# A DSL for Formally Verified Interactive Robotic Applications in Service Settings

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Davide Zerla**

Student ID: 945279
Advisor: Prof. Matteo Rossi
Co-advisors: Livia Lestingi
Academic Year: 2021-22

# Abstract

Nowadays, we hear more and more of the rapid spread of robots and their application. Furthermore, creating robotic applications is a hard work that requires a heterogeneous team. Domain Specific Languages (DSLs) represent an interesting tool to support the development of robotic applications.

The objective of the thesis is to present a DSL for formally verified interactive robotic applications in service settings and present the implementation of this language in the form of a plugin. Considering a specific framework for robotic applications developed by Politecnico di Milano, the purpose of the language and its implementation is to provide the necessary tools to make the aforementioned framework more accessible and easier to use. In fact, the developed DSL is a high-level language that allows the user to write and read the scenarios in a simple and intuitive way and the plugin offers a powerful and simple interface for using the framework itself.

This document first describes the defined DSL. Then, it provides a description of the implementation of the developed language. Finally, to evaluate the correct functioning of the language implementation and its integration with the framework, some integration tests were conducted. Thanks to these tests it has been verified that the plugin actually offers the user the desired services.

**Keywords:** DSL, robotic applications, service robotics, model-driven development

# Abstract in lingua italiana

Al giorno d'oggi, sentiamo sempre di più della rapida diffusione dei robot e delle loro applicazioni. Inoltre, la creazione di applicazioni robotiche è un duro lavoro che richiede un team eterogeneo. I linguaggi specifici di dominio (Domain Specific Languages, DSLs) rappresentano uno strumento interessante per supportare lo sviluppo di applicazioni robotiche.

L'obiettivo della tesi è presentare un DSL per applicazioni robotiche interattive formalmente verificate in impostazioni di servizio e presentare l'implementazione di questo linguaggio sotto forma di plugin. Considerando un framework specifico per applicazioni robotiche sviluppato dal Politecnico di Milano, lo scopo del linguaggio e la sua implementazione è quello di fornire gli strumenti necessari per rendere il framework suddetto più accessibile e più facile da usare. Infatti il DSL sviluppato è un linguaggio di alto livello che permette all'utente di scrivere e leggere gli scenari in modo semplice ed intuitivo e il plugin offre un'interfaccia potente e semplice per l'utilizzo del framework stesso.

Questo documento descrive innanzitutto il DSL definito. Quindi, fornisce una descrizione dell'implementazione del linguaggio sviluppato. Infine, per valutare il corretto funzionamento dell'implementazione del linguaggio e la sua integrazione con il framework, sono stati condotti alcuni test di integrazione. Grazie a questi test è stato verificato che il plugin offre effettivamente all'utente i servizi desiderati.

**Parole chiave:** DSL, applicazioni robotiche, robotica di servizio, sviluppo basato sui modelli

# Contents

**Abstract** **i**

**Abstract in lingua italiana** **iii**

**Contents** **v**

**1 Introduction** **1**

1.1 Contributions of the thesis . . . 1

1.2 Overview . . . 2

**2 State of the art** **3**

**3 Background** **7**

**4 The Domain Specific Language** **11**

4.1 Domain . . . 11

4.1.1 Scenario . . . 12

4.2 Meta-model . . . 14

4.3 Concrete syntax . . . 19

4.3.1 The unit of measurement . . . 20

4.3.2 Floor planimetry . . . 20

4.3.3 Definition of robots . . . 23

4.3.4 Definition of human . . . 23

4.3.5 Mission definition . . . 23

4.3.6 Definition of user queries . . . 24

4.4 Abstract syntax . . . 24

**5 Implementation** **27**

5.1 Overview . . . 27

5.2 Grammar . . . 28

5.3 User interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.4 Syntax validation rules . . . . . . . . . . . . . . . . . . . . . . . . . . 32
5.5 Code generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5.5.1 Intersection points . . . . . . . . . . . . . . . . . . . . . . . . 34
5.5.2 Identification number . . . . . . . . . . . . . . . . . . . . . . 35
5.5.3 The type of robot . . . . . . . . . . . . . . . . . . . . . . . . 35
5.5.4 Enumeratives . . . . . . . . . . . . . . . . . . . . . . . . . . 36

**6 Evaluation** **37**
6.1 Boundary conditions . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.2 Framework experiments . . . . . . . . . . . . . . . . . . . . . . . . . 42

**7 Conclusion** **47**

**Bibliography** **49**

**A Appendix A** **53**

**List of Figures** **69**

**List of Listings** **71**

**Acknowledgements** **73**

# 1 | Introduction

Nowadays, we hear more and more of the rapid spread of robots and their application not only in industries, but also in catering, hospitals, for domestic cleaning, etc.. For this reason there are many frameworks developed for this sector.

An example is the framework developed by Politecnico di Milano [23] and described in [25–27] that allows the user of the framework to analyze scenarios about interaction between humans and service robots on a specific floor.

Furthermore, creating robotic applications often requires a team of people with different knowledge and skills. Communication between people with such heterogeneous knowledge therefore becomes a fundamental issue. Domain Specific Languages (DSLs) therefore represent an interesting tool to support the development of robotic applications. In fact, thanks to their simplicity and specificity, DSLs represent an easy to use and quick to learn tool, both for expert users and for users without technical knowledge. They also provide a common ground for heterogeneous teams.

## 1.1. Contributions of the thesis

The purpose of this thesis is to present a DSL for Formally Verified Interactive Robotic Applications in Service Settings and a plugin for the Eclipse IDE [14] that implements the developed DSL. This DSL is a high-level and user-friendly language that allows the user to specify scenarios about human-robot interactions in service settings (such as hospitals, home assistance, education). Some examples of possible scenarios are:

- the ward of a hospital where a robot carries the medicines that the various patients of the ward must take daily;
- a retirement home where a service robot accompanies the elderly to their rooms;
- a restaurant where robots accompany customers to their tables.

In particular, the DSL was created for the framework developed by Politecnico di Milano [23] and described in [25–27]. The goal of the DSL is therefore to provide a high-level language that allows the user of the framework to write and read the scenarios in a simple and intuitive way and that includes all the primitives necessary to configure the scenarios for the framework, making thus the framework more accessible even to users with non-technical knowledge. The plugin that implement the developed language, is instead a powerful and simple interface for using the framework itself. The plugin was developed using Xtext [2].

## 1.2. Overview

An overview of the thesis structure is provided below:

- Chapter 2 discusses related literature, and in particular works that introduce DSLs for robotic applications.
- Chapter 3 presents the concepts that form the background to the themes developed in the thesis. In particular, the concept of framework, of DSL and Xtext are introduced.
- Chapter 4 introduces the Domain Specific Language and the elements that led to its development. First the framework for which the DSL was produced and the domain of the DSL are introduced. Then the metamodel of the DSL is presented. Then the concrete syntax of the DSL is presented. Finally the abstract syntax of the DSL is presented.
- Chapter 5 provides a description of the implementation of the developed DSL. First the plugin is introduced. Then the model of the grammar of the implemented language is presented. Then the user interface is presented. Then the validation rules implemented for the developed language are introduced. Finally the code generator is presented.
- Chapter 6 provides a description of the evaluation of the plugin. First the integration test with scenarios in boundary conditions is presented. Then the integration test with scenarios of experiments conducted directly on the framework is introduced.
- Chapter 7 presents the considerations that represent the conclusion of the thesis.
- Appendix A shows the DSL-compliant descriptions for the most relevant scenarios tested in Chapter 6.

# 2 | State of the art

Thanks to the property of DSLs of having a syntax that is limited to their application domain, they represent a valid tool to facilitate the use of complex systems. Various works have tried to give guidelines for producing good DSLs (e.g. [20]).

The rest of this chapter describes the most relevant works introducing DSLs for robotic applications.

Bersani et al. [9] presented PuRSUE-ML, a DSL to describe the model of a robotic application in a simple and human-readable way. Thanks to this DSL they also provide a tool to make the PuRSUE framework (which allows the user to generate controllers for robotic applications) accessible even to non-expert users. PuRSUE-ML allows you to represent with a high-level language: the elements present in the environment considered, as a set of points connected to each other; the events that may occur in the environment and the actions that the agents can perform; the constraints on the order of events; the agents within the environment and what actions they can perform and what events they can react to; the initial state of the environment; the agent's mission or goals.

Detzner et al. [11] proposed a solution to make the interaction between man and robot, in warehouse logistics, simpler and therefore more efficient. As part of this solution a DSL called LoTLan is presented. This DSL allows the user to describe the material flow processes between the various departments of the warehouse. Through LoTLan it is therefore possible to represent: the logistical tasks; the elements necessary to describe a task.

García et al. [16] presented PROMISE, a DSL that allows users to define missions for multiple robots in a user-friendly way and with a high-level description. Thanks to the use of operators with which to combine the tasks of the robots, it is possible to describe articulated missions. In particular, it is possible to specify the mission both through a graphical and textual syntax.

Buch et al. [10] proposed a system to facilitate the programming of robots for small batch productions. To allow the user to easily use this system, an internal Domain-

Specific Language is also presented. The proposed DSL allows the user to define the tasks of the robot as a sequence of actions and the elements involved in the task.

Loetzsch et al. [28] presented XABSL, a DSL designed to define the behaviors and decision-making processes of autonomous agents in dynamic contexts. With XABSL it is therefore possible to represent both the agents considered and their attitudes. In particular, the latter are represented as a set of finite state machines and each behavior can then be used to compose more complex attitudes.

Götz et al. [18] presented NaoText a DSL that allows the user to express the behavior of collaborative robots in a user-friendly way with a syntax similar to object-oriented languages such as Java. In NaoText the method used to achieve this goal is to define roles for system entities. The role, which is defined within a context, determines how the robot must behave.

Finucane et al. [15] have proposed a DSL that allows the user to define the tasks of a robot, within an environment, in a human-readable and user-friendly way. In fact, the DSL allows you to write specifications using structured English. The DSL allows the user to represent: the behavior of the robot in relation to certain events; locations and elements within the environment; assumptions about the environment.

Tousignant et al. [34] proposed XRobots, a DSL designed to specify the behavior of mobile robots with a high-level description representing a hierarchical state machines. Specifically, the tasks of the robot are represented by states and the events that allow you to move from one state to another are represented by transitions. Furthermore, each task can be composed of other behaviors, thus obtaining more articulated routines.

Doherty et al. [13] presented a solution to the problem of interaction between humans and robots in collaborative multi-agent systems. As part of this solution a DSL is proposed to model missions involving multiple agents. The mission is then represented as a set of tasks composed of actions in a given context represented by constraints.

Zhang et al. [39] presented a DSL to represent the elements that make up a mobile robot in an easy and practical way. In fact, since the syntax of a DSL is limited only to the elements of the domain it must describe, it is easier to use. Given a robot, thanks to the proposed DSL it is therefore possible to represent every hardware and software part.

While reusing existing elements is an efficient practice, it is not always a good idea in the case of DSLs. In fact, a DSL is designed to fulfill a specific function. In other words, they are like tailor-made clothes built to best fit a specific person. In this way it is possible to get a more effective tool. For this reason, instead of using an existing DSL, a new one was built that best fit the considered framework presented in Section 4.1.

# 3 | Background

In this chapter the concepts that form the background to the themes developed in the thesis are presented. In particular, the concept of DSL is the first to be introduced, then Xtext is presented as a tool to implement DSL, and finally the concept of framework is introduced.

**Domain Specific Language** A Domain Specific Language (DSL) is a language, with an unambiguous grammar. This grammar is designed for a single and small application domain that's the target of the language and to which its grammar is restricted [36].

The most interesting aspect of DSLs is that thanks to their simplicity and specificity, DSLs represent an easy to use and quick to learn tool, both for expert users and for users without technical knowledge.

There are two types of DSLs: textual DSLs (e.g. [9]) where the language is displayed as text and graphical DSLs (e.g. [16]) where the language is displayed as graphic elements.

In this case a textual DSL for Formally Verified Interactive Robotic Applications in Service Settings is developed. Thanks to the use of a textual DSL it is possible to easily use the language developed in any text file such as the documentation of a project to describe the considered scenarios.

Nowadays there are many handy tools that allow you to implement a textual Domain Specific Language. These include:

- JetBrains MPS [19];
- Xtext [2];
- Racket [4];
- ANTLR [3].

To implement the developed language we will use Xtext.

**Xtext** Xtext [2] is an open-source resource that provides all the elements needed to develop a DSL quickly and easily. In fact, Xtext not only offers a parser generator and a convenient text editor, within Eclipse IDE [14], with which to specify your own language, but it also offers all the tools you need to create a specific text editor for your language in the form of plugins for the Eclipse IDE.

**Framework** A framework is a tool that offers a support structure to perform specific operations, which can be extended by adding other components on top of that structure [37].

In this case a framework for robotic applications developed by Politecnico di Milano is considered [23]. As described in [24–27] the framework is designed to provide a tool with which the user can analyze scenarios about interaction between humans and service robots on a specific floor. In particular, this operation is carried out in three phases (design-time analysis, deployment, model adjustment):

- First phase. Given the scenario proposed by the user, a model of this scenario is created that represents the starting point of the experiment and it is calculated with what probability this scenario can be successful. The framework does this verification with the Uppaal tool [22] that is a useful tool which allows the user to model, simulate and verify real-time systems, based on constraint-solving and on-the-fly techniques. So if the data obtained are acceptable it is possible to move on to the second phase.
- Second phase. Execute the model thus obtained in an environment which can be real or virtual.
- Third phase. The data collected in the second phase are used to proceed with an improvement of the model developed in the first phase in order to update the model of behavior of the human.

In particular as described in [24–27], the data representing the elements of the considered scenario are used to generate a formal model. This formal model is a Stochastic Hybrid Automata network with which it is possible to predict the variability of human behavior. Once the formal model has been obtained, this model is verified through Statistical Model Checking [8] experiments that estimates the probability of success of the considered scenario.

Figure 3.1 obtained from [24] provides an overview of the three phases of the framework.

As described in [25–27], currently the user can access the framework with commands given

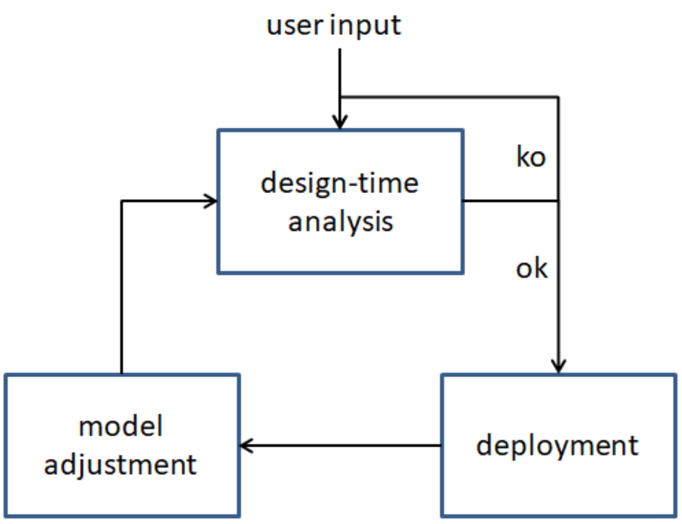


Figure 3.1: overview of the phases of the framework.

from the terminal. The DSL implementation extends the framework by providing a new way in which the user can access it more easily.

As shown in Figure 3.2 the DSL implementation is placed on top of the aforementioned framework. In this way the plugin offers a new and simple interface for using the framework.

Thanks to the plugin for the Eclipse IDE the user can write the scenario in a dsl file, then he can get the json files corresponding to the dsl file, finally he can start the framework by giving it the selected json file as input.

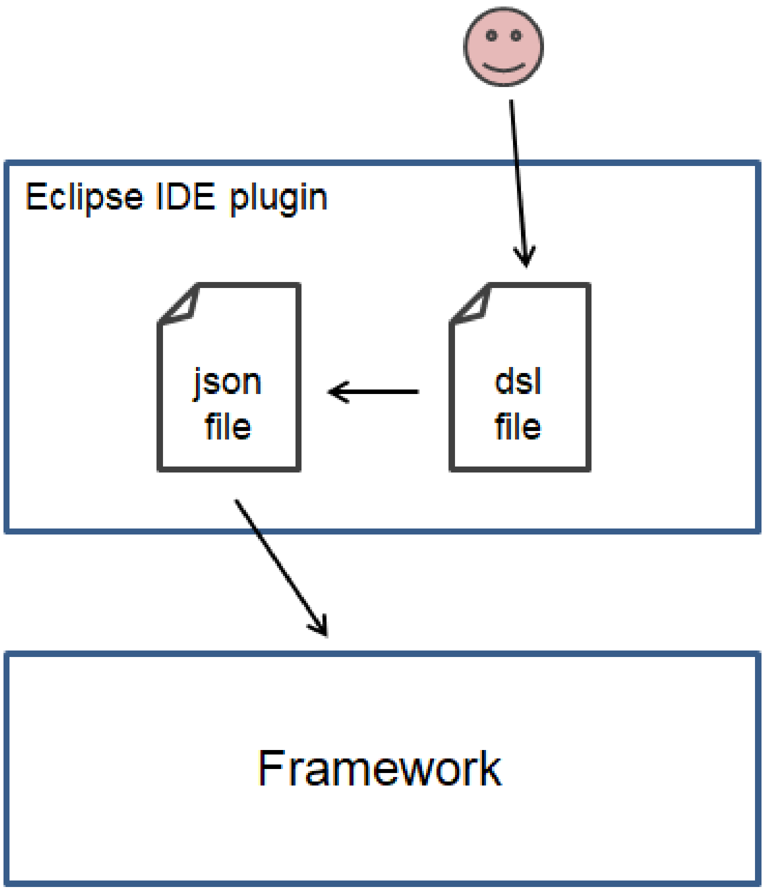


Figure 3.2: plugin workflow overview.

# 4 | The Domain Specific Language

The main contribution of this thesis is a DSL developed for the framework presented in Section 4.1. This DSL is a high-level and user-friendly language that allows the user to specify scenarios about human-robot interactions in service settings (such as hospitals, home assistance, education).

This chapter introduces the Domain Specific Language and the elements that led to its development. Section 4.1 introduces the framework for which the DSL was produced and the domain of the DSL. Section 4.2 describes how the DSL metamodel was derived. Section 4.3 describes the transition from metamodel to concrete syntax. Section 4.4 describes the transition from concrete to abstract syntax.

## 4.1. Domain

As described in [25–27] Politecnico di Milano has developed a framework [23] that allows the user to analyze scenarios about interaction between humans and robots in service settings. This means that only scenarios in non-industrial settings are considered, in which service robots (see, e.g., [32], [12]) are used.

Some examples of possible scenarios are:

- the ward of a hospital where a robot carries the medicines that the various patients of the ward must take daily;
- a retirement home where a service robot accompanies the elderly to their rooms;
- a restaurant where robots accompany customers to their tables.

Although this framework is a useful tool to support the creation of robotic applications in service settings, as described in [25–27], currently the workflow of the framework start with a formal model, quite complex, represented by a json file [1] that the framework takes as input with a command given from the terminal. Furthermore, the parameters of the formal model should be changed every time the scenario to be analyzed changes. For example if you want to change the number of humans or the type of robot used. All this makes the framework challenging to use if not outright inaccessible, especially for users who are not familiar with these formal modeling techniques or who have no knowledge in computer science.

The DSL presented in this thesis and its implementation described in Chapter 5 are proposed as a solution to these problems. The goal of the DSL is therefore to provide a high-level language that allows the user to write and read the scenarios in a simple and intuitive way and that includes all the primitives necessary to configure the scenarios for the framework, making thus the framework more accessible even to users with non-technical knowledge.

As known and indicated for example in [20, 21, 29, 35], the first point to be addressed in order to develop a Domain Specific Language is the definition of its domain from which it is then possible to specify the elements described by the language. Furthermore, the category of users for which the DSL is designed must also be defined, the type of syntax used will in fact also depend on this factor. In this case, since the DSL was created as a specific support of the aforementioned framework and therefore represents the target of the language, the domain of the Domain Specific Language can easily be defined as the same as the application domain of the framework. Consequently we can define that the domain of the developed DSL is represented by the interactive robotic applications sector in Service Settings limited to the elements necessary to configure the scenarios of the framework. Furthermore, all possible users of the framework are considered as DSL users, in particular both expert staff and staff with non-technical knowledge.

As an example of the DSL domain, the scenario used for the development of the DSL is shown below.

### 4.1.1. Scenario

On the first floor of a nursing home schematized in Figure 4.1, a service robot called Bicin is used as a support tool for the elderly on that floor.

- bringing breakfast to the guests of rooms 1, 2 and 3 directly in the room, as they are sick;
- accompanying the guests of rooms 4, 5, and 6 to the common room for breakfast, as they are healthy.

Naturally, the user of the framework is interested in obtaining the probability of success and the probability of failure, if the robot is perfectly charged.

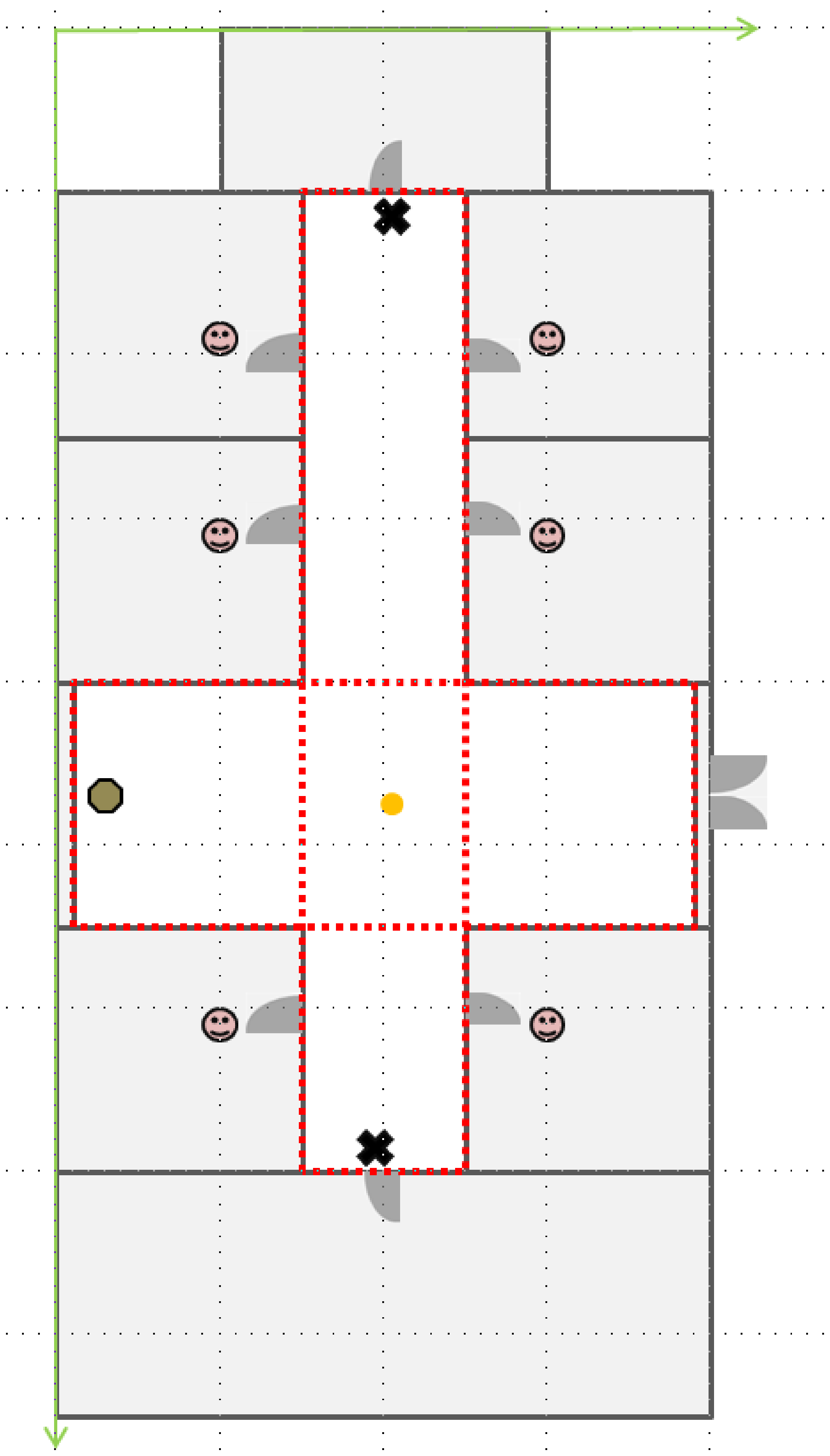


Figure 4.1: picture scenario.

## 4.2. Meta-model

The development of the Domain Specific Language presented in this chapter is driven by the metamodel. First, the metamodel that represents the input of the considered framework was defined. To derive this metamodel, the description present in [25–27] and the json file shown in Listing 4.1 corresponding to the example scenario described in Section 4.1.1 were analyzed, as it contains the most updated specifications of the framework.

The elements that describe the input of the framework obtained from [25–27] are presented below.

- A floor represented, within a two-dimensional reference system (width and length), by a series of rectangular surfaces defined by the coordinates of four vertices each. These rectangular surfaces intersect to form the floor plan.
- A set of intersection points obtained from the intersections of the rectangular surfaces mentioned above and useful for the framework to define the displacements of the entities present on the floor.
- The maximum number of intersection points that can be reached from a position with a single move, called the maximum number of neighbors.
- The set of robots present on the floor each characterized by a name, an identification number, a movement speed, an acceleration, a starting position with respect to the scenario considered, and a percentage that represents the charge level of the robot's batteries.
- The set of humans present on the floor and each requiring a service represented by one of the interaction patterns that the framework supports. Such interaction patterns are:
    1. Human-Follower, where the human follows the robot to a specific destination, for example the robot can guide the customer of a restaurant to his table;
    2. Human-Leader, where the robot follows the human to a specific destination, for example the robot can accompany an elderly person to the bathroom;
    3. Human-Recipient, where the man asks the robot to bring him an object, for example the robot can bring medicines to a patient forced to stay in bed;
    4. Human-Rescuer, where the robot requires human assistance for a specific action, for example, the robot may need a door to be opened in order to pass

through;

5. Human-Competitor, where humans and robots compete for the use of a specific resource, for example a robot and a nurse who must use bandages;
6. Human-Assistant, where the human requires the assistance of the robot for a specific action, for example an elderly person who needs a door to be opened.

The robot's mission is therefore to serve all the humans in the group by performing the pattern required by each. Each human is characterized by a name, an identification number, a speed of movement, a speed of execution of actions represented by a value that indicates his dexterity, an interaction pattern, a starting position and a destination with respect to the scenario considered, a fatigue profile determined by age and health condition, a profile of freedom of action determined by the propensity of man to do what he wants.

- The set of queries requested by the user for the selected scenario.

Listing 4.1 shows the json file corresponding to the example scenario described in Section 4.1.1 and which represents the current input of the framework.

From the analysis of the elements discussed above, the metamodel of the framework input was then obtained, shown in Figure 4.2.

Once the language domain and the framework input metamodel have been established, it is then possible to define the elements that the DSL must represent and derive a metamodel of the DSL. This metamodel is shown in Figure 4.3.

As you can see from the Figure 4.3, in order for the DSL to easily adhere to the framework, which facilitates the integration of a DSL implementation with the framework, the DSL metamodel is kept as close as possible to the framework metamodel. Furthermore, in the definition of the DSL metamodel, the fewest possible abstractions are introduced in order to keep the DSL as simple as possible, as suggested in [20]. In fact, as described in [36] and [31], one advantage of DSLs consists in their simplicity and in the small size of their application domain.

The variations to the DSL metamodel versus the framework metamodel are presented below.

**The introduction of a unit of measurement for length.** The unit of measurement of length is necessary as each scenario is described within a two-dimensional reference system that requires the user to define the coordinates of the points represented and which allows the user to understand the real physical distance between these points.

```
{
 "queries": [
        { "type": "pscs", "tau": 120, "n": 1 },
        { "type": "pfail", "tau": 120, "n": 1 }],
 "humans": [
        { "name": "guest_1", "h_id": 1, "v": 0.3, "ptrn": "FOLLOWER", "
           p_f": "e/h", "p_fw": "d", "start": [3.2, 10.0], "dest": [4.0,
            0.14999962], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "guest_2", "h_id": 2, "v": 0.3, "ptrn": "RECIPIENT", "
           p_f": "e/s", "p_fw": "d", "start": [4.8, 10.0], "dest": [4.0,
            11.85], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "guest_3", "h_id": 3, "v": 0.3, "ptrn": "FOLLOWER", "
           p_f": "e/h", "p_fw": "d", "start": [3.2, 8.0], "dest": [4.0,
           0.14999962], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "guest_4", "h_id": 4, "v": 0.3, "ptrn": "RECIPIENT", "
           p_f": "e/s", "p_fw": "d", "start": [4.8, 8.0], "dest": [4.0,
           11.85], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "guest_5", "h_id": 5, "v": 0.3, "ptrn": "RECIPIENT", "
           p_f": "e/s", "p_fw": "d", "start": [3.2, 2.0], "dest": [4.0,
           11.85], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "guest_6", "h_id": 6, "v": 0.3, "ptrn": "FOLLOWER", "
           p_f": "e/h", "p_fw": "d", "start": [4.8, 2.0], "dest": [4.0,
           0.14999962], "dext": -1, "same_as": -1, "path": -1 }],
 "robots": [
        { "name": "Bicin", "r_id": 1, "v": 0.6, "a": 0.6, "start":
           [0.65, 4.5], "chg": 100.0 }],
 "areas": [
        { "p1": [3.0, 0.0], "p2": [3.0, 12.0], "p3": [5.0, 12.0], "p4":
           [5.0, 0.0] },
        { "p1": [0.3, 3.0], "p2": [0.3, 6.0], "p3": [7.7, 6.0], "p4":
           [7.7, 3.0] }],
 "intersect": [
        { "p": [4.0, 4.5] }],
 "max_neigh": 1
}
```

**Listing 4.1:** scenario json file

**The introduction of the possibility of specifying multiple scenarios.** To increase the effectiveness of the DSL it allows the user to describe multiple scenarios together, unlike the current input of the framework which allows the specification of a single scenario at a time corresponding to a robot mission. Instead, with the developed language it is not only possible to describe a scenario for each floor, but it is also possible to specify multiple missions for the same floor. This makes it easier to manage complex scenarios such as the daily tasks of robots distributed on the floors of a building.

**The reduction of the number of vertices of the rectangular surfaces.** The number of vertices used is changed from four to two as more than enough to represent a rectangle, while the grammar of the language is thus simplified, making the writing of a scenario

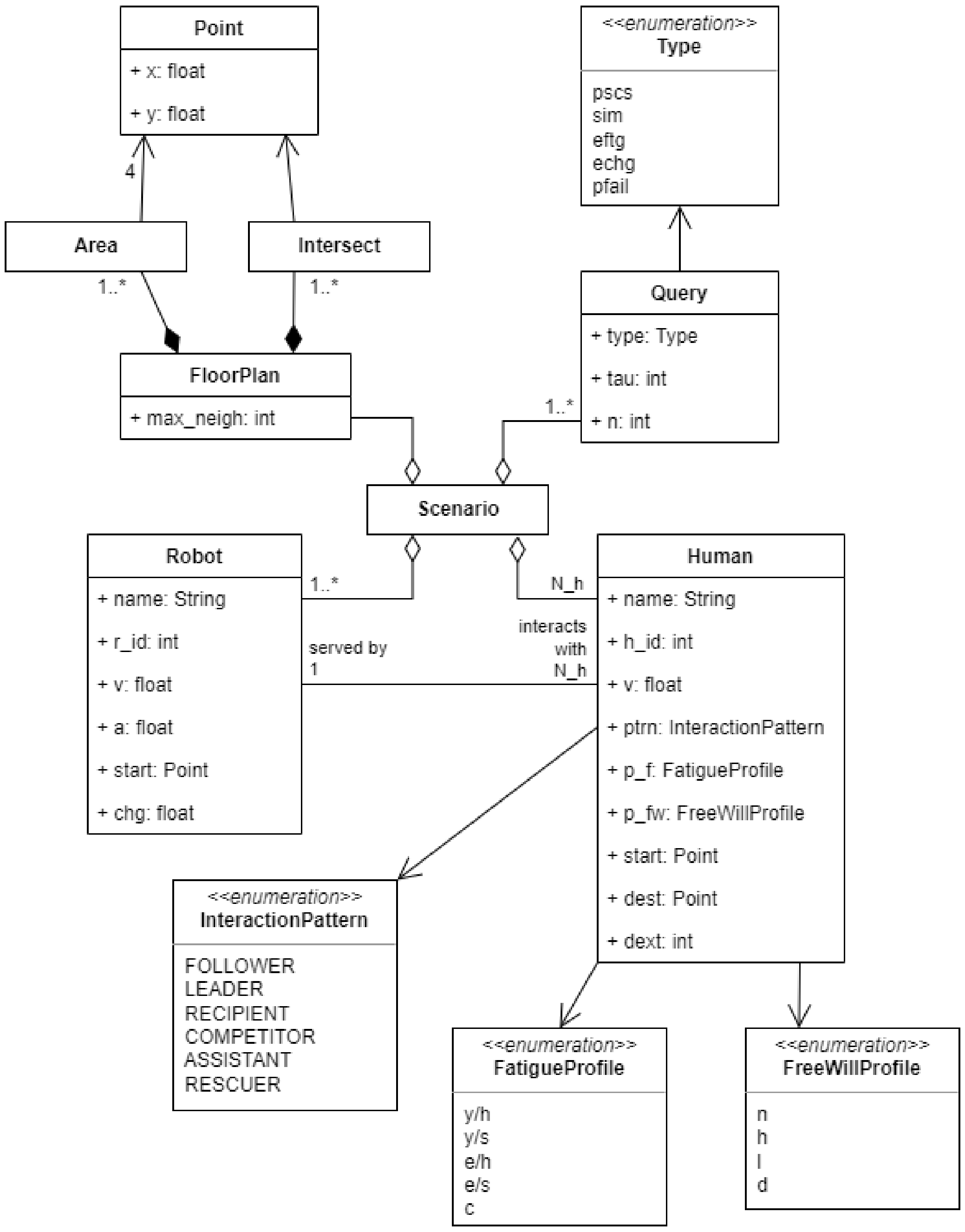


Figure 4.2: UML input framework.

easier and faster without losing anything in terms of understanding of what is described in a scenario with the DSL.

**The use of points of interest.** Inspired by [9], the use of points of interest is introduced which represent an abstraction of the position of objects or places necessary to express the services requested by humans in a clearer and more intuitive way.

**The introduction of an abstraction for the robot mission, as already done for**

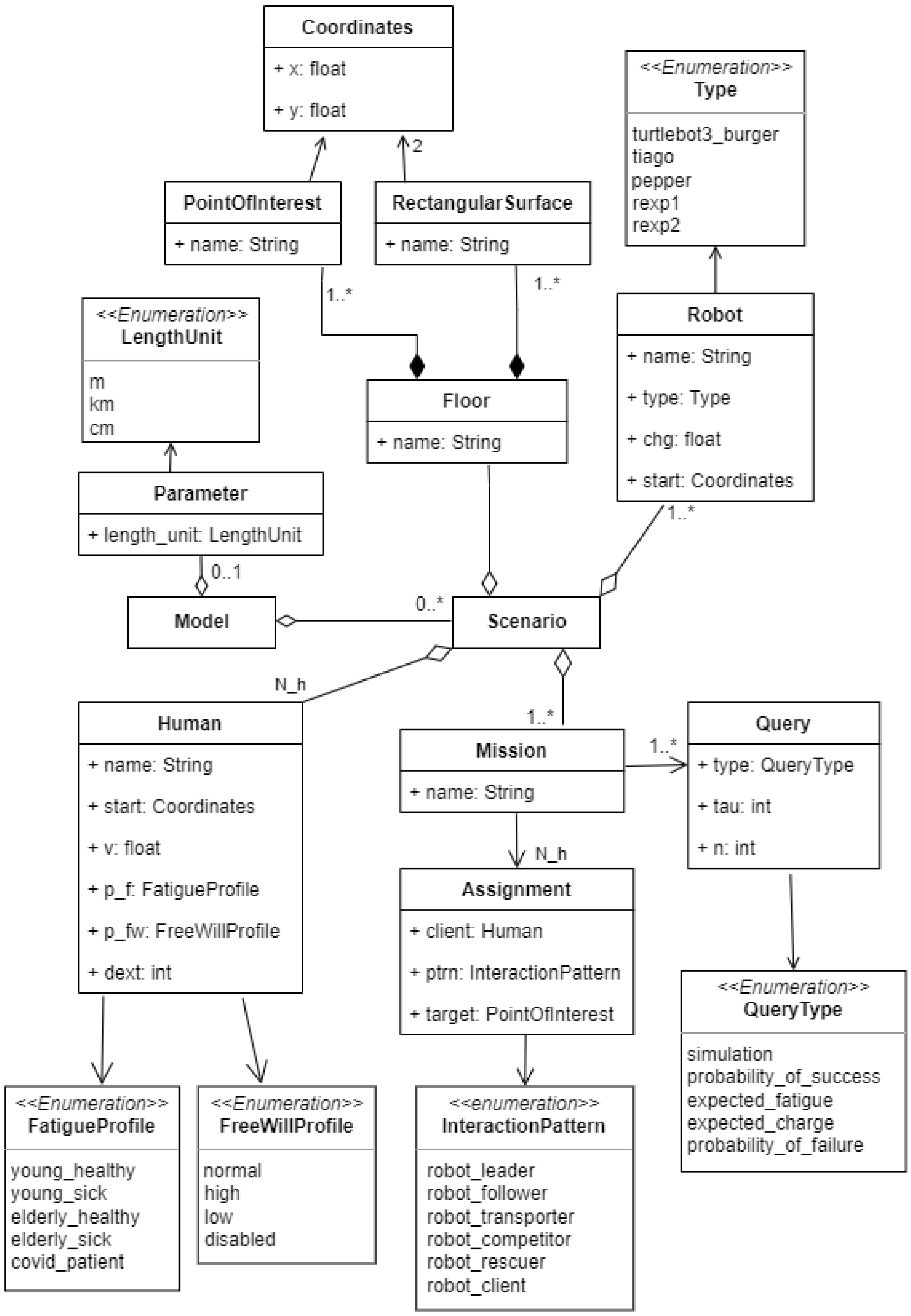


Figure 4.3: UML DSL elements.

**example in [16].** While in [25] the mission of the robot, that is the execution of all the assignments that the group of humans requires, is represented in the scenario only indirectly, in the developed DSL the mission is represented directly and the assignment is specified as a tuple represented from three elements: the interaction pattern, the human that requires the service and the object or place involved. This makes the scenario easier to understand.

**The elimination of the set of intersection points and the maximum number of neighbors.** In order to simplify the grammar of the developed language, thus making it easier to write a scenario with the DSL, the specification of the set of intersection points and the maximum number of neighbors is removed, as they are not strictly necessary for understanding a scenario, but used for technical reasons by the framework. Furthermore, this is done because, once the rectangular surfaces that define the planimetry of a floor have been specified, it is possible to automatically obtain the intersection points and the maximum number of neighbors in the implementation of the DSL.

**The elimination of the id attribute of humans and robots.** Again for the simplification of the grammar of the developed language, thus making the DSL more efficient, the id attribute of humans and robots is removed. Furthermore, this is done as it is possible to automatically obtain the id attribute of humans and robots in the implementation of the DSL. As defined in the framework [25], the id attribute of humans and robots is a simple numbering from 1 to n in the order in which humans and robots are presented.

**The insertion of a type of robot.** Instead of specifying the speed and acceleration of the robot, the type of robot used is introduced with the type attribute. This makes it possible to simplify the understanding of the scenario as all the technical characteristics dependent on the robot model used are represented by the type attribute. In the implementation of the DSL it is therefore possible to derive the speed and acceleration of the robot automatically from the type attribute. Regarding the type of robots currently considered, the DSL includes TurtleBot3 Burger [33] as `turtlebot3_burger`, TIAGo [30] as `tiago`, Pepper [38] as `pepper` and two dummy robots, called RExp1 as `rexp1` and RExp2 as `rexp2`.

**Modification of the enums used.** To make the DSL more understandable, the framework input enums are replaced with more human-readable terms.

## 4.3. Concrete syntax

Once the Domain Specific Language metamodel has been established, it is therefore possible to define the concrete syntax of the language. In this case the DSL syntax is a textual syntax. It is designed to make the described scenarios easily understandable even to non-expert users and to make it easier to write such scenarios even for users without computer knowledge. For example, all the keywords of the syntax are written in full without abbreviations to make their understanding immediate. Furthermore, the use of elements such as the ";" after each line or the curly brackets to indicate a block (typical instead of languages such as JAVA) which would make the writing of a scenario more laborious and above all more prone to errors, especially for users without computer knowledge.

Listing 4.2 presents an example of the concrete syntax of the DSL corresponding to the example scenario described in Section 4.1.1.

A description of the concrete syntax of the language is provided below.

### 4.3.1. The unit of measurement

Each scenario of the Domain Specific Language takes place within a floor that is projected on a two-dimensional reference system, together with all the elements of the floor. The position of each element, therefore, is indicated through the use of coordinates, respectively the x and y coordinate typical of a Cartesian plane, each expressed with a float. Furthermore, the speed of humans and robots is also expressed with a float. Consequently, the precision with which positions and speed are indicated must be consistent, but to leave the choice of precision to be used by the user, a specific command is then introduced to indicate the unit of measurement of length. Explaining the unit of measurement of length also allows the user to have a better perception of the real physical distances between the elements of the scenario. The unit of measurement of length is indicated with the command:

```
parameter length_unit m
```

The keyword "`parameter`" introduces a parameter that will be valid for the whole document (as a sort of global variable), in this case "`length_unit`" which expresses the unit of measurement of length. It is an enum value, so it is possible to set the unit of measurement of the length in meters "`m`", centimeters "`cm`" and kilometers "`km`". If the parameter is not specified by default it is considered in meters as it allows the user to obtain the best precision for the scenarios considered, that is 9999 meters with an accuracy in millimeters.

### 4.3.2. Floor planimetry

Each scenario considered takes place above a specific floor. It is possible to specify a floor and its name with the command:

```
build floor floor_name:
```

```
// scenario
// the unit of measure of the length is the meter
parameter length_unit m
// description of the first floor
build floor first:
// planimetry
add rectangular_surface corridor with coordinates_vertex_A (3.0;2.0)
                                      coordinates_vertex_C (5.0;14.0)
add rectangular_surface hall     with coordinates_vertex_A (0.30;8.0)
                                      coordinates_vertex_C (7.70;11.0)
// points of interest
add point_of_interest breakfast_tray       with coordinates (4.0;2.15)
add point_of_interest common_room_entrance with coordinates (4.0;13.85)
// description of the robot
define robots:
robot Bicin with coordinates (0.65;9.5) type rexp1
            charge_percentage 100.0
// description of the humans
define humans:
human guest_1 with coordinates (3.20;4.0)  speed 0.3 dexterity 0
            fatigue_profile elderly_healthy free_will_profile disabled
human guest_2 with coordinates (4.80;4.0)  speed 0.3 dexterity 0
            fatigue_profile elderly_sick    free_will_profile disabled
human guest_3 with coordinates (3.20;6.0)  speed 0.3 dexterity 0
            fatigue_profile elderly_healthy free_will_profile disabled
human guest_4 with coordinates (4.80;6.0)  speed 0.3 dexterity 0
            fatigue_profile elderly_sick    free_will_profile disabled
human guest_5 with coordinates (3.20;12.0) speed 0.3 dexterity 0
            fatigue_profile elderly_sick    free_will_profile disabled
human guest_6 with coordinates (4.80;12.0) speed 0.3 dexterity 0
            fatigue_profile elderly_healthy free_will_profile disabled
// mission of the robot
define mission breakfast:
do robot_transporter for guest_2 with target breakfast_tray
do robot_transporter for guest_4 with target breakfast_tray
do robot_transporter for guest_5 with target breakfast_tray
do robot_leader for guest_1 with target common_room_entrance
do robot_leader for guest_3 with target common_room_entrance
do robot_leader for guest_6 with target common_room_entrance
// queries of the user
define queries of mission breakfast:
compute probability_of_success with duration 120 runs 1
compute probability_of_failure with duration 120 runs 1
```

**Listing 4.2:** scenario DSL file

This command introduces a block in which it is possible to define the floor plan, where `"floor_name"` represents the name of the floor chosen by the user. The planimetry of each floor is specified through the use of rectangular surfaces that overlap at specific intersection points. These rectangular surfaces are identified by using the coordinates of two vertices on a diagonal, respectively the vertex A and the vertex C. By convention, vertex A is considered the top left one and the others are defined clockwise. However, it is also possible for the user to use another convention as long as the vertices belong to the same diagonal of the rectangle. The command with which it is possible to define a rectangular surface and its coordinates is:

```
add rectangular_surface name with coordinates_vertex_A (float; float)
                                 coordinates_vertex_C (float; float)
```

An example of how to use this command is shown in the Listing 4.2 at lines 7-10.

After having specified the rectangular surfaces that represent the floor, it is possible to specify the points of interest of the floor with the command:

```
add point_of_interest name with coordinates (float; float)
```

An example of how to use this command is shown in the Listing 4.2 at lines 12-13.

This command specifies a point of interest called `"name"` and with coordinates, within the floor, `"(float; float)"`. With `"point_of_interest"` we mean a place or an object that are the target of one of the services requested by the human. The position of this point actually represents an abstraction independent of the real physical position of the point. It is therefore up to the user whether to make the `"point_of_interest"` coincide with the barycentre of an object, or with another position from which it is easy to access the reference object or use another convention. It is also possible to specify multiple floors within the same file as in the example below:

```
build floor first:
add rectangular_surface corridor with coordinates_vertex_A (3.0;2.0)
                                    coordinates_vertex_C (5.0;14.0)
add rectangular_surface hall     with coordinates_vertex_A (0.30;8.0)
                                    coordinates_vertex_C (7.70;11.0)
[...]
build floor second:
add rectangular_surface corridor with coordinates_vertex_A (3.0;2.0)
                                    coordinates_vertex_C (4.0;12.0)
add rectangular_surface hall     with coordinates_vertex_A (1.0;8.0)
                                    coordinates_vertex_C (7.0;11.0)
[...]
```

### 4.3.3. Definition of robots

Since more robots can be assigned to a specific floor, it is possible to define a set of robots with the command:

```
define robots:
```

For each robot, a name is defined, a starting position within the scenario, a type related to the robot model from which any technical specifications are derived and a percentage of the battery charge. The command with which it is possible to define the characteristics of a robot is:

```
robot name with coordinates (float; float)
                type robot_type charge_percentage float
```

An example of how to use this command is shown in the Listing 4.2 at lines 16-17.

### 4.3.4. Definition of human

Since several humans who require a service can be assigned to a specific floor, it is possible to define a set of humans with the command:

```
define humans:
```

For each human, a name is defined, a starting position within the scenario, a movement speed `"speed"`, a speed of execution of the actions `"dexterity"`, a fatigue profile, and a free will profile. The command with which it is possible to define the characteristics of a human is:

```
human name with coordinates (float; float) speed float dexterity int
          fatigue_profile fprofile_type free_will_profile fwprofile_type
```

An example of how to use this command is shown in the Listing 4.2 at lines 20-31.

### 4.3.5. Mission definition

It is possible to define the mission of the robot with the command:

```
define mission mission_name:
```

A name is associated with the mission and consists in the execution by the robot of the request made by each human of the group defined with the command `"define humans:"` and represented by one of the interaction patterns specified in Figure 4.3. To define one of the mission assignments the command is:

```
do pattern for human_name with target point_of_interest
```

An example of how to use this command is shown in the Listing 4.2 at lines 34-39.

For each assignments it is necessary to specify the type of interaction patterns, the name of the human involved and the point of interest target of the action. It is possible to specify multiple missions on the same level as in the example below:

```
define mission breakfast:
do robot_transporter for guest_1 with target breakfast_tray
[...]
define mission lunch:
do robot_transporter for guest_1 with target lunch_tray
[...]
```

### 4.3.6. Definition of user queries

Each mission is associated with the queries made by the user relating to the mission to which the query is associated. The command with which to indicate the block of queries is:

```
define queries of mission mission_name:
```

For each query it is possible to indicate the duration of the experiment and the number of attempts. The command to specify the query is:

```
compute query_type with duration int runs int
```

An example of how to use this command is shown in the Listing 4.2 at lines 42-43.

## 4.4. Abstract syntax

Given the concrete syntax of a Domain Specific Language, the abstract syntax can then be easily obtained. In this case, the Extended Backus-Naur Form (EBNF) [17] is used to represent the abstract syntax of the DSL, represented in Listing 4.3.

As shown in Listing 4.3 at lines 27-28, in order for grammar to guide the user's speech, various parameters such as `"QUERY_TYPE"` represent enums.

As shown in Listing 4.3 at line 13, regarding the types of service robots currently considered, the DSL includes TurtleBot3 Burger [33] as `turtlebot3_burger`, TIAGo [30] as `tiago`, Pepper [38] as `pepper` and two dummy robots, called RExp1 as `rexp1` and RExp2 as `rexp2`.

```
MODEL := (PARAMETER)? (SCENARIO)*
PARAMETER := 'parameter' 'length_unit' LENGTH_UNIT
LENGTH_UNIT := 'm' | 'km' | 'cm'
SCENARIO := FLOOR ROBOTS HUMANS (MISSION)+
FLOOR := 'build' 'floor' ID ':' (SURFACE)+  (POINT)+
SURFACE := 'add' 'rectangular_surface' ID 'with' VERTICES
VERTICES := 'coordinates_vertex_A' COORDINATES
            'coordinates_vertex_C' COORDINATES
POINT := 'add' 'point_of_interest' ID 'with' 'coordinates' COORDINATES
ROBOTS := 'define' 'robots' ':' (ROBOT)+
ROBOT := 'robot' ID 'with' 'coordinates' COORDINATES 'type' TYPE
         'charge_percentage' FLOAT
TYPE := 'turtlebot3_burger' | 'tiago' | 'pepper' | 'rexp1' | 'rexp2'
HUMANS := 'define' 'humans' ':' (HUMAN)+
HUMAN := 'human' ID 'with' 'coordinates' COORDINATES 'speed' FLOAT
         'dexterity' INT 'fatigue_profile' FATIGUE_PROFILE
         'free_will_profile' FREE_WILL_PROFILE
FATIGUE_PROFILE := 'young_healthy' | 'young_sick' | 'elderly_healthy'
                   | 'elderly_sick' | 'covid_patient'
FREE_WILL_PROFILE := 'normal' | 'high' | 'low' | 'disabled'
MISSION := 'define' 'mission' ID ':' (ASSIGNMENT)+ QUERIES
ASSIGNMENT := 'do' PATTERN 'for' ID 'with' 'target' ID
PATTERN := 'robot_leader' | 'robot_follower' | 'robot_transporter'
           | 'robot_competitor' | 'robot_rescuer' | 'robot_client'
QUERIES := 'define' 'queries' 'of' 'mission' ID ':' (QUERY)+
QUERY := 'compute' QUERY_TYPE 'with' 'duration' INT 'runs' INT
QUERY_TYPE := 'simulation' | 'probability_of_success'
    | 'expected_fatigue' | 'expected_charge' | 'probability_of_failure'
COORDINATES := '(' FLOAT ';' FLOAT ')'
FLOAT := INT DEC
DEC := '.' INT
ID := LETTER (LETTER|DIGIT)*
INT := (DIGIT)+
DIGIT := '0' | ... | '9'
LETTER := '_' | 'a' | ... | 'z' | 'A' | ... | 'Z'
```

**Listing 4.3:** EBNF of the DSL

# 5 | Implementation

As part of the contribution given by this thesis, a plugin for the Eclipse IDE [14] is presented. This plugin represents the implementation of the DSL presented in the Chapter 4. This plugin is an interface to the framework developed by Politecnico di Milano [23].

This chapter provides a description of the implementation of the developed DSL. Section 5.1 introduces the plugin. Section 5.2 presents the model of the grammar of the implemented language. Section 5.3 presents the user interface. Section 5.4 describes the validation rules implemented for the developed language. In Section 5.5 the code generator is presented.

## 5.1. Overview

Nowadays there are many handy tools that allow you to implement a textual Domain Specific Language. These include:

- JetBrains MPS [19];
- Xtext [2];
- Racket [4];
- ANTLR [3].

To implement the developed language we will use Xtext [2]. Xtext [2] is an open-source resource that provides all the elements needed to develop a DSL quickly and easily. In fact, Xtext not only offers a parser generator and a convenient text editor, within Eclipse IDE [14], with which to specify your own language, but it also offers all the tools you need to create a specific text editor for your language in the form of plugins for the Eclipse IDE. Which is precisely the goal of this thesis.

As indicated in the Chapter 1 the purpose of this thesis is to provide the tools to make the framework developed by Politecnico di Milano [23] more accessible. To achieve this, while the DSL is a useful tool for specifying the scenarios that are the input of the framework, the plugin is instead a powerful and simple interface for using the framework itself. In fact, if before the framework was usable only with commands given from the terminal, thanks to the plugin for the Eclipse IDE the user of the framework has at his disposal the comfortable and customizable (e.g. it is possible to change the settings of the syntax coloring) Eclipse IDE [14] with which he can easily write the scenarios in the developed DSL and start the experiments he wants to conduct with the framework in a simple and fast way.

The Figure 5.1 provides a simplified description of the use of the framework by the user through the plugin installed in the Eclipse IDE. Thanks to the plugin for the Eclipse IDE the user can get the json files corresponding to the considered scenario, then he can start the framework by giving it the selected json file as input.

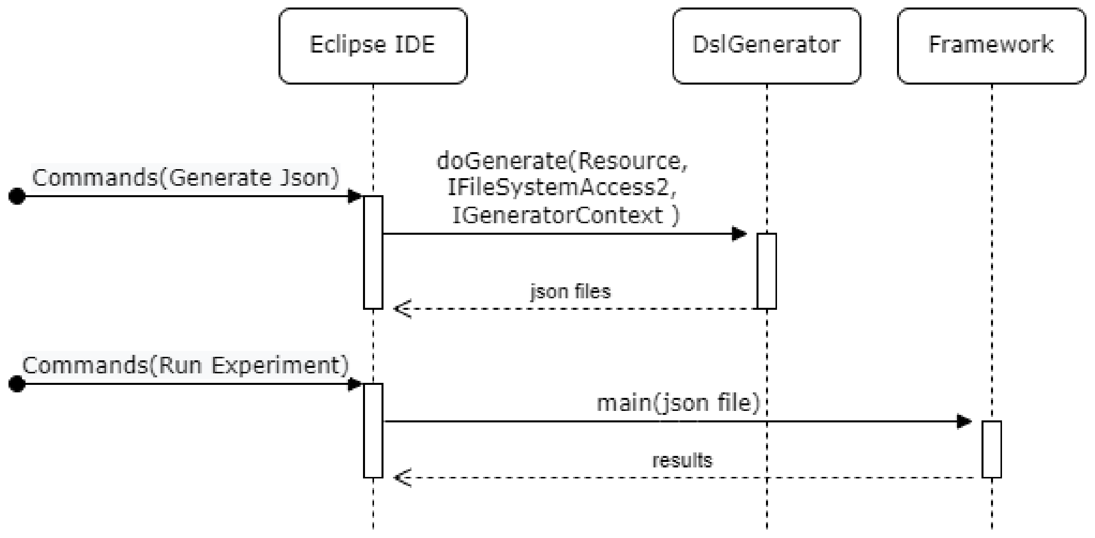


Figure 5.1: Plugin interaction diagram.

## 5.2. Grammar

As indicated in [7], the first thing to do to implement the Domain Specific Language developed through Xtext is to define the grammar of the language within a specific file with the .xtext extension. To achieve this, the language description provided in the Extended Backus-Naur Form [17] in Section 4.4 is used and adapted to the syntax of Xtext.

The resulting .xtext file is shown in Listing 5.1 and contains the definition of all elements of the DSL syntax described in Section 4.3.

```
grammar it.polimi.framework.dsl.Dsl with org.eclipse.xtext.common.Terminals
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
generate dsl "http://www.polimi.it/framework/dsl/Dsl"

Model:
	parameter=(Parameter)? scenarios+=(Scenario)*;
Parameter:
	'parameter' 'length_unit' length_unit=Length_unit;
enum Length_unit:
	METER='m'
	|KILOMETER='km'
	|CENTIMETER='cm';
Scenario:
	floor=Floor robots=Robots humans=Humans missions+=(Mission)+;
Floor:
	'build' 'floor' floor_name=ID ':' surfaces+=(Surface)+  points+=(Point)+;
Surface:
	'add' 'rectangular_surface' name=ID 'with' vertices=Vertices;
Vertices:
	'coordinates_vertex_A' vertex_A=Coordinates 'coordinates_vertex_C' vertex_C=Coordinates;
Point:
	'add' 'point_of_interest' name=ID 'with' 'coordinates' coordinates=Coordinates;
Robots:
	'define' 'robots' ':' robots+=(Robot)+;
Robot:
	'robot' name=ID 'with' 'coordinates' coordinates=Coordinates 'type' type=Type 'charge_percentage' charge_percentage=FLOAT;
enum Type:
	TURTLEBOT3_BURGER='turtlebot3_burger'
	|TIAGO='tiago'
	|PEPPER='pepper'
	|REXP1='rexp1'
	|REXP2='rexp2';
Humans:
	'define' 'humans' ':' humans+=(Human)+;
Human:
	'human' name=ID 'with' 'coordinates' coordinates=Coordinates 'speed' speed=FLOAT 'dexterity' dext=INT 'fatigue_profile' fatigue_profile=Fatigue_profile 'free_will_profile' free_will_profile=Free_will_profile;
```

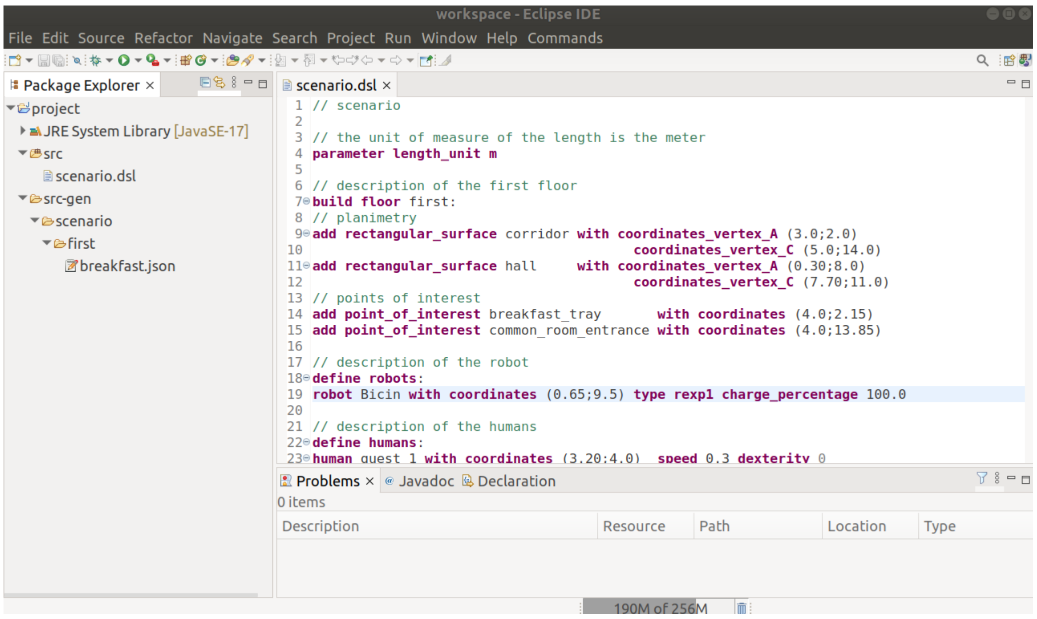
```
enum Fatigue_profile:
        YOUNG_HEALTHY='young_healthy'
        | YOUNG_SICK='young_sick'
        | ELDERLY_HEALTHY='elderly_healthy'
        | ELDERLY_SICK='elderly_sick'
        | COVID_PATIENT='covid_patient';
enum Free_will_profile:
        NORMAL='normal'
        | HIGH='high'
        | LOW='low'
        | DISABLED='disabled';
Mission:
        'define' 'mission' name=ID ':' assignments+=(Assignment)+
           queries=Queries;
Assignment:
        'do' pattern=Pattern 'for' client=ID 'with' 'target' target=ID;
enum Pattern:
        ROBOT_LEADER='robot_leader'
        | ROBOT_FOLLOWER='robot_follower'
        | ROBOT_TRANSPORTER='robot_transporter'
        | ROBOT_COMPETITOR='robot_competitor'
        | ROBOT_RESCUER='robot_rescuer'
        | ROBOT_ASSISTANT='robot_client';
Queries:
        'define' 'queries' 'of' 'mission' mission=ID ':' queries+=(Query
           )+;
Query:
        'compute' query_type=Query_type 'with' 'duration' duration=INT '
           runs' runs=INT;
enum Query_type:
        SIMULATION='simulation'
        | PROBABILITY_OF_SUCCESS='probability_of_success'
        | EXPECTED_FATIGUE='expected_fatigue'
        | EXPECTED_CHARGE='expected_charge'
        | PROBABILITY_OF_FAILURE='probability_of_failure';
Coordinates:
        '(' x=FLOAT ';' y=FLOAT ')';
terminal FLOAT returns ecore::EFloat:
        INT DEC;
terminal DEC:
        '.' INT;
```

**Listing 5.1:** Xtext model of the DSL grammar

As you can see, the generated grammar is enriched with a basic grammar already present in Xtext and called Common Terminals [5]. This grammar contains the definition of some basic primitives such as ID and INT with which it is possible to define alphanumeric identifiers and integers respectively. It also introduces the possibility to write single-line comments and multi-line comments.

## 5.3. User interface

Thanks to the plugin, as shown in Figure 5.2 the user is provided with a convenient user interface, with all the features made available by the Eclipse IDE [14]. Specifically, the plugin provides a specific text editor for the grammar described in Section 4.3. To use this text editor, the user must describe the scenario in a file with the .dsl extension.



```
// scenario

// the unit of measure of the length is the meter
parameter length_unit m

// description of the first floor
build floor first:
// planimetry
add rectangular_surface corridor with coordinates_vertex_A (3.0;2.0)
                                      coordinates_vertex_C (5.0;14.0)
add rectangular_surface hall     with coordinates_vertex_A (0.30;8.0)
                                      coordinates_vertex_C (7.70;11.0)
// points of interest
add point_of_interest breakfast_tray       with coordinates (4.0;2.15)
add point_of_interest common_room_entrance with coordinates (4.0;13.85)

// description of the robot
define robots:
robot Bicin with coordinates (0.65;9.5) type rexp1 charge_percentage 100.0

// description of the humans
define humans:
human guest_1 with coordinates (3.20;4.0)  speed 0.3 dexterity 0
```

Figure 5.2: text editor provided with the Eclipse IDE plugin.

The user interface obtained from Xtext includes many useful features [2]: content assist, quick fixes, syntax coloring, outline view. . . .

As shown in Figure 5.2 the plugin developed adds the Commands menu to the Eclipse IDE. Within this menu, the user can find two useful commands:

- Generate Json;
- Run Experiment.

Once a file with the .dsl extension has been created, thanks to the "Generate Json" command, the user can save the .dsl file he is writing and if there are no errors in this file, the json files corresponding to the saved file will be generated. Specifically, a json file is produced for each mission specified by the user, for each single floor described within the file. After selecting one of the generated json files, with the "Run Experiment" command the user can start his experiment with the framework described in Section 4.1. In fact, the "Run Experiment" command starts the framework by giving it the selected json file as input and once the framework has finished its computation, the result is reported in a console within the Eclipse IDE.

## 5.4. Syntax validation rules

To provide maximum user support to reduce errors and increase productivity, the validation present by default in Xtext [6] is then enriched with the validation rules listed below.

**checkUniqueFloorName** Within a file it is possible to specify multiple scenarios, one for each floor. However, the name of each floor must be unique. If two floors have the same name, an error is reported to the user.

**checkUniqueSurfaceName** The `rectangular_surface` name must be unique for the same floor. If two `rectangular_surface` have the same name, an error is reported to the user.

**checkVerticesValidity** The validity of the vertices of a `rectangular_surface` is checked, that is, it is checked that the vertices really represent a rectangle. In fact, to represent a `rectangular_surface` the vertices A and C cannot be overlapped or aligned. If the vertices A and C do not form the diagonal of a rectangle, an error is reported to the user.

**checkOverlappingRectangles** There cannot be two `rectangular_surface` with the same vertices. If two rectangles overlap, an error is reported to the user.

**checkUniquePointName** The `point_of_interest` name must be unique for the same floor. If two `point_of_interest` have the same name, an error is reported to the user.

**checkPointValidity** The validity of the `point_of_interest` is checked, that is, that there is actually a point with the coordinates indicated within the scenario. The coordinates of the `point_of_interest` must be into a `rectangular_surface`. If this is not the case, an error is reported to the user.

**checkUniqueRobotName** The robot name must be unique for the same floor. If several robots on the same floor have the same name, an error is reported to the user.

**checkUniqueRobotHumanPosition** The starting coordinates of humans and robots must be different. If the initial position between humans and robots is superimposed, an error is reported to the user.

**checkChargePercentage** The robot `charge_percentage` must be a value between 0 and 100. If `charge_percentage` does not represent a percentage, an error is reported to the user. In addition, the user can use a quick fixes to set the maximum percentage value to 100.

**checkRobotPositionValidity** The validity of the initial coordinates of a robot is checked, i.e. the coordinates of the robot must be into a `rectangular_surface`. If the initial position of the robot does not correspond to a point on the floor, an error is reported to the user.

**checkUniqueHumanName** The human name must be unique for the same floor. If there are more humans with the same name on the same floor, an error is reported to the user.

**checkHumanPositionValidity** The validity of the initial coordinates of a human is checked, that is, the coordinates of the human must be into a `rectangular_surface`. If the initial position of the human does not correspond to a point on the floor, an error is reported to the user.

**checkUniqueMissionName** The mission name must be unique for the same floor. If there are multiple missions with the same name, an error is reported to the user.

**checkAllHumansAreServed** The mission must contain an assignment for each human on the floor. If an assignment for each human is not specified, an error is reported to the user.

**checkExistingHuman** Humans for which an assignment has been specified must be present in the humans block. If an assignment is specified for a human that has not been defined, an error is reported to the user.

**checkDuplicates** Within a mission, each human can specify only one assignment. If there are multiple assignments for the same human, an error is reported to the user.

**checkExistingPoint** For each assignment a target corresponding to a `point_of_interest` must be specified. If a target is used that has not been defined before, an error is reported to the user.

**checkQuery** Each mission must have its own queries. If the queries block does not report the name of the mission to which they are associated, an error is reported to the user. The user can use a quick fix to add the mission name.

**checkpoint** If a `point_of_interest` is defined and not used, a warning is reported to the user.

## 5.5. Code generator

The DslGenerator class represents the code generator [6] of the DSL implementation. Thanks to it, it is possible to generate json files corresponding to the model specified by the user with the DSL. Specifically, to increase the efficiency of the tool provided to the user, a json file is generated for each mission of each floor described in the .dsl file. In this way, the user does not have to re-enter the data of the floor plan and the elements present in it every time.

Below are presented other elements that have been introduced in the DslGenerator class to make the tool provided to the user more user-friendly.

### 5.5.1. Intersection points

Within the json file that represents the input of the framework, as presented in Section 4.2 and described in [25–27], it was necessary for the user to manually specify the intersection points of the rectangular surfaces. However the coordinates of these points can be easily calculated automatically. To reduce the user's work by making it easier to specify the scenarios, the DslGenerator enters the coordinates of the intersection points as the barycentre of the surface obtained by overlapping two rectangular areas, as shown in the Figure 5.3. Furthermore, the maximum number of intersection points is also calculated automatically.

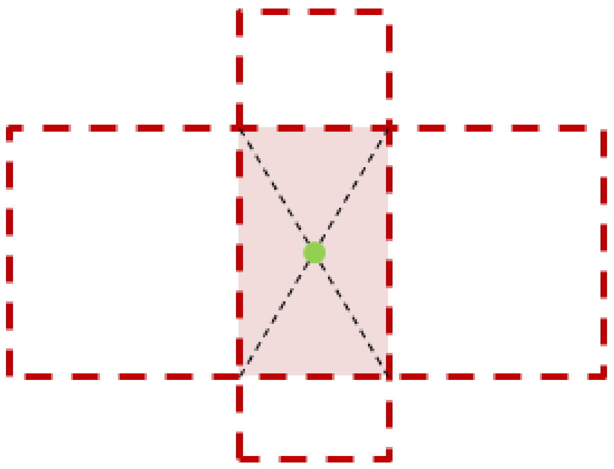


Figure 5.3: example point of intersection.

### 5.5.2. Identification number

Within the json file that represents the input of the framework, as presented in Section 4.2 and described in [25–27], it was necessary for the user to manually specify the id of humans and robots. However, it is possible to easily create an automation that enters this value, saving the user effort. In fact, the DslGenerator inserts this value in the user's place with a simple numbering from 1 to n in the order in which humans and robots are defined, as defined in the framework [25] for the id attribute.

### 5.5.3. The type of robot

Within the json file that represents the input of the framework, as presented in Section 4.2 and described in [25–27], it was necessary for the user to manually specify the speed and acceleration of the robot. However, the technical characteristics of the robot can easily be derived from the type of robot used. The DslGenerator enters the robot speed and acceleration corresponding to the value of the `type` attribute. Currently supported as robot type are the TurtleBot3 Burger [33] as `turtlebot3_burger`, the TIAGo [30] as `tiago`, the Pepper [38] as `pepper` and two dummy robots, called RExp1 as `rexp1` and RExp2 as `rexp2`, with speed (meters per second) and acceleration (meters per second squared) (0.60, 0.60) and (0.20, 0.20) respectively.

### 5.5.4. Enumeratives

Within the json file that represents the input of the framework, as presented in Section 4.2 and described in [25–27], there are many enum parameters. To make it easier to understand these values in the DSL are replaced with equivalent terms written in full, as shown in Figure 4.3.

# 6 | Evaluation

Recalling that the purpose of this thesis is to provide the tools to facilitate the use of the framework presented in Section 4.1, an important step to take is to verify that the plugin actually offers the user the desired services. Therefore the set objective is to evaluate the correct functioning of the DSL implementation and its integration with the framework. For this purpose, some integration tests have been conducted.

This chapter provides a description of the evaluation of the plugin. Section 6.1 presents the integration test with scenarios in boundary conditions. Section 6.2 presents the integration test with scenarios of experiments conducted directly on the framework.

## 6.1. Boundary conditions

For the first integration test of the plugin with the framework, two scenarios in boundary conditions are used. In the first case (test1), the minimum number of elements necessary to compose a scenario ("room") is used. In the second case (test2), several elements are used for each component of a complex scenario ("building"). From the tests it was observed that with the plugin the framework starts and works correctly returning the results of the experiments started.

**Scenario of "room"** On one floor with a simple floor plan are a human named Clara Giniro and a robot. Clara Giniro is young and healthy with speed 0.5. The robot of type turtlebot3 burger is fully charged. The robot's mission is to guide Clara Giniro to the door. The user wants to know the probability of success of the mission.

The .dsl file of the "room" scenario is shown in Listing 6.1.

Given the .dsl file in Listing 6.1, using the Generate Json command, the expected json file in Listing 6.2 was obtained. As shown at line 3 of Listing 6.2, the probability of success of the considered scenario is requested by the user.

Given the json file in Listing 6.2, using the Run Experiment command, the results in Figure 6.1 was obtained from the framework which calculates them through the Uppaal

```
// minimal components

build floor f1:
add rectangular_surface s1 with coordinates_vertex_A (0.0;0.0)
   coordinates_vertex_C (10.0;10.0)
add point_of_interest door with coordinates (9.0;7.0)

define robots:
robot r1 with coordinates (3.0;1.0) type turtlebot3_burger
   charge_percentage 100.0

define humans:
human Clara_Giniro with coordinates (1.0;3.0) speed 0.5 dexterity 0
   fatigue_profile young_healthy free_will_profile disabled

define mission m1:
do robot_leader for Clara_Giniro with target door
define queries of mission m1:
compute probability_of_success with duration 100 runs 1
```

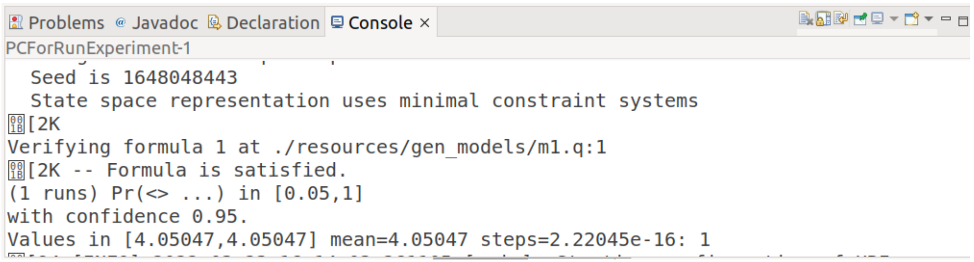
**Listing 6.1:** "room" scenario DSL file

```
{
 "queries": [
       { "type": "pscs", "tau": 100, "n": 1 }],
 "humans": [
       { "name": "Clara_Giniro", "h_id": 1, "v": 0.5, "ptrn": "FOLLOWER
          ", "p_f": "y/h", "p_fw": "d", "start": [1.0, 7.0], "dest":
          [9.0, 3.0], "dext": -1, "same_as": -1, "path": -1 }],
 "robots": [
       { "name": "r1", "r_id": 1, "v": 0.22, "a": 0.22, "start": [3.0,
          9.0], "chg": 100.0 }],
 "areas": [
       { "p1": [0.0, 0.0], "p2": [0.0, 10.0], "p3": [10.0, 10.0], "p4":
           [10.0, 0.0] }],
 "intersect": [
       { "p": [5.0, 5.0] }],
 "max_neigh": 1
}
```

**Listing 6.2:** "room" scenario json file

tool [22], showing that the framework returns the respective results of the experiments started. In particular, Figure 6.1 shows a snippet of the results regarding the probability of success of the considered scenario.

**Scenario of "building"** Considering the distances in meters and the time in seconds, the scenario is composed of a building with two floors. On the ground floor there are four patients, two doctors and three robots. Patient 029 is young, with normal free will



```
  Seed is 1648048443
  State space representation uses minimal constraint systems
[2K
Verifying formula 1 at ./resources/gen_models/m1.q:1
[2K -- Formula is satisfied.
(1 runs) Pr(<> ...) in [0.05,1]
with confidence 0.95.
Values in [4.05047,4.05047] mean=4.05047 steps=2.22045e-16: 1
```

Figure 6.1: results regarding user requests of the "room" scenario.

and speed 0.4. Patient 030 has the covid, with speed 0.4 and dexterity 1. Patient 032 is elderly, with high free will, with speed 0.6 and dexterity 1. Patient 034 is young, with normal free will, with speed 0.6 and dexterity 1. Doctor 009 is young, with low free will, speed 0.4 and dexterity 1. Doctor 010 is elderly, with normal free will, with speed 0.4 and dexterity 1. The first robot is of type pepper and with charge percentage 0.0. The second robot is of the tiago type and with a 50.0 charge. The third robot is turtlebot3 burger type and with charge percentage 100.0. In the morning the robot's mission is: to bring patient 029 to the medicines; deliver medicines to patient 030; follow patient 032 up to the medicines; compete with doctor 009 for the kit; assist patient 034 for the door; ask doctor 010 for assistance for the door. The user wants to know the probability of success, the probability of failure, the expected value of the charge, the expected value of human fatigue and run a simulation. On the first floor there are three covid patients and a robot. Patient patient 101 has speed 0.3. Patient patient 102 has speed 0.3. Patient patient 103 has speed 0.3. The robot is of the tiago type and with 90.0 charge. In the morning the robot's mission is to transport the medicines to the three patients. The user wants to know the probability of success. At lunch, the robot's mission is to deliver lunch to the three patients. The user wants to know the probability of failure.

A snippet of the .dsl file of the "building" scenario is shown in Listing 6.3.

Given the .dsl file in Listing 6.3, using the Generate Json command, the three expected json files corresponding to the three missions specified in the scenario of test2, have been obtained. A snippet of the three json files is shown in Listing 6.4, in Listing 6.5 and in Listing 6.6. As shown at lines 3-7 of Listing 6.4, are requested by the user: the simulation, the probability of success and the probability of failure of the considered scenario; the expected value of the charge of the robot; the expected value of human fatigue. As shown at line 3 of Listing 6.5, the probability of success of the considered scenario is requested by the user. As shown at line 3 of Listing 6.6, the probability of failure of the considered scenario is requested by the user.

Given the json files in Listing 6.4, in Listing 6.5 and in Listing 6.6, using the Run Exper-

```
// multiple components

parameter length_unit m

build floor ground_floor:
add rectangular_surface s1 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (1.0;5.0)
[...]
define mission morning_assignments:
do robot_leader for patient_029 with target medicines
do robot_transporter for patient_030 with target medicines
do robot_follower for patient_032 with target medicines
do robot_competitor for doctor_009 with target kit
do robot_rescuer for patient_034 with target door
do robot_client for doctor_010 with target door
define queries of mission morning_assignments:
compute simulation with duration 100 runs 1
compute probability_of_success with duration 70 runs 2
compute expected_charge with duration 80 runs 1
compute expected_fatigue with duration 80 runs 1
compute probability_of_failure with duration 70 runs 0

build floor first_floor:
add rectangular_surface s1 with coordinates_vertex_A (1.0;0.0)
    coordinates_vertex_C (3.5;3.0)
[...]
define mission morning_assignments:
do robot_transporter for patient_101 with target medicines
do robot_transporter for patient_102 with target medicines
do robot_transporter for patient_103 with target medicines
define queries of mission morning_assignments:
compute probability_of_success with duration 200 runs 1
define mission lunch_assignments:
do robot_transporter for patient_101 with target food
do robot_transporter for patient_102 with target food
do robot_transporter for patient_103 with target food
define queries of mission lunch_assignments:
compute probability_of_failure with duration 200 runs 1
```

**Listing 6.3:** snippet "building" scenario DSL file

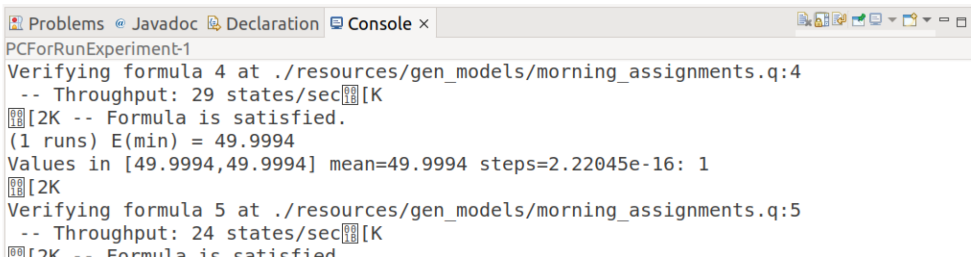
iment command, the results in Figure 6.2, in Figure 6.3 and in Figure 6.4 was obtained from the framework which calculates them through the Uppaal tool [22], showing that the framework returns the respective results of the experiments started. In particular, Figure 6.2 shows a snippet of the results regarding the expected value of the charge of the robot. In particular, Figure 6.3 shows a snippet of the results regarding the probability of success of the considered scenario. In particular, Figure 6.4 shows a snippet of the results regarding the probability of failure of the considered scenario.

```
{
 "queries": [
        { "type": "sim", "tau": 100, "n": 1 },
        { "type": "pscs", "tau": 70, "n": 2 },
        { "type": "echg", "tau": 80, "n": 1 },
        { "type": "eftg", "tau": 80, "n": 1 },
        { "type": "pfail", "tau": 70, "n": -1 }],
 "humans": [
        { "name": "patient_029", "h_id": 1, "v": 0.4, "ptrn": "FOLLOWER
           ", "p_f": "y/s", "p_fw": "n", "start": [6.5, 4.9], "dest":
           [2.2, 3.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_030", "h_id": 2, "v": 0.4, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 4.9], "dest":
           [2.2, 3.5], "dext": 1, "same_as": -1, "path": -1 },
        { "name": "patient_032", "h_id": 3, "v": 0.6, "ptrn": "LEADER",
           "p_f": "e/s", "p_fw": "h", "start": [8.5, 4.9], "dest": [2.2,
            3.5], "dext": 1, "same_as": -1, "path": -1 },
 [...]
}
```

**Listing 6.4:** snippet "building" scenario json file 1

```
{
 "queries": [
        { "type": "pscs", "tau": 200, "n": 1 }],
 "humans": [
        { "name": "patient_101", "h_id": 1, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 4.5], "dest":
           [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_102", "h_id": 2, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 3.5], "dest":
           [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_103", "h_id": 3, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 2.5], "dest":
           [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 }],
 [...]
}
```

**Listing 6.5:** snippet "building" scenario json file 2

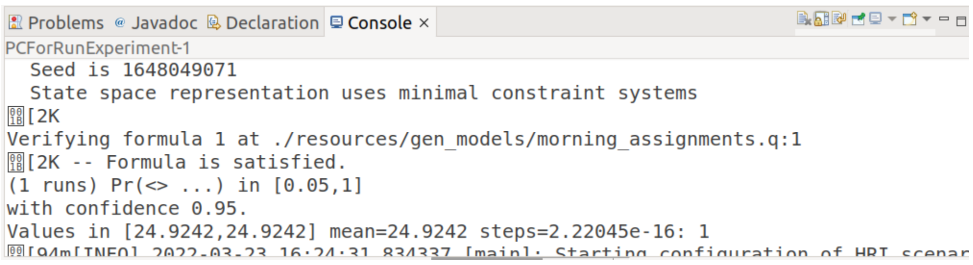


```
Verifying formula 4 at ./resources/gen_models/morning_assignments.q:4
 -- Throughput: 29 states/sec␛[K
␛[2K -- Formula is satisfied.
(1 runs) E(min) = 49.9994
Values in [49.9994,49.9994] mean=49.9994 steps=2.22045e-16: 1
␛[2K
Verifying formula 5 at ./resources/gen_models/morning_assignments.q:5
 -- Throughput: 24 states/sec␛[K
␛[2K -- Formula is satisfied
```

Figure 6.2: results regarding user requests of the "building" scenario json file 1.

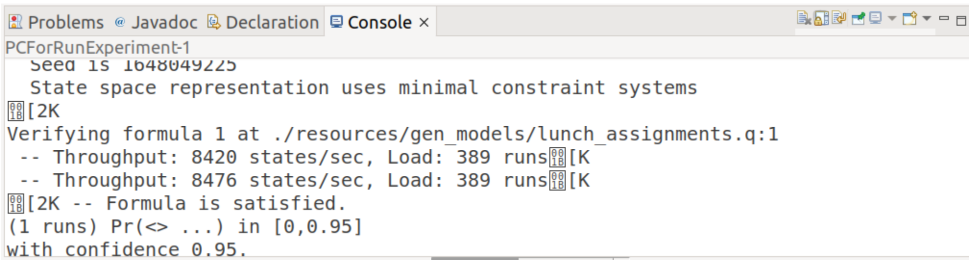
```
{
 "queries": [
        { "type": "pfail", "tau": 200, "n": 1 }],
 "humans": [
        { "name": "patient_101", "h_id": 1, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 4.5], "dest": [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_102", "h_id": 2, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 3.5], "dest": [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_103", "h_id": 3, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 2.5], "dest": [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 }],
 [...]
}
```

**Listing 6.6:** snippet "building" scenario json file 3



```
  Seed is 1648049071
  State space representation uses minimal constraint systems
[2K
Verifying formula 1 at ./resources/gen_models/morning_assignments.q:1
[2K -- Formula is satisfied.
(1 runs) Pr(<> ...) in [0.05,1]
with confidence 0.95.
Values in [24.9242,24.9242] mean=24.9242 steps=2.22045e-16: 1
[94m[INFO] 2022-03-23 16:24:31 834337 [main]: Starting configuration of HRI scenar
```

Figure 6.3: results regarding user requests of the "building" scenario json file 2.



```
  Seed is 1648049225
  State space representation uses minimal constraint systems
[2K
Verifying formula 1 at ./resources/gen_models/lunch_assignments.q:1
 -- Throughput: 8420 states/sec, Load: 389 runs[K
 -- Throughput: 8476 states/sec, Load: 389 runs[K
[2K -- Formula is satisfied.
(1 runs) Pr(<> ...) in [0,0.95]
with confidence 0.95.
```

Figure 6.4: results regarding user requests of the "building" scenario json file 3.

## 6.2. Framework experiments

For the second integration test of the plugin with the framework presented in Section 4.1, the provided scenarios of experiments conducted directly on the framework are used. These scenarios, corresponding to the experiments "small office", "laboratory" and "large office", were used to model the more complex case of a building with three floors each corresponding to one of the three experiments (test3). From the test it was observed that with the plugin the user can easily specify complex scenarios in a single file and obtain the expected json files and starts the framework that returns the respective results of the experiments started.

**Scenario of "small office"** Considering the distances in centimeters and the time in seconds, there are five humans and a robot on one floor. The human one is young and sick, with low free will and 40.0 speed. Human two is elderly and healthy and with a speed of 40.0. Human three is elderly and healthy and with a speed of 40.0. Human four is young and sick and with speed 40.0. Human five is young and sick and with 40.0 speed. The robot is of type rexp1 and with charge percentage 12.40. The robot's mission is: to guide human one to point three; follow human two up to point one; follow human three up to point two; follow human four to point four and lead human five to point two. The user wants to know the probability of success, the probability of failure, the expected value of the charge, the expected value of human fatigue and run a simulation.

**Scenario of "laboratory"** Considering the distances in centimeters and the time in seconds, there are five humans and a robot on one floor. The human one is young and sick, with low free will and 40.0 speed. Human two is elderly and healthy and with a speed of 40.0. Human three is elderly and healthy and with a speed of 40.0. Human four is young and sick and with speed 40.0. Human five is young and sick and with 40.0 speed. The robot is of type rexp1 and with charge percentage 12.40. The robot's mission is: to guide the human one to point five; to transport the object in point two to human two; follow human three up to point four; follow human four to point one and lead human five to point three. The user wants to know the probability of success, the probability of failure, the expected value of the charge, the expected value of human fatigue and run a simulation.

**Scenario of "large office"** Considering the distances in centimeters and the time in seconds, there are ten humans and a robot on one floor. Human one is young and sick, with low free will and 20.0 speed. Human two is elderly and sick and with speed 20.0. Human three is elderly and sick and with speed 20.0. Human four is young and healthy and with speed 20.0. Human five is elderly and healthy and with speed 20.0. Human six is elderly and healthy and with speed 20.0. Human seven is young and sick and with speed 20.0. Human eight is young and sick and with speed 20.0. Human nine is elderly and sick and with speed 20.0. Human ten is elderly and sick and with speed 20.0. The robot is of type rexp2 and with charge percentage 12.40. The robot's mission is: to guide human one to point two; follow human two up to point three; lead human three to point four; carry the object in point five to human four; follow human five up to point six; follow human six to point seven; follow human seven up to point one; lead the human eight to point eight; follow human nine to point four and lead human ten to point nine. The user wants to know the probability of success and run a simulation.

A snippet of the .dsl file of the considered scenario is shown in Listing 6.7. In particular on floor `f_1` "small office" scenario is considered, on floor `f_2` "laboratory" scenario is considered, on floor `f_3` "large office" scenario is considered.

```
// framework experiments

parameter length_unit cm

build floor f_1:
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
    coordinates_vertex_C (1550.0;740.0)
[...]
define mission exp4:
do robot_leader for h_1 with target p_3
do robot_follower for h_2 with target p_1
do robot_follower for h_3 with target p_2
do robot_follower for h_4 with target p_4
do robot_leader for h_5 with target p_2
[...]

build floor f_2:
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
    coordinates_vertex_C (1550.0;740.0)
[...]
define queries of mission exp5 :
compute probability_of_success with duration 280 runs 10
compute probability_of_failure with duration 400 runs 1
compute simulation with duration 400 runs 1
compute expected_fatigue with duration 400 runs 0
compute expected_charge with duration 400 runs 0

build floor f_3 :
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
    coordinates_vertex_C (1550.0;740.0)
[...]
define queries of mission exp6 :
compute probability_of_success with duration 1300 runs 10
compute simulation with duration 1300 runs 1
```

**Listing 6.7:** snippet test3 DSL file

Given the .dsl file in Listing 6.7, using the Generate Json command, the three expected json files corresponding to the three missions specified in the scenario of test3, have been obtained. A snippet of the three json files is shown in Listing 6.8, in Listing 6.9 and in Listing 6.10. As shown at lines 3-7 of Listing 6.8, are requested by the user: the simulation, the probability of success and the probability of failure of the "small office" scenario; the expected value of the charge of the robot; the expected value of human fatigue. As shown at lines 3-7 of Listing 6.9, are requested by the user: the simulation, the probability of success and the probability of failure of the "laboratory" scenario; the expected value of the charge of the robot; the expected value of human fatigue. As shown at lines 3-4 of Listing 6.10, the probability of success and the simulation of the "large office" scenario is requested by the user.

```
1  {
2   "queries": [
3          { "type": "pscs", "tau": 280, "n": 10 },
4          { "type": "pfail", "tau": 400, "n": 1 },
5          { "type": "sim", "tau": 400, "n": 1 },
6          { "type": "eftg", "tau": 400, "n": -1 },
7          { "type": "echg", "tau": 400, "n": -1 }],
8   "humans": [
9          { "name": "h_1", "h_id": 1, "v": 40.0, "ptrn": "FOLLOWER", "p_f
              ": "y/s", "p_fw": "1", "start": [1400.0, 200.0], "dest":
              [400.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
10 [...]
11 }
```

**Listing 6.8:** snippet "small office" scenario json file 1

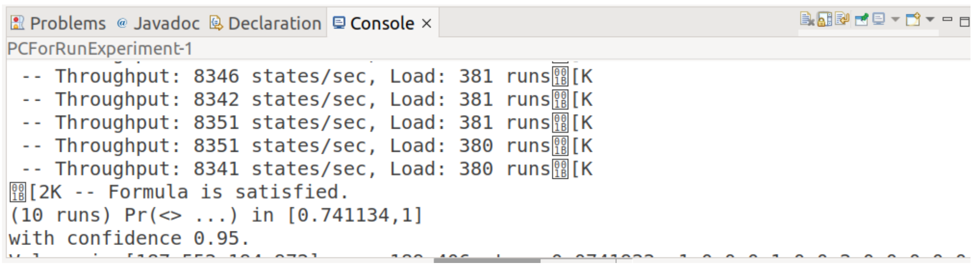
```
1  {
2   "queries": [
3          { "type": "pscs", "tau": 280, "n": 10 },
4          { "type": "pfail", "tau": 400, "n": 1 },
5          { "type": "sim", "tau": 400, "n": 1 },
6          { "type": "eftg", "tau": 400, "n": -1 },
7          { "type": "echg", "tau": 400, "n": -1 }],
8   "humans": [
9          { "name": "h_1", "h_id": 1, "v": 40.0, "ptrn": "FOLLOWER", "p_f
              ": "y/s", "p_fw": "1", "start": [2500.0, 200.0], "dest":
              [1500.0, 400.0], "dext": -1, "same_as": -1, "path": -1 },
10 [...]
11 }
```

**Listing 6.9:** snippet "laboratory" scenario json file 2

Given the json files in Listing 6.8, in Listing 6.9 and in Listing 6.10, using the Run Experiment command, the results in Figure 6.5, in Figure 6.6 and in Figure 6.7 was obtained from the framework which calculates them through the Uppaal tool [22], showing that the framework returns the respective results of the experiments started. In particular, Figure 6.5 shows a snippet of the results regarding the probability of success of the "small office" scenario. In particular, Figure 6.6 shows a snippet of the results regarding the expected value of the charge of the robot. In particular, Figure 6.7 shows a snippet of the results regarding the simulation of the "large office" scenario.

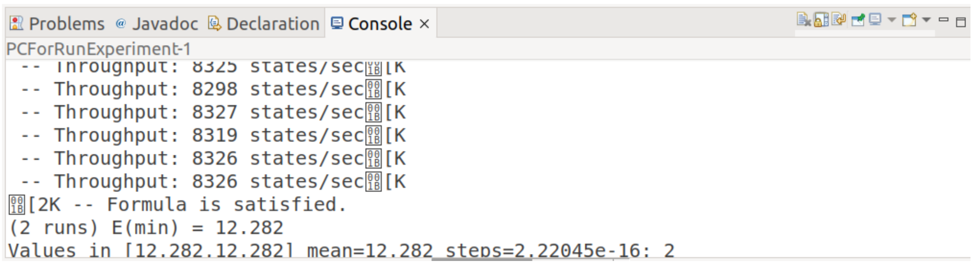
```
{
 "queries": [
         { "type": "pscs", "tau": 1300, "n": 10 },
         { "type": "sim", "tau": 1300, "n": 1 }],
 "humans": [
         { "name": "h_1", "h_id": 1, "v": 20.0, "ptrn": "FOLLOWER", "p_f
            ": "y/s", "p_fw": "l", "start": [1500.0, 200.0], "dest":
            [1500.0, 700.0], "dext": -1, "same_as": -1, "path": -1 },
         { "name": "h_2", "h_id": 2, "v": 20.0, "ptrn": "LEADER", "p_f":
            "e/s", "p_fw": "d", "start": [1400.0, 400.0], "dest":
            [1400.0, 500.0], "dext": -1, "same_as": -1, "path": -1 },
 [...]
}
```

**Listing 6.10:** snippet "large office" scenario json file 3



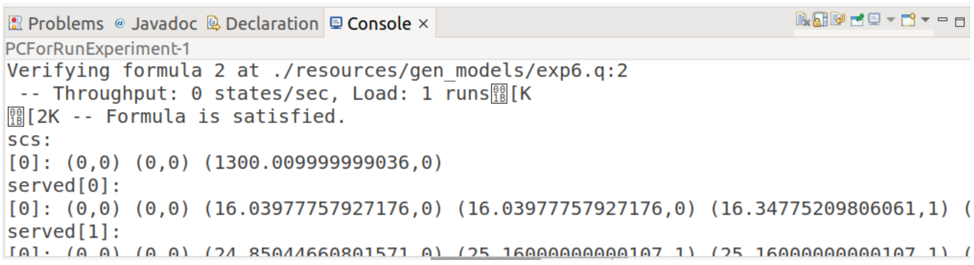
```
Problems  Javadoc  Declaration  Console ×
PCForRunExperiment-1
 -- Throughput: 8346 states/sec, Load: 381 runs▯[K
 -- Throughput: 8342 states/sec, Load: 381 runs▯[K
 -- Throughput: 8351 states/sec, Load: 381 runs▯[K
 -- Throughput: 8351 states/sec, Load: 380 runs▯[K
 -- Throughput: 8341 states/sec, Load: 380 runs▯[K
▯[2K -- Formula is satisfied.
(10 runs) Pr(<> ...) in [0.741134,1]
with confidence 0.95.
```

Figure 6.5: results regarding user requests of the "small office" scenario.



```
Problems  Javadoc  Declaration  Console ×
PCForRunExperiment-1
 -- Throughput: 8325 states/sec▯[K
 -- Throughput: 8298 states/sec▯[K
 -- Throughput: 8327 states/sec▯[K
 -- Throughput: 8319 states/sec▯[K
 -- Throughput: 8326 states/sec▯[K
 -- Throughput: 8326 states/sec▯[K
▯[2K -- Formula is satisfied.
(2 runs) E(min) = 12.282
Values in [12.282,12.282] mean=12.282 steps=2.22045e-16: 2
```

Figure 6.6: results regarding user requests of the "laboratory" scenario.



```
Problems  Javadoc  Declaration  Console ×
PCForRunExperiment-1
Verifying formula 2 at ./resources/gen_models/exp6.q:2
 -- Throughput: 0 states/sec, Load: 1 runs▯[K
▯[2K -- Formula is satisfied.
scs:
[0]: (0,0) (0,0) (1300.009999999036,0)
served[0]:
[0]: (0,0) (0,0) (16.03977757927176,0) (16.03977757927176,0) (16.34775209806061,1) (
served[1]:
```

Figure 6.7: results regarding user requests of the "large office" scenario.

# 7 | Conclusion

In this work it is introduced how DSLs are an interesting tool to support the development of robotic applications, providing a common ground for heterogeneous teams and an easy to use and quick to learn tool also for users without technical knowledge. In particular, the purpose of this thesis is to present:

- a DSL developed as a support tool for a specific framework;
- a plugin that implements the developed DSL.

For this purpose the language domain is first defined. Then the metamodel of the DSL elements is derived. Then the concrete syntax of the DSL is obtained. Finally, the abstract syntax is obtained. Once the language is finished, the developed plugin is presented. Finally, to evaluate the correct functioning of the DSL implementation and its integration with the framework, some integration tests have been conducted. For the first integration test of the plugin with the framework, two scenarios in boundary conditions are used. For the second integration test of the plugin with the framework, the provided scenarios of experiments conducted directly on the framework are used. Thanks to these tests it has been verified that the plugin actually offers the user the desired services. In fact, from the tests it was observed:

- how the user can easily specify complex scenarios in a single file and obtain the respective json files with just a few clicks;
- how with the plugin, the framework starts and works correctly returning the results of the experiments started.

To conclude we consider some possible extensions of the work presented, as future work:

- involve potential DSL users as testers from which to collect feedback about how make the language more user-friendly and extend the DSL based on the preferences expressed by users;
- improve the implementation of the DSL by adding a graphic support that synthetically represents the scenario specified with the DSL in order to give the user

an immediate view of the arrangement of the various elements represented in the scenario.

# Bibliography

[1] Introducing JSON. URL `https://www.json.org/json-en.html`.

[2] Xtext. URL `https://www.eclipse.org/Xtext/`.

[3] ANTLR. URL `https://www.antlr.org/`.

[4] Racket. URL `https://racket-lang.org/`.

[5] Xtext - The Grammar Language, . URL `https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html`.

[6] Xtext - Language Implementation, . URL `https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html`.

[7] Xtext - 15 Minutes Tutorial, . URL `https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html`.

[8] G. Agha and K. Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1):1–39, 2018.

[9] M. M. Bersani, M. Soldo, C. Menghi, P. Pelliccione, and M. Rossi. Pursue-from specification of robotic environments to synthesis of controllers. *Formal Aspects of Computing*, 32(2):187–227, 2020.

[10] J. P. Buch, J. S. Laursen, L. C. Sørensen, L.-P. Ellekilde, D. Kraft, U. P. Schultz, and H. G. Petersen. Applying simulation and a domain-specific language for an adaptive action library. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 86–97. Springer, 2014.

[11] P. Detzner, T. Kirks, and J. Jost. A novel task language for natural interaction in human-robot systems for warehouse logistics. In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pages 725–730. IEEE, 2019.

[12] Diligent Robotics. Moxi. URL `https://www.diligentrobots.com/moxi`.

[13] P. Doherty, F. Heintz, and D. Landén. A distributed task specification language for

mixed-initiative delegation. In *International conference on principles and practice of multi-agent systems*, pages 42–57. Springer, 2010.

[14] Eclipse Foundation. Eclipse desktop & web IDEs. URL `https://www.eclipse.org/ide/`.

[15] C. Finucane, G. Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1988–1993. IEEE, 2010.

[16] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures. High-level mission specification for multiple robots. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*, pages 127–140, 2019.

[17] L. M. Garshol. BNF and EBNF: What are they and how do they work?, 2008. URL `https://www.garshol.priv.no/download/text/bnf.html`.

[18] S. Götz, M. Leuthäuser, J. Reimann, J. Schroeter, C. Wende, C. Wilke, and U. Aßmann. A role-based language for collaborative robot applications. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 1–15. Springer, 2011.

[19] JetBrains. MPS: The Domain-Specific Language Creator. URL `https://www.jetbrains.com/mps/`.

[20] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*, 2014.

[21] D. Kramer, T. Clark, and S. Oussena. Mobdsl: A domain specific language for multiple mobile platform deployment. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, pages 1–7. IEEE, 2010.

[22] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1):134–152, 1997.

[23] L. Lestingi. HRI Designtime. URL `https://github.com/LesLivia/hri_designtime`.

[24] L. Lestingi. Model-driven development of formally verified human-robot interactions. In *Doctoral Symposium-24th International Symposium on Formal Methods (FM 2021)*, 2021.

[25] L. Lestingi, M. Askarpour, M. M. Bersani, and M. Rossi. Formal verification of

human-robot interaction in healthcare scenarios. In *International Conference on Software Engineering and Formal Methods*, pages 303–324. Springer, 2020.

[26] L. Lestingi, M. Askarpour, M. M. Bersani, and M. Rossi. A model-driven approach for the formal analysis of human-robot interaction scenarios. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1907–1914. IEEE, 2020.

[27] L. Lestingi, M. Askarpour, M. M. Bersani, and M. Rossi. A deployment framework for formally verified human-robot interactions. *IEEE Access*, 9:136616–136635, 2021.

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

[29] R. Membarth, O. Reiche, F. Hannig, J. Teich, M. Körner, and W. Eckert. Hipa cc: A domain-specific language and compiler for image processing. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):210–224, 2015.

[30] PAL Robotics. TIAGo. URL `https://pal-robotics.com/robots/tiago/`.

[31] L. Réveillere, F. Mérillon, C. Consel, R. Marlet, and G. Muller. A dsl approach to improve productivity and safety in device drivers development. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, pages 101–109. IEEE, 2000.

[32] RobotikLAB. Alice. URL `https://robotiklab.co.uk/alice/`.

[33] ROBOTIS. TurtleBot3. URL `https://emanual.robotis.com/docs/en/platform/turtlebot3/features/`.

[34] S. Tousignant, E. Van Wyk, and M. Gini. Xrobots: A flexible language for programming mobile robots based on hierarchical state machines. In *2012 IEEE International Conference on Robotics and Automation*, pages 1773–1778. IEEE, 2012.

[35] A. Van Deursen and P. Klint. Domain-specific language design requires feature descriptions. *Journal of computing and information technology*, 10(1):1–17, 2002.

[36] A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[37] Wikipedia contributors. Framework — Wikipedia, The Free Encyclopedia, 2021. URL `https://en.wikipedia.org/wiki/Framework`.

[38] Wikipedia contributors. Pepper (robot) — Wikipedia, The Free Encyclopedia, 2022. URL `https://en.wikipedia.org/wiki/Pepper_(robot)`.

[39] L. Zhang, H. Zhang, Z. Fang, R. Zapata, and M. Huchard. A domain specific architecture description language for autonomous mobile robots. In *2012 IEEE International Conference on Information and Automation*, pages 283–288. IEEE, 2012.

# A | Appendix A

The appendix shows the code of the dsl file and the code of the json files of both test2 and test3.

The .dsl file of the "building" scenario is shown in Listing A.1.

```
// multiple components

parameter length_unit m

build floor ground_floor:
add rectangular_surface s1 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (1.0;5.0)
add rectangular_surface s2 with coordinates_vertex_A (0.0;2.0)
    coordinates_vertex_C (9.0;5.0)
add rectangular_surface s3 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (3.0;1.0)
add rectangular_surface s4 with coordinates_vertex_A (2.0;0.0)
    coordinates_vertex_C (3.0;5.0)
add rectangular_surface s5 with coordinates_vertex_A (2.0;1.0)
    coordinates_vertex_C (7.0;5.0)
add rectangular_surface s6 with coordinates_vertex_A (6.0;0.0)
    coordinates_vertex_C (9.0;1.0)
add rectangular_surface s7 with coordinates_vertex_A (6.0;0.0)
    coordinates_vertex_C (7.0;5.0)
add rectangular_surface s8 with coordinates_vertex_A (8.0;0.0)
    coordinates_vertex_C (9.0;5.0)
add point_of_interest kit with coordinates (2.2;1.5)
add point_of_interest medicines with coordinates (2.2;1.5)
add point_of_interest door with coordinates (4.5;1.0)

define robots:
robot r1 with coordinates (4.0;2.8) type pepper charge_percentage 0.0
robot r2 with coordinates (4.5;2.8) type tiago charge_percentage 50.0
robot r3 with coordinates (5.0;2.8) type turtlebot3_burger
    charge_percentage 100.0

```

```
define humans:
human patient_029 with coordinates (6.5;0.1) speed 0.4 dexterity 0
    fatigue_profile young_sick free_will_profile normal
human patient_030 with coordinates (7.5;0.1) speed 0.4  dexterity 1
    fatigue_profile covid_patient free_will_profile disabled
human patient_032 with coordinates (8.5;0.1) speed 0.6 dexterity 1
    fatigue_profile elderly_sick free_will_profile high
human doctor_009 with coordinates (8.9;0.5) speed 0.4 dexterity 1
    fatigue_profile young_healthy free_will_profile low
human patient_034 with coordinates (8.9;1.5) speed 0.6 dexterity 1
    fatigue_profile young_sick free_will_profile normal
human doctor_010 with coordinates (8.9;2.5) speed 0.4 dexterity 1
    fatigue_profile elderly_healthy free_will_profile normal

define mission morning_assignments:
do robot_leader for patient_029 with target medicines
do robot_transporter for patient_030 with target medicines
do robot_follower for patient_032 with target medicines
do robot_competitor for doctor_009 with target kit
do robot_rescuer for patient_034 with target door
do robot_client for doctor_010 with target door

define queries of mission morning_assignments:
compute simulation with duration 100 runs 1
compute probability_of_success with duration 70 runs 2
compute expected_charge with duration 80 runs 1
compute expected_fatigue with duration 80 runs 1
compute probability_of_failure with duration 70 runs 0

//-------------------------------------------------

build floor first_floor:
add rectangular_surface s1 with coordinates_vertex_A (1.0;0.0)
    coordinates_vertex_C (3.5;3.0)
add rectangular_surface s2 with coordinates_vertex_A (5.5;0.0)
    coordinates_vertex_C (8.0;3.0)

add rectangular_surface s3 with coordinates_vertex_A (2.0;1.0)
    coordinates_vertex_C (7.0;5.0)
add rectangular_surface s4 with coordinates_vertex_A (0.0;4.0)
    coordinates_vertex_C (9.0;5.0)

add point_of_interest medicines with coordinates (0.5;4.5)
add point_of_interest food with coordinates (8.5;4.5)
```

```

define robots:
robot r1 with coordinates (1.5;2.0) type tiago charge_percentage 90.0

define humans:
human patient_101 with coordinates (7.5;0.5) speed 0.3 dexterity 0
    fatigue_profile covid_patient free_will_profile disabled
human patient_102 with coordinates (7.5;1.5) speed 0.3 dexterity 0
    fatigue_profile covid_patient free_will_profile disabled
human patient_103 with coordinates (7.5;2.5) speed 0.3 dexterity 0
    fatigue_profile covid_patient free_will_profile disabled

define mission morning_assignments:
do robot_transporter for patient_101 with target medicines
do robot_transporter for patient_102 with target medicines
do robot_transporter for patient_103 with target medicines
define queries of mission morning_assignments:
compute probability_of_success with duration 200 runs 1

define mission lunch_assignments:
do robot_transporter for patient_101 with target food
do robot_transporter for patient_102 with target food
do robot_transporter for patient_103 with target food
define queries of mission lunch_assignments:
compute probability_of_failure with duration 200 runs 1
```

**Listing A.1:** "building" scenario DSL file

The three json files of the "building" scenario are shown in Listing A.2, in Listing A.3 and in Listing A.4.

```
{
 "queries": [
        { "type": "sim", "tau": 100, "n": 1 },
        { "type": "pscs", "tau": 70, "n": 2 },
        { "type": "echg", "tau": 80, "n": 1 },
        { "type": "eftg", "tau": 80, "n": 1 },
        { "type": "pfail", "tau": 70, "n": -1 }],
 "humans": [
        { "name": "patient_029", "h_id": 1, "v": 0.4, "ptrn": "FOLLOWER
           ", "p_f": "y/s", "p_fw": "n", "start": [6.5, 4.9], "dest":
           [2.2, 3.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_030", "h_id": 2, "v": 0.4, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 4.9], "dest":
           [2.2, 3.5], "dext": 1, "same_as": -1, "path": -1 },
        { "name": "patient_032", "h_id": 3, "v": 0.6, "ptrn": "LEADER", "p_f": "e/s", "p_fw": "h", "start": [8.5, 4.9], "dest": [2.2, 3.5], "dext": 1, "same_as": -1, "path": -1 },
        { "name": "doctor_009", "h_id": 4, "v": 0.4, "ptrn": "COMPETITOR", "p_f": "y/h", "p_fw": "l", "start": [8.9, 4.5], "dest": [2.2, 3.5], "dext": 1, "same_as": -1, "path": -1 },
        { "name": "patient_034", "h_id": 5, "v": 0.6, "ptrn": "ASSISTANT", "p_f": "y/s", "p_fw": "n", "start": [8.9, 3.5], "dest": [4.5, 4.0], "dext": 1, "same_as": -1, "path": -1 },
        { "name": "doctor_010", "h_id": 6, "v": 0.4, "ptrn": "RESCUER", "p_f": "e/h", "p_fw": "n", "start": [8.9, 2.5], "dest": [4.5, 4.0], "dext": 1, "same_as": -1, "path": -1 }],
  "robots": [
        { "name": "r1", "r_id": 1, "v": 0.83, "a": 0.83, "start": [4.0, 2.2], "chg": 0.0 },
        { "name": "r2", "r_id": 2, "v": 1.0, "a": 1.0, "start": [4.5, 2.2], "chg": 50.0 },
        { "name": "r3", "r_id": 3, "v": 0.22, "a": 0.22, "start": [5.0, 2.2], "chg": 100.0 }],
  "areas": [
        { "p1": [0.0, 0.0], "p2": [0.0, 5.0], "p3": [1.0, 5.0], "p4": [1.0, 0.0] },
        { "p1": [0.0, 0.0], "p2": [0.0, 3.0], "p3": [9.0, 3.0], "p4": [9.0, 0.0] },
        { "p1": [0.0, 4.0], "p2": [0.0, 5.0], "p3": [3.0, 5.0], "p4": [3.0, 4.0] },
        { "p1": [2.0, 0.0], "p2": [2.0, 5.0], "p3": [3.0, 5.0], "p4": [3.0, 0.0] },
        { "p1": [2.0, 0.0], "p2": [2.0, 4.0], "p3": [7.0, 4.0], "p4": [7.0, 0.0] },
        { "p1": [6.0, 4.0], "p2": [6.0, 5.0], "p3": [9.0, 5.0], "p4": [9.0, 4.0] },
        { "p1": [6.0, 0.0], "p2": [6.0, 5.0], "p3": [7.0, 5.0], "p4": [7.0, 0.0] },
        { "p1": [8.0, 0.0], "p2": [8.0, 5.0], "p3": [9.0, 5.0], "p4": [9.0, 0.0] }],
  "intersect": [
        { "p": [0.5, 4.5] },
        { "p": [2.5, 1.5] },
        { "p": [8.5, 4.5] },
        { "p": [6.5, 2.0] },
        { "p": [0.5, 1.5] },
        { "p": [2.5, 4.5] },
        { "p": [8.5, 1.5] },
        { "p": [4.5, 1.5] },
        { "p": [6.5, 4.5] },
        { "p": [2.5, 2.0] },
        { "p": [6.5, 1.5] }],
 "max_neigh": 8
}
```

**Listing A.2:** "building" scenario json file 1

```
{
 "queries": [
        { "type": "pscs", "tau": 200, "n": 1 }],
 "humans": [
        { "name": "patient_101", "h_id": 1, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 4.5], "dest": [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_102", "h_id": 2, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 3.5], "dest": [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_103", "h_id": 3, "v": 0.3, "ptrn": "RECIPIENT", "p_f": "c", "p_fw": "d", "start": [7.5, 2.5], "dest": [0.5, 0.5], "dext": -1, "same_as": -1, "path": -1 }],
 "robots": [
        { "name": "r1", "r_id": 1, "v": 1.0, "a": 1.0, "start": [1.5, 3.0], "chg": 90.0 }],
 "areas": [
        { "p1": [1.0, 2.0], "p2": [1.0, 5.0], "p3": [3.5, 5.0], "p4": [3.5, 2.0] },
        { "p1": [5.5, 2.0], "p2": [5.5, 5.0], "p3": [8.0, 5.0], "p4": [8.0, 2.0] },
        { "p1": [2.0, 0.0], "p2": [2.0, 4.0], "p3": [7.0, 4.0], "p4": [7.0, 0.0] },
        { "p1": [0.0, 0.0], "p2": [0.0, 1.0], "p3": [9.0, 1.0], "p4": [9.0, 0.0] }],
 "intersect": [
        { "p": [2.75, 3.0] },
        { "p": [6.25, 3.0] },
        { "p": [4.5, 0.5] }],
 "max_neigh": 3
}
```

**Listing A.3:** "building" scenario json file 2

```
{
 "queries": [
        { "type": "pfail", "tau": 200, "n": 1 }],
 "humans": [
        { "name": "patient_101", "h_id": 1, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 4.5], "dest":
           [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_102", "h_id": 2, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 3.5], "dest":
           [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "patient_103", "h_id": 3, "v": 0.3, "ptrn": "RECIPIENT
           ", "p_f": "c", "p_fw": "d", "start": [7.5, 2.5], "dest":
           [8.5, 0.5], "dext": -1, "same_as": -1, "path": -1 }],
 "robots": [
        { "name": "r1", "r_id": 1, "v": 1.0, "a": 1.0, "start": [1.5,
           3.0], "chg": 90.0 }],
 "areas": [
        { "p1": [1.0, 2.0], "p2": [1.0, 5.0], "p3": [3.5, 5.0], "p4":
           [3.5, 2.0] },
        { "p1": [5.5, 2.0], "p2": [5.5, 5.0], "p3": [8.0, 5.0], "p4":
           [8.0, 2.0] },
        { "p1": [2.0, 0.0], "p2": [2.0, 4.0], "p3": [7.0, 4.0], "p4":
           [7.0, 0.0] },
        { "p1": [0.0, 0.0], "p2": [0.0, 1.0], "p3": [9.0, 1.0], "p4":
           [9.0, 0.0] }],
 "intersect": [
        { "p": [2.75, 3.0] },
        { "p": [6.25, 3.0] },
        { "p": [4.5, 0.5] }],
 "max_neigh": 3
}
```

**Listing A.4:** "building" scenario json file 3

The .dsl file of the test3 is shown in Listing A.5.

```
// framework experiments

parameter length_unit cm

build floor f_1:
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
   coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_2 with coordinates_vertex_A (0.0;0.0)
   coordinates_vertex_C (185.0;740.0)
add rectangular_surface s_3 with coordinates_vertex_A (0.0;0.0)
   coordinates_vertex_C (1550.0;177.5)
```

```
add rectangular_surface s_4 with coordinates_vertex_A (1352.0;0.0)
    coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_5 with coordinates_vertex_A (2970.0;550.5)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_6 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_7 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (4512.5;177.5)
add rectangular_surface s_8 with coordinates_vertex_A (4322.0;0.0)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_9 with coordinates_vertex_A (1945.0;155.0)
    coordinates_vertex_C (2670.0;850.0)
add rectangular_surface s_10 with coordinates_vertex_A (1352.0;425.0)
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_11 with coordinates_vertex_A (1802.5;740.0)
    coordinates_vertex_C (2800.0;850.0)
add point_of_interest p_1 with coordinates (1400.0;690.0)
add point_of_interest p_2 with coordinates (3000.0;650.0)
add point_of_interest p_3 with coordinates (400.0;650.0)
add point_of_interest p_4 with coordinates (1400.0;650.0)

define robots :
robot r_1 with coordinates (1400.0;640.0) type rexp1 charge_percentage
    12.40

define humans :
human h_1 with coordinates (1400.0;650.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile low
human h_2 with coordinates (3000.0;650.0) speed 40.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_3 with coordinates (1400.0;690.0) speed 40.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_4 with coordinates (400.0;650.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled
human h_5 with coordinates (1500.0;650.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled

define mission exp4:
do robot_leader for h_1 with target p_3
do robot_follower for h_2 with target p_1
do robot_follower for h_3 with target p_2
do robot_follower for h_4 with target p_4
do robot_leader for h_5 with target p_2
define queries of mission exp4:
```

```
compute probability_of_success with duration 280 runs 10
compute probability_of_failure with duration 400 runs 1
compute simulation with duration 400 runs 1
compute expected_fatigue with duration 400 runs 0
compute expected_charge with duration 400 runs 0

//----------------------------------------------

build floor f_2:
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
    coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_2 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (185.0;740.0)
add rectangular_surface s_3 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (1550.0;177.5)
add rectangular_surface s_4 with coordinates_vertex_A (1352.0;0.0)
    coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_5 with coordinates_vertex_A (2970.0;550.5)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_6 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_7 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (4512.5;177.5)
add rectangular_surface s_8 with coordinates_vertex_A (4322.0;0.0)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_9 with coordinates_vertex_A (1945.0;155.0)
    coordinates_vertex_C (2670.0;850.0)
add rectangular_surface s_10 with coordinates_vertex_A (1352.0;425.0)
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_11 with coordinates_vertex_A (1802.5;740.0)
    coordinates_vertex_C (2800.0;850.0)
add point_of_interest p_1 with coordinates (900.0;150.0)
add point_of_interest p_2 with coordinates (3000.0;650.0)
add point_of_interest p_3 with coordinates (3000.0;700.0)
add point_of_interest p_4 with coordinates (3500.0;700.0)
add point_of_interest p_5 with coordinates (1500.0;450.0)

define robots :
robot r_1 with coordinates (2500.0;450.0) type rexp1 charge_percentage
    12.40

define humans :
human h_1 with coordinates (2500.0;650.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile low
```

```
human h_2 with coordinates (4000.0;700.0) speed 40.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_3 with coordinates (4100.0;700.0) speed 40.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_4 with coordinates (800.0;150.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled
human h_5 with coordinates (1900.0;650.0) speed 40.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled

define mission exp5 :
do robot_leader for h_1 with target p_5
do robot_transporter for h_2 with target p_2
do robot_follower for h_3 with target p_4
do robot_follower for h_4 with target p_1
do robot_leader for h_5 with target p_3
define queries of mission exp5 :
compute probability_of_success with duration 280 runs 10
compute probability_of_failure with duration 400 runs 1
compute simulation with duration 400 runs 1
compute expected_fatigue with duration 400 runs 0
compute expected_charge with duration 400 runs 0

//-------------------------------------------

build floor f_3 :
add rectangular_surface s_1 with coordinates_vertex_A (0.0;550.5)
    coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_2 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (185.0;740.0)
add rectangular_surface s_3 with coordinates_vertex_A (0.0;0.0)
    coordinates_vertex_C (1550.0;177.5)
add rectangular_surface s_4 with coordinates_vertex_A (1352.0;0.0)
    coordinates_vertex_C (1550.0;740.0)
add rectangular_surface s_5 with coordinates_vertex_A (2970.0;550.5)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_6 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_7 with coordinates_vertex_A (2970.0;0.0)
    coordinates_vertex_C (4512.5;177.5)
add rectangular_surface s_8 with coordinates_vertex_A (4322.0;0.0)
    coordinates_vertex_C (4512.5;740.0)
add rectangular_surface s_9 with coordinates_vertex_A (1945.0;155.0)
    coordinates_vertex_C (2670.0;850.0)
add rectangular_surface s_10 with coordinates_vertex_A (1352.0;425.0)
```

```
    coordinates_vertex_C (3155.0;740.0)
add rectangular_surface s_11 with coordinates_vertex_A (1802.5;740.0)
    coordinates_vertex_C (2800.0;850.0)
add point_of_interest p_1 with coordinates (1100.0;150.0)
add point_of_interest p_2 with coordinates (1500.0;150.0)
add point_of_interest p_3 with coordinates (1400.0;350.0)
add point_of_interest p_4 with coordinates (2500.0;650.0)
add point_of_interest p_5 with coordinates (1500.0;650.0)
add point_of_interest p_6 with coordinates (3000.0;650.0)
add point_of_interest p_7 with coordinates (4000.0;650.0)
add point_of_interest p_8 with coordinates (500.0;650.0)
add point_of_interest p_9 with coordinates (3500.0;650.0)

define robots :
robot r_1 with coordinates (1500.0;450.0) type rexp2 charge_percentage
    12.40

define humans :
human h_1 with coordinates (1500.0;650.0) speed 20.0 dexterity 0
    fatigue_profile young_sick free_will_profile low
human h_2 with coordinates (1400.0;450.0) speed 20.0 dexterity 0
    fatigue_profile elderly_sick free_will_profile disabled
human h_3 with coordinates (1400.0;650.0) speed 20.0 dexterity 0
    fatigue_profile elderly_sick free_will_profile disabled
human h_4 with coordinates (200.0;700.0) speed 20.0 dexterity 0
    fatigue_profile young_healthy free_will_profile disabled
human h_5 with coordinates (4000.0;650.0) speed 20.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_6 with coordinates (3000.0;650.0) speed 20.0 dexterity 0
    fatigue_profile elderly_healthy free_will_profile disabled
human h_7 with coordinates (1000.0;150.0) speed 20.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled
human h_8 with coordinates (1100.0;650.0) speed 20.0 dexterity 0
    fatigue_profile young_sick free_will_profile disabled
human h_9 with coordinates (1500.0;550.0) speed 20.0 dexterity 0
    fatigue_profile elderly_sick free_will_profile disabled
human h_10 with coordinates (2500.0;650.0) speed 20.0 dexterity 0
    fatigue_profile elderly_sick free_will_profile disabled

define mission exp6 :
do robot_leader for h_1 with target p_2
do robot_follower for h_2 with target p_3
do robot_leader for h_3 with target p_4
do robot_transporter for h_4 with target p_5
do robot_follower for h_5 with target p_6
do robot_follower for h_6 with target p_7
do robot_follower for h_7 with target p_1
do robot_leader for h_8 with target p_8
do robot_follower for h_9 with target p_4
do robot_leader for h_10 with target p_9
define queries of mission exp6 :
compute probability_of_success with duration 1300 runs 10
compute simulation with duration 1300 runs 1
```

**Listing A.5:** test3 DSL file

The json file of the "small office" scenario is shown in Listing A.6.

```
{
 "queries": [
        { "type": "pscs", "tau": 280, "n": 10 },
        { "type": "pfail", "tau": 400, "n": 1 },
        { "type": "sim", "tau": 400, "n": 1 },
        { "type": "eftg", "tau": 400, "n": -1 },
        { "type": "echg", "tau": 400, "n": -1 }],
 "humans": [
        { "name": "h_1", "h_id": 1, "v": 40.0, "ptrn": "FOLLOWER", "p_f
           ": "y/s", "p_fw": "l", "start": [1400.0, 200.0], "dest":
           [400.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_2", "h_id": 2, "v": 40.0, "ptrn": "LEADER", "p_f":
           "e/h", "p_fw": "d", "start": [3000.0, 200.0], "dest":
           [1400.0, 160.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_3", "h_id": 3, "v": 40.0, "ptrn": "LEADER", "p_f":
           "e/h", "p_fw": "d", "start": [1400.0, 160.0], "dest":
           [3000.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_4", "h_id": 4, "v": 40.0, "ptrn": "LEADER", "p_f":
           "y/s", "p_fw": "d", "start": [400.0, 200.0], "dest": [1400.0,
            200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_5", "h_id": 5, "v": 40.0, "ptrn": "FOLLOWER", "p_f
           ": "y/s", "p_fw": "d", "start": [1500.0, 200.0], "dest":
           [3000.0, 200.0], "dext": -1, "same_as": -1, "path": -1 }],
 "robots": [
        { "name": "r_1", "r_id": 1, "v": 60.000004, "a": 60.000004, "
           start": [1400.0, 210.0], "chg": 12.4 }],
 "areas": [
        { "p1": [0.0, 110.0], "p2": [0.0, 299.5], "p3": [1550.0, 299.5],
            "p4": [1550.0, 110.0] },
        { "p1": [0.0, 110.0], "p2": [0.0, 850.0], "p3": [185.0, 850.0],
           "p4": [185.0, 110.0] },
        { "p1": [0.0, 672.5], "p2": [0.0, 850.0], "p3": [1550.0, 850.0], "p4": [1550.0, 672.5] },
        { "p1": [1352.0, 110.0], "p2": [1352.0, 850.0], "p3": [1550.0, 850.0], "p4": [1550.0, 110.0] },
        { "p1": [2970.0, 110.0], "p2": [2970.0, 299.5], "p3": [4512.5, 299.5], "p4": [4512.5, 110.0] },
        { "p1": [2970.0, 110.0], "p2": [2970.0, 850.0], "p3": [3155.0, 850.0], "p4": [3155.0, 110.0] },
        { "p1": [2970.0, 672.5], "p2": [2970.0, 850.0], "p3": [4512.5, 850.0], "p4": [4512.5, 672.5] },
        { "p1": [4322.0, 110.0], "p2": [4322.0, 850.0], "p3": [4512.5, 850.0], "p4": [4512.5, 110.0] },
        { "p1": [1945.0, 0.0], "p2": [1945.0, 695.0], "p3": [2670.0, 695.0], "p4": [2670.0, 0.0] },
        { "p1": [1352.0, 110.0], "p2": [1352.0, 425.0], "p3": [3155.0, 425.0], "p4": [3155.0, 110.0] },
        { "p1": [1802.5, 0.0], "p2": [1802.5, 110.0], "p3": [2800.0, 110.0], "p4": [2800.0, 0.0] }],
 "intersect": [
        { "p": [4417.25, 204.75] },
        { "p": [2307.5, 55.0] },
        { "p": [1451.0, 267.5] },
        { "p": [3062.5, 267.5] },
        { "p": [92.5, 204.75] },
        { "p": [3062.5, 204.75] },
        { "p": [2307.5, 267.5] },
        { "p": [1451.0, 204.75] },
        { "p": [92.5, 761.25] },
        { "p": [3062.5, 761.25] },
        { "p": [1451.0, 761.25] },
        { "p": [4417.25, 761.25] }],
 "max_neigh": 7
}
```

**Listing A.6:** "small office" scenario json file 1

The json file of the "laboratory" scenario is shown in Listing A.7.

```
{
 "queries": [
        { "type": "pscs", "tau": 280, "n": 10 },
        { "type": "pfail", "tau": 400, "n": 1 },
        { "type": "sim", "tau": 400, "n": 1 },
        { "type": "eftg", "tau": 400, "n": -1 },
        { "type": "echg", "tau": 400, "n": -1 }],
"humans": [
        { "name": "h_1", "h_id": 1, "v": 40.0, "ptrn": "FOLLOWER", "p_f
            ": "y/s", "p_fw": "l", "start": [2500.0, 200.0], "dest":
            [1500.0, 400.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_2", "h_id": 2, "v": 40.0, "ptrn": "RECIPIENT", "p_f
            ": "e/h", "p_fw": "d", "start": [4000.0, 150.0], "dest":
            [3000.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_3", "h_id": 3, "v": 40.0, "ptrn": "LEADER", "p_f":
            "e/h", "p_fw": "d", "start": [4100.0, 150.0], "dest":
            [3500.0, 150.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_4", "h_id": 4, "v": 40.0, "ptrn": "LEADER", "p_f":
            "y/s", "p_fw": "d", "start": [800.0, 700.0], "dest": [900.0,
            700.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_5", "h_id": 5, "v": 40.0, "ptrn": "FOLLOWER", "p_f
            ": "y/s", "p_fw": "d", "start": [1900.0, 200.0], "dest":
            [3000.0, 150.0], "dext": -1, "same_as": -1, "path": -1 }],
"robots": [
        { "name": "r_1", "r_id": 1, "v": 60.000004, "a": 60.000004, "
            start": [2500.0, 400.0], "chg": 12.4 }],
"areas": [
        { "p1": [0.0, 110.0], "p2": [0.0, 299.5], "p3": [1550.0, 299.5],
             "p4": [1550.0, 110.0] },
        { "p1": [0.0, 110.0], "p2": [0.0, 850.0], "p3": [185.0, 850.0],
            "p4": [185.0, 110.0] },
        { "p1": [0.0, 672.5], "p2": [0.0, 850.0], "p3": [1550.0, 850.0],
             "p4": [1550.0, 672.5] },
        { "p1": [1352.0, 110.0], "p2": [1352.0, 850.0], "p3": [1550.0,
            850.0], "p4": [1550.0, 110.0] },
        { "p1": [2970.0, 110.0], "p2": [2970.0, 299.5], "p3": [4512.5,
            299.5], "p4": [4512.5, 110.0] },
        { "p1": [2970.0, 110.0], "p2": [2970.0, 850.0], "p3": [3155.0,
            850.0], "p4": [3155.0, 110.0] },
        { "p1": [2970.0, 672.5], "p2": [2970.0, 850.0], "p3": [4512.5,
            850.0], "p4": [4512.5, 672.5] },
        { "p1": [4322.0, 110.0], "p2": [4322.0, 850.0], "p3": [4512.5,
            850.0], "p4": [4512.5, 110.0] },
        { "p1": [1945.0, 0.0], "p2": [1945.0, 695.0], "p3": [2670.0,
            695.0], "p4": [2670.0, 0.0] },
        { "p1": [1352.0, 110.0], "p2": [1352.0, 425.0], "p3": [3155.0,
            425.0], "p4": [3155.0, 110.0] },
        { "p1": [1802.5, 0.0], "p2": [1802.5, 110.0], "p3": [2800.0,
            110.0], "p4": [2800.0, 0.0] }],
"intersect": [
        { "p": [4417.25, 204.75] },
        { "p": [2307.5, 55.0] },
        { "p": [1451.0, 267.5] },
        { "p": [3062.5, 267.5] },
        { "p": [92.5, 204.75] },
        { "p": [3062.5, 204.75] },
        { "p": [2307.5, 267.5] },
        { "p": [1451.0, 204.75] },
        { "p": [92.5, 761.25] },
        { "p": [3062.5, 761.25] },
        { "p": [1451.0, 761.25] },
        { "p": [4417.25, 761.25] }],
 "max_neigh": 7
}
```

**Listing A.7:** "laboratory" scenario json file 2

The json file of the "large office" scenario is shown in Listing A.8.

```
{
 "queries": [
        { "type": "pscs", "tau": 1300, "n": 10 },
        { "type": "sim", "tau": 1300, "n": 1 }],
 "humans": [
        { "name": "h_1", "h_id": 1, "v": 20.0, "ptrn": "FOLLOWER", "p_f
           ": "y/s", "p_fw": "l", "start": [1500.0, 200.0], "dest":
           [1500.0, 700.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_2", "h_id": 2, "v": 20.0, "ptrn": "LEADER", "p_f":
           "e/s", "p_fw": "d", "start": [1400.0, 400.0], "dest":
           [1400.0, 500.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_3", "h_id": 3, "v": 20.0, "ptrn": "FOLLOWER", "p_f
           ": "e/s", "p_fw": "d", "start": [1400.0, 200.0], "dest":
           [2500.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_4", "h_id": 4, "v": 20.0, "ptrn": "RECIPIENT", "p_f
           ": "y/h", "p_fw": "d", "start": [200.0, 150.0], "dest":
           [1500.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_5", "h_id": 5, "v": 20.0, "ptrn": "LEADER", "p_f":
           "e/h", "p_fw": "d", "start": [4000.0, 200.0], "dest":
           [3000.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_6", "h_id": 6, "v": 20.0, "ptrn": "LEADER", "p_f":
           "e/h", "p_fw": "d", "start": [3000.0, 200.0], "dest":
           [4000.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_7", "h_id": 7, "v": 20.0, "ptrn": "LEADER", "p_f":
           "y/s", "p_fw": "d", "start": [1000.0, 700.0], "dest":
           [1100.0, 700.0], "dext": -1, "same_as": -1, "path": -1 },
        { "name": "h_8", "h_id": 8, "v": 20.0, "ptrn": "FOLLOWER", "p_f
        ": "y/s", "p_fw": "d", "start": [1100.0, 200.0], "dest":
        [500.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
    { "name": "h_9", "h_id": 9, "v": 20.0, "ptrn": "LEADER", "p_f":
        "e/s", "p_fw": "d", "start": [1500.0, 300.0], "dest":
        [2500.0, 200.0], "dext": -1, "same_as": -1, "path": -1 },
    { "name": "h_10", "h_id": 10, "v": 20.0, "ptrn": "FOLLOWER", "
        p_f": "e/s", "p_fw": "d", "start": [2500.0, 200.0], "dest":
        [3500.0, 200.0], "dext": -1, "same_as": -1, "path": -1 }],
"robots": [
    { "name": "r_1", "r_id": 1, "v": 20.0, "a": 20.0, "start":
        [1500.0, 400.0], "chg": 12.4 }],
"areas": [
    { "p1": [0.0, 110.0], "p2": [0.0, 299.5], "p3": [1550.0, 299.5],
         "p4": [1550.0, 110.0] },
    { "p1": [0.0, 110.0], "p2": [0.0, 850.0], "p3": [185.0, 850.0],
        "p4": [185.0, 110.0] },
    { "p1": [0.0, 672.5], "p2": [0.0, 850.0], "p3": [1550.0, 850.0],
         "p4": [1550.0, 672.5] },
    { "p1": [1352.0, 110.0], "p2": [1352.0, 850.0], "p3": [1550.0,
        850.0], "p4": [1550.0, 110.0] },
    { "p1": [2970.0, 110.0], "p2": [2970.0, 299.5], "p3": [4512.5,
        299.5], "p4": [4512.5, 110.0] },
    { "p1": [2970.0, 110.0], "p2": [2970.0, 850.0], "p3": [3155.0,
        850.0], "p4": [3155.0, 110.0] },
    { "p1": [2970.0, 672.5], "p2": [2970.0, 850.0], "p3": [4512.5,
        850.0], "p4": [4512.5, 672.5] },
    { "p1": [4322.0, 110.0], "p2": [4322.0, 850.0], "p3": [4512.5,
        850.0], "p4": [4512.5, 110.0] },
    { "p1": [1945.0, 0.0], "p2": [1945.0, 695.0], "p3": [2670.0,
        695.0], "p4": [2670.0, 0.0] },
    { "p1": [1352.0, 110.0], "p2": [1352.0, 425.0], "p3": [3155.0,
        425.0], "p4": [3155.0, 110.0] },
    { "p1": [1802.5, 0.0], "p2": [1802.5, 110.0], "p3": [2800.0,
        110.0], "p4": [2800.0, 0.0] }],
"intersect": [
    { "p": [4417.25, 204.75] },
    { "p": [2307.5, 55.0] },
    { "p": [1451.0, 267.5] },
    { "p": [3062.5, 267.5] },
    { "p": [92.5, 204.75] },
    { "p": [3062.5, 204.75] },
    { "p": [2307.5, 267.5] },
    { "p": [1451.0, 204.75] },
    { "p": [92.5, 761.25] },
        { "p": [3062.5, 761.25] },
        { "p": [1451.0, 761.25] },
        { "p": [4417.25, 761.25] }],
 "max_neigh": 7
}
```

**Listing A.8:** "large office" scenario json file 3

# List of Figures

3.1 overview of the phases of the framework. . . . . . . . . . . . . . . . . . . . . 9
3.2 plugin workflow overview. . . . . . . . . . . . . . . . . . . . . . . . . . . 10

4.1 picture scenario. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.2 UML input framework. . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.3 UML DSL elements. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

5.1 Plugin interaction diagram. . . . . . . . . . . . . . . . . . . . . . . . . . 28
5.2 text editor provided with the Eclipse IDE plugin. . . . . . . . . . . . . . . 31
5.3 example point of intersection. . . . . . . . . . . . . . . . . . . . . . . . . 35

6.1 results regarding user requests of the "room" scenario. . . . . . . . . . . . 39
6.2 results regarding user requests of the "building" scenario json file 1. . . . . 41
6.3 results regarding user requests of the "building" scenario json file 2. . . . . 42
6.4 results regarding user requests of the "building" scenario json file 3. . . . . 42
6.5 results regarding user requests of the "small office" scenario. . . . . . . . . 46
6.6 results regarding user requests of the "laboratory" scenario. . . . . . . . . . 46
6.7 results regarding user requests of the "large office" scenario. . . . . . . . . 46

# List of Listings

4.1 scenario json file . . . 16
4.2 scenario DSL file . . . 21
4.3 EBNF of the DSL . . . 25
5.1 Xtext model of the DSL grammar . . . 28
6.1 "room" scenario DSL file . . . 38
6.2 "room" scenario json file . . . 38
6.3 snippet "building" scenario DSL file . . . 40
6.4 snippet "building" scenario json file 1 . . . 41
6.5 snippet "building" scenario json file 2 . . . 41
6.6 snippet "building" scenario json file 3 . . . 42
6.7 snippet test3 DSL file . . . 44
6.8 snippet "small office" scenario json file 1 . . . 45
6.9 snippet "laboratory" scenario json file 2 . . . 45
6.10 snippet "large office" scenario json file 3 . . . 46
A.1 "building" scenario DSL file . . . 53
A.2 "building" scenario json file 1 . . . 55
A.3 "building" scenario json file 2 . . . 57
A.4 "building" scenario json file 3 . . . 57
A.5 test3 DSL file . . . 58
A.6 "small office" scenario json file 1 . . . 63
A.7 "laboratory" scenario json file 2 . . . 64
A.8 "large office" scenario json file 3 . . . 66

# Acknowledgements

At the end of the work, my thanks go to Prof. Matteo Rossi and Livia Lestingi for having guided me with firmness and passion in this incredible journey and for their great availability.

Special thanks go to my family for always being close to me and supporting me.

Finally, I thank all my friends for being amazing companions and for all the great memories you have given me.