformation of the instantiated scenarios into the scenario PROV and the oracle configuration. Figure 11 shows how this process for the BDD specification. The BDD fluents and constraints should be reflected in the implementation of the monitors, that is, the output metrics should be collected conforming to the BDD scenario description by (over)writing the metric methods in `RobotMetrics` and `MissionMetrics` as described in Section 6.

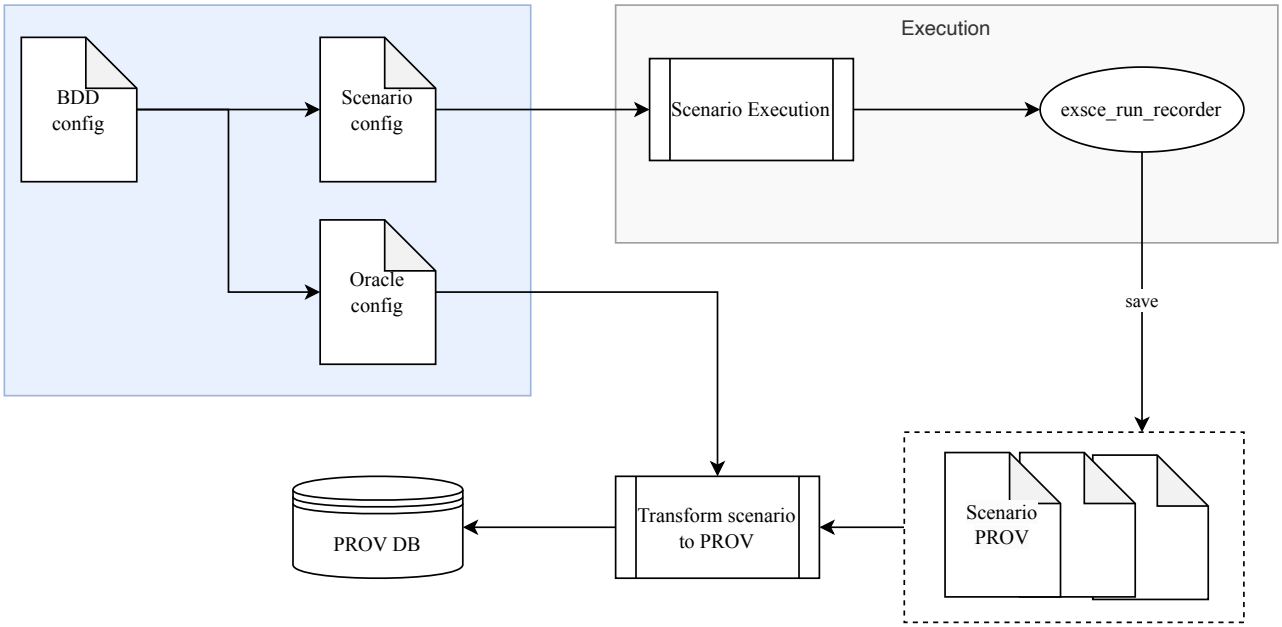

Figure 11: Transformations of the BDD tools in the ExSce Worbench (in blue) for the ExSce Management approach

The ExSce Management has been fully integrated with the Floorplan DSL. Scenarios in this deliverable use the occupancy grids and gazebo models generated from the DSL. Furthermore, the examples for `scenario_a` [18] and `scenario_b`[19] employ tasks generated by the Floorplan DSL. This integration opens the door to creating new scenarios based on environmental variations, such as the open/closed status of doors and the location of obstacles in the environment. Similarly, if available, the ExSce Management can exploit semantic information for the metamorphic relations, e.g. the estimates we make in the absence of baseline data look for poses in the same room or area.

7.2 Simulation-based testing

Fuzzing approaches presented in D6.2[20] are input transformations which could be composed into a metamorphic relation, e.g., so that we validate whether or not safety violations occurr in a simulation-based test. Fuzzed

[18] https://github.com/hbrs-sesame/models/blob/main/scenarios/scenario_a.yaml
[19] https://github.com/hbrs-sesame/models/blob/main/scenarios/scenario_b.yaml
[20] https://www.sesame-project.org/results

tests could, in general, use the safety invariant relationships to ensure that corner cases do not result in undesired or unexpected behavior. For performance metrics, a careful consideration of the effects of the fuzzing operation would be required and, most likely, both monitors and estimates would need to be adapted.

7.3 Generalizing to other use cases

As mentioned in Section 3 , abstracting the task specification allows us to reuse some of the scenario transformations to other domains or applications. For the Autonomous Pest Management use case, this means that *how* their tasks are specified (e.g., using formats of commercial software or custom specifications, such as the Plan from QGroundControl[21] or the routes that can be exported/imported into the Ground Station Software from UgCS[22], the optimization parameters for the sensor fusion drone, etc.). For example, this means that we could easily specify a multi-UAV data collection as shown below:

```
id: viti_data_collection_a
mission:
  id: viti_data_collection_mission_a
  type: parallel
  allocation:
    - robot: uav1
      pkg: viticulture_tasks
      file_path: tasks/data_collection_dkox.json
    - robot: uav2
      pkg: viticulture_tasks
      file_path: tasks/follow_data_uav_params.json
environment:
  id: dkox-vineyard-001
  robots:
    uav1:
      start_pose:
        id: home_pose-w001
        x: 11.7211
        y: 53.5011
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
    uav2:
      start_pose:
        id: home_pose-w002
        x: 12.6870
        y: 55.1296
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
robots:
  - robot_id: uav1
    robot_namespace: uav1
    robot_type: matrice_600
  - robot_id: uav2
    robot_namespace: uav2
    robot_type: matrice_210
```

Listing 55: An example of a data collection scenario with a sensor fusion drone for the Autonomous Pest Management use case

A spraying task could be specified similarly, as shown in Listing 56.

```
id: viti_spraying_a
mission:
  id: viti_spraying_mission_a
  type: parallel
  allocation:
    - robot: uav3
```

[21] https://dev.qgroundcontrol.com/master/en/file_formats/plan.html
[22] https://www.ugcs.com/

```
      pkg: viticulture_tasks
      file_path: tasks/spraying_route_a.json
    - robot: uav2
      pkg: viticulture_tasks
      file_path: tasks/follow_spraying_uav_params.json
environment:
  id: dkox-vineyard-001
  robots:
    uav3:
      start_pose:
        id: home_pose-w001
        x: 11.7211
        y: 53.5011
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
    uav2:
      start_pose:
        id: home_pose-w002
        x: 12.6870
        y: 55.1296
        z: 0.0
        roll: 0.0
        pitch: 0.0
        yaw: 0.0
robots:
  - robot_id: uav3
    robot_namespace: uav3
    robot_type: agv2
  - robot_id: uav2
    robot_namespace: uav2
    robot_type: matrice_210
```

Listing 56: An example of a spraying scenario with a sensor fusion drone for the Autonomous Pest Management use case

Note that these two scenarios are ROS-independent, and transformations, such as `replace_robot` to test different drone models would still apply. However, this still requires modifications to both adapt the methodology to non-ROS systems and to incorporate the domain specific knowledge of the task specifications, e.g. to properly invert the waypoints in a task, one must understand the semantics of the Ardupilot commands, and the specific syntax used by the software of these new SUTs.

References

[1] "PROV Model Primer." https://www.w3.org/TR/prov-primer/.

[2] A. Afzal, C. L. Goues, M. Hilton, and C. S. Timperley, "A Study on Challenges of Testing Robotic Systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, Oct. 2020, pp. 96–107. doi: 10.1109/ICST46399.2020.00020[23].

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015, doi: 10.1109/TSE.2014.2372785[24].

[4] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, "A Survey on Metamorphic Testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, Sep. 2016, doi: 10.1109/TSE.2016.2532875[25].