Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

European Master in Software Engineering

Master Thesis

Ontology Generator and Backend Environment for New World of Agents Game

Author: Adrián Sánchez Rodero

June, 2025

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis

European Master in Software Engineering

Title: Ontology Generator and Backend Environment for New World of Agents Game
June, 2025

Author: Adrián Sánchez Rodero

Supervisor:

Ricardo Imbert Paredes

Ph.D. in Computer Science

Universidad Politécnica de Madrid

Computer Languages and Systems and Software Engineering

Universidad Politécnica de Madrid

Co-supervisor:

José María Barambones Ramírez

Ph.D. in Computer Science

Universidad Politécnica de Madrid

Computer Languages and Systems and Software Engineering

Universidad Politécnica de Madrid

Abstract

This master thesis is part of the Agent-based Software Development course of the European Master in Software Engineering which explains the world of agents with a practical project. The purpose of this master thesis is to renew the practical project of that course and to simplify the students' work. The project consists in the development of a strategy game in which teams implement a platform and different types of units using agents. Teams prepare a strategy that their units should use in order to defeat other teams' units.

One of the deliverables of this work is an ontology generator. An ontology is the language that agents use to communicate between them. The created Ontology Generator is a flexible program that offers two ways to generate the ontology: a user interface or using the terminal. For the user interface, first a design of the different screens has been created with a prototype exemplifying several flows, and then the necessary source code has been developed in Java. This reduces the time that students dedicate to the implementation of the ontology, and therefore the teams will have more time to prepare their strategies.

Moreover, this master thesis provides to the students a basic structure for the source code of the practical project. This structure has some useful implementations. First, it has an http client to communicate with the frontend part of the game through an API also defined in this thesis. Secondly, it has a base agent implementation with the capacity of sending all types of messages between agents. This functionality is essential for communication between units of a team and the platform to perform actions such as moving or attacking, and communications of units between them to carry out the prepared strategy.

In addition, the structure contains a logger functionality to print messages in the output console of the program. This logger is incorpored into the BaseAgent and the http client. In the case of the agents, this helps in the integration process where there is a high probability of errors (team agents are executed on other team platforms to test the communications with the platform).

Finally, in order to use the deliverables products of this master thesis, two manuals have been written: Ontology Generator: User Manual and Team - Platform: Developer Manual.

Resumen

Este trabajo de fin de máster forma parte de la asignatura Agent-based Software Development del European Master in Software Engineering que explica el mundo de los agentes con un proyecto práctico. El objetivo de este trabajo de fin de máster es renovar el proyecto práctico de dicho curso y simplificar el trabajo de los alumnos. El proyecto es un juego de estrategia en el que los equipos implementan una plataforma y diferentes tipos de unidades utilizando agentes. Los equipos preparan una estrategia que sus unidades deben utilizar para derrotar a las unidades de otros equipos.

Uno de los resultados de este trabajo es un generador de ontologías. Una ontología es el lenguaje que utilizan los agentes para comunicarse entre sí. El generador de ontologías creado es un programa flexible que ofrece dos formas de generar la ontología: una interfaz de usuario o utilizando la terminal. Para la interfaz de usuario, primero se ha creado un diseño de las diferentes pantallas con un prototipo que ejemplifica varios flujos, y después se ha desarrollado el código fuente necesario en Java. Esto reduce el tiempo que los estudiantes dedican a la implementación de la ontología y, por tanto, los equipos dispondrán de más tiempo para preparar sus estrategias.

Además, este trabajo de fin de máster proporciona a los estudiantes una estructura básica para el código fuente del proyecto práctico. Esta estructura tiene algunas implementaciones útiles. En primer lugar, dispone de un cliente http para comunicarse con la parte frontend del juego a través de una API también definida en este trabajo. En segundo lugar, dispone de una implementación de un agente genérico con capacidad para enviar todo tipo de mensajes entre agentes. Esta funcionalidad es esencial para la comunicación entre las unidades de un equipo y la plataforma para realizar acciones como moverse o atacar, y para las comunicaciones de las unidades entre sí para llevar a cabo la estrategia preparada.

Además, la estructura contiene una funcionalidad para imprimir mensajes en la consola de salida del programa. Esta funcionalidad está incorporado en el agente genérico y el cliente http. En el caso de los agentes, esto ayuda en el proceso de integración donde hay una alta probabilidad de errores (los agentes específicos de cada equipo se ejecutan en las plataformas de otros equipos para probar las comunicaciones con la plataforma).

Por último, para poder utilizar los productos entregables de este trabajo de fin de máster, se han redactado dos manuales: Generador de Ontologías: Manual de Usuario y Equipo - Plataforma: Manual del Desarrollador.

Table of Contents

1 Introduction

1.1. Context

This master thesis is related to the Agent-based Software Development subject of the European Master in Software Engineering. This subject explains the agents world interspersing practical and theoretical parts through a project. The project consists in the development of a strategy game in which the students are divided into different teams that compete between them, implementing their own agents. Each team has to develop platform agents to control the game (a common part for all teams) and team agents to implement the strategy they consider to win the game.

The origin of this master thesis comes from the fact that the teachers of the Agent-based Software Development subject (supervisor and co-supervisor of this work) wanted to renew the game they were using in the subject and simplify the students' work.

This work tries to make it easier for students the development of those agents taking into account the basis of the new game. We provide them with the implementation of the most costly and tedious parts, so they can focus on their strategies.

1.2. Objectives of this work

The main objective of this master thesis is to help students of the Agent-based Software Development course develop the project of the subject. This support should allow them to focus more on the definition of their strategies and to reduce the time they spend on basic programming aspects. This objective is divided into four subgoals described in the next paragraphs.

In previous years, the ontology for the communication between agents was developed for a team each sprint, but because it was a part that all teams needed to perform the sprint tasks, the team in charge had to develop it in the first days of the sprint. This meant that the team in charge had the added pressure of having to implement the ontology quickly and without errors to avoid having to make changes in the middle of the sprint that could cause problems in the implementations. For this reason, one of the objectives of this work is to create an ontology generator so that each team, starting from the definition established in the classes of the subject, can generate the necessary ontology implementation for the development of the sprint tasks.

The connection between the agents (backend part) and the visual interface (fronted part) is another important aspect of the project that is carried out through an API (Application Programming Interface). Implementing API calls is a process that every master's student has probably gone through in the past. It is a basic implementation that takes time for the teams. To reduce that time by allowing students to spend it on strategy preparation or other more complex tasks, this

work should incorporate that functionality into the project structure provided to students.

Related to the first goal, another aspect that students spend a lot of time on is the implementation of communication between game agents. It is understood that this is a fundamental point in the Agent-based Software Development course, so the students will have to spend some time on it. But to make this process easier, one objective of this work is to define the methods to send messages with each of type of performative, and it will be up to the students to decide which performative to use and with which content according to what they define in the classes.

Finally, one of the biggest headaches for students is the location of errors, especially in the integration processes of the different team agents on the platform developed by other teams. This integration involves a large number of agents, increasing the probability of errors, so it is crucial to know whom each agent belongs. For this reason, another of the goals of this project is to implement a tool that allows to distinguish the logs dumped by each element of the game.

1.3. Motivation

In the second semester of the master's degree, I took the Agent-based Software Development course, which aroused a great interest in me about the study of the agents world. During the course we implemented some agents to control and compete in a strategy game where agents explore and collect resources while simultaneously creating new units and constructing settlements.

When the teachers offered me this master thesis, I thought it was a very interesting project because in addition to go deeper into the agents world, my work would help future students to better understand the Agent-based Software Development subject. I was also motivated by the fact that if my master thesis was part of the project used in the course, it would be useful in the future and would not simply be a work to finish the master's degree.

1.4. Structure of the document

This document details all relevant information about the SCRUM development process [1] of this master thesis that has had the Ontology Generator and the Team - Platform Environment as main results.

First, this chapter gives a brief introduction to the subject of the work, indicating the context in which it has been developed, the motivation behind it, and an explanation of the objectives of the thesis.

The second chapter (2 Game Rules) gives an overview of the characteristics and rules of the new game defined in this master thesis and José Rodríguez de Santiago master thesis [2] for the Agent-based Software Development course.

Third, the Development Methodology chapter (4) goes into more depth about the development process followed in this master thesis. This chapter is divided into several sections. First, the project kick-off section describes the main epics of the project with some key points. After the kick-off, each sprint is organized

into its own section that explains the user stories planned with a description, acceptance criteria, and assigned story points.

In the following chapters, the products resulting from the SCRUM process are described in detail.

On the one hand, the Ontology Generator chapter (5) includes a class diagram illustrating the main components and their relationships, an explanation of the source code structure, a description of the specification file used in the generation process, a presentation of the user interface, and a discussion of the response codes.

In addition, the Ontology Generator has a second chapter, 6 Ontology Generator - Black Box, which explains a test plan designed for the generator using the Black Box testing technique.

On the other hand, the Team – Platform Environment chapter (7) presents the source code structure and introduces the AgentsApplication, which contains setup information, the BaseAgent, which provides the code shared by all agents, the GameHttpClient, which manages the connection with the game's frontend interface, and the log standardization feature, designed to facilitate debugging across both team and platform agent operations.

Chapter 8, Results, provides an overview of the outcomes achieved throughout the development of this work. The chapter includes a dedicated section for each sprint, presenting the corresponding burndown chart along with a summary of completed, in-progress, and pending user stories. Each sprint section also has a description of the deliverables produced.

The next chapter (9 Discussion) has a global analysis of the results of the different sprints. And chapter 10, Conclusions, presents the general conclusions drawn from the work carried out based on the analysis of the sprints and the products obtained.

Finally, this document includes a bibliography with references and links to the documentation of the various technological tools used. In addition, three appendices are included at the end of the document: a user manual for the Ontology Generator (12.1), the API specification for communication with the frontend (12.2), and a manual for the Team – Platform Environment (12.3) intended for developers who will use this environment to implement the agents game.

2 Game Rules

The following chapter was written with the collaboration of José Rodríguez de Santiago whose master thesis is related to the NewWoA frontend part [2].

2.1. Motivation

In the *New World of Agents* (NewWoA) game, teams should create different types of units (2.6) during the game phases (2.4) with the objective of killing other team's units and being the only team alive on the game board (2.7).

Team's units should move around the board (2.4) to escape from other unit´s attacks and, at the same time, attack other units.

2.2. Map

The map in which NewWoA takes place has very few restrictions. It is a bidimensional flat map without geographical barriers that allows the free movement of units around the world. It starts at (1,1) in the lower left corner and expands to the right top corner one unit at a time. The size of the map is determined by the following formula:

```
Width = MAX(4, ROUND(4 + SQRT(totalUnits) * 1.5))
Height = MAX(4, ROUND(4 + SQRT(totalUnits) * 1.2))
```

Where totalUnits is calculated as follows:

```
totalUnits = numberOfPlayers * 12;
```

Considering 12 as the maximum units a player can have (stablished by the rules, see registration phase in section 2.4).

2.3. Starting distribution

The starting distribution will be set by the system taking into account the number of teams and the map size (See recommendations section 2.8).

2.4. Phases

The game is composed by three phases.

1. **Registration phase**: Teams must choose and register their units on the board in the *duration_registration* time.

   In this phase, each unit should be identified with a unique name.

At the **beginning**, each team will have **12 deployment points** to spend on troops, which must be spent according to the deployment cost of each unit specified in the units section (2.6).

Once the troops have been chosen, they will automatically be placed in each team's designated spawn zones (2.3).

If a team **has not chosen any units**, it **cannot participate in the game** either. However, it is not necessary to spend the 12 deployment points because the **starting score** of each team is equal to the **unspent deployment points**.

2. **Match phase**: During the game phase the teams must give instructions to each of their units, choosing between:
   - **ATTACK**: The unit will attack an enemy unit selected by the team based on the range of the attacking unit. Then, an amount of health points equal to the attacking unit's damage will be deducted from the attacked unit.

     If the resulting health points are less than or equal to 0, the attacked unit is removed from the board, and the attacking unit's team gains an amount of points equal to the deployment cost of the removed unit.
   - **MOVEMENT**: The unit will move to a square on the board adjacent to its current location and selected by the team.

   Each selected action will **fatigue the unit** that performed it, blocking it **2 seconds before** that unit can perform **another action**.

   If **30 seconds pass without** any unit of a team taking **any action**, then the team will be **eliminated**, and its final score will be reduced to **half the score** it had before this happened.

   If **2 minutes pass without** any unit of a team having **eliminated another unit**, its **lowest cost unit** will be eliminated.

   This phase will end when only one team has units on the board or when *duration_match* has passed, when one of these two occurs, the team(s) still standing will be awarded an amount of points equal to the deployment points of the units still alive multiplied by 5.
3. **Results phase**: The board shows the results at the end of the game.

2.5. Activity visualization

Concurrent activity occurs during the development of a NewWoA match. Every team is aware of everything that occurs during the whole game, including:

- Content of every cell in the map.
- Movement of all units.
- Creation of team's new units.
- Attacks carried out by all units.
- Killing of all units on the map.

- Health loss of all units on the map.

There is no activity contemplated in the game which is not known by all the teams.

2.6. Units

In NewWoA there are 4 types of Units: Warrior, Archer, Healer and General. Each of them have 4 stats:

- **Deployment Points:** The cost of deploying that unit on game start.
- **Health:** The points that a unit can receive from attacks before being killed.
- **Damage:** The points that each attack of the unit takes from the attacked unit health.
- **Reach:** The number of cells from which a unit can attack other units.

The stats for aech unit are:

- **Warrior**

  Deployment: 1

  Health: 100

  Damage: 50

  Reach: 1

- **Archer**

  Deployment: 2

  Health: 60

  Damage: 90

  Reach: 2

- **Healer**

  Deployment: 3

  Health: 60

  Damage: 50

  Reach: 1

  Special ability: Adjacent allied units gain 50 more health points.

- **General**

  Deployment: 3

  Health: 60

  Attack: 50

  Range: 1

  Special ability: Adjacent allied units get 10 more damage points.

2.7. Results

The results will consist of a comparison between the points of all the teams, with the team with the most points coming first, and the rest of the teams being ranked according to the same criteria. In the event of a tie in points, the team that has been eliminated later will be in the first place. If they have been eliminated at the same time as well, the winner will be selected randomly between both. *That´s how life works*.

2.8. Development recommendations

In order to achieve the best results with the proposed game, it is recommended to follow the next indications:

- **Starting positions of the units**: The suggested starting points are:

```
Team 1: 1,1
Team 2: X, Y
Team 3: X,1
Team 4: 1, Y
Team 5: X/2,1
Team6: 1, Y/2
```

  Being X the Width and Y the Height of the map.

- **Block unit on attack**: When a unit is being attacked, fix its position and do not allow any movement in order to have a clear visualisation of the attack.
- **Movement invincibility**: When a unit is moving, consider attacks targeted at it as failures and do not send them to the interface.
- **Unit names**: To improve the debugging of the program, the names of the units should start with the prefix "TeamX"where X is the number of the team to recognize easily the team in charge of that unit.
- **Development order - units**: The development of some units is complex and can lead to some problems. Because of that, it is recommended to start with the warrior. After that, advance to the archer in order to better understand and control the range of attacks. And finally, implement general and healer. A very basic version can be obtained by only implementing warrior and archer, but general and healer are also available in the game, so if wanted, they can be added, bearing in mind that it can be a rather difficult implementation.
- **Development order - game characteristics**: The same happens with some game characteristics. It is recommended to forget about the inactivity constraints, they are just there to weaken some possible non-fighting strategies, but the development can also be difficult. So, it is better to develop everything without these features and in the end, if you feel confident, to add them.

3 State of the Art: Agent based Software Development with JADE

3.1. Description

Agent-based Software Development (ABSD) is an approach that models software systems as a collection of autonomous and interactive agents. These agents are designed to operate independently, making decisions and collaborating to achieve complex objectives.

JADE (Java Agent Development Framework) [3] is an open source platform for developing multi-agent systems in Java. Provides a robust environment for building interoperable and distributed agent-based applications.

3.2. Architecture

JADE's architecture is designed to facilitate the development and execution of agent-based applications. The main components of the framework are [4]:

- **Agent**: The fundamental unit in JADE, each agent is an instance of the Agent class. Developers create custom agents by extending this class and defining specific behaviors.
- **Behaviors**: Agents perform tasks through behaviors, which are modular units of code that define specific actions.
- **Containers and Platform**: A JADE platform consists of multiple containers, each capable of hosting several agents.
- **Agent Management System (AMS)**: This special agent oversees the overall management of the platform, including the creation and deletion of agents.
- **Directory Facilitator (DF)**: Acts as a yellow pages service, allowing agents to register their services and discover others.

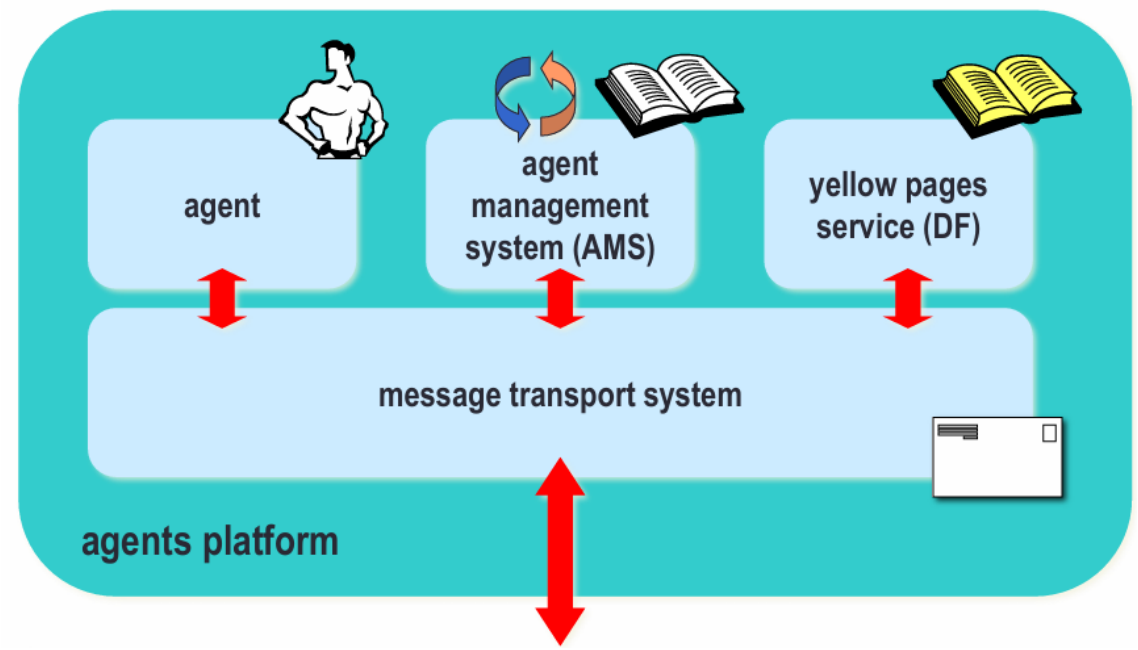


Figure 3.1: JADE architecture design

3.3. Applications and Use Cases

JADE has been successfully applied in various domains, including [5]:

- **Smart Homes and IoT (Internet of Things)**: Agents coordinate devices and services within smart environments.
- **Autonomous Vehicles**: JADE facilitates communication and coordination between vehicles in intelligent transportation systems.
- **Supply Chain Management**: Agents manage logistics, inventory, and scheduling in complex supply chains.
- **Healthcare Systems**: JADE supports the development of systems for patient monitoring and resource allocation.

3.4. Future Directions

The evolution of JADE is focused on several key areas [6]:

- **Integration with Modern Technologies**: Enhancing compatibility with emerging technologies, such as cloud computing and edge devices, to expand JADE's applicability.
- **Improved Tooling**: Developing advanced debugging, testing, and monitoring tools to streamline the development process.
- **Community Engagement**: Fostering a vibrant community to contribute to the framework's growth and support.

3.5. Relevance for this work

Agent-based software development offers a powerful approach to build intelligent behaviors in strategy games. Unlike traditional game AI, multi-agent systems enable dynamic, decentralized decision-making where each agent (different type of units) acts based on local objectives and communication with others.

JADE has been applied in academic projects to model adaptive AI in military simulations, resource management, and cooperative game strategies. Its modular design and support for coordinated agent behavior make it a strong tool for prototyping strategy games in educational environments.

4 Development Methodology

This chapter describes the development methodology used in this Master's Thesis. To carry out this project, I followed a SCRUM methodology [1] with several sprints. The first section enumerates the epics of the work (4.1). Then each sprint is explained in detail in its corresponding section, indicating the user stories planned, and the stats of the sprint (number of epics addressed, total number of stories planned and total number of story points).

4.1. Kick-off of the project

At the beginning of the project, my tutors and I had a kick-off session to establish the basis of the work. In that session, we defined the epics along with a list of general tasks that would be developed in the different sprints.

We came up with six main areas on which to focus the work. They are listed below with their associated tasks.

1. **Ontology generator**
   - 1.1. Generate the ontology from a specification in a txt file.
     - 1.1.1. Generate concepts, actions and predicates.
     - 1.1.2. Generate ontology vocabulary file.
     - 1.1.3. Generate ontology file.
   - 1.2. Implement a basic interface to build the ontology from user inputs.
     - 1.2.1. Design and implement the interface.
     - 1.2.2. Incorporate ontology generation logic to the interface.
2. **API**
   - 2.1. Definition of the API (endpoints, request bodies, responses...).
   - 2.2. Implementation of the API client.
   - 2.3. Management of possible errors.
   - 2.4. Testing of the API client.
     - 2.4.1. Happy paths.
     - 2.4.2. Error cases.
3. **Environment definition**
   - 3.1. Structure definition (organization of team and platform agents).
   - 3.2. Implementation of a BaseAgent with builders of messages of each type of protocol.
   - 3.3. Creation of example agents.
   - 3.4. Testing of the environment.

3.4.1. Communications between team and platform agents.

3.4.2. Messages with each type of protocol.

4. **Documentation**
   4.1. User Manual: Ontology Generator.
   4.2. Developer Manual: Definition of environment (Team - Platform).
   4.3. Documentation of each sprint.
5. **Log standardization**
   5.1. Definition of log format (info, warn, error).
   5.2. Implementation of log generator.
   5.3. Testing of the log generator.
6. **Testing of the agents**

4.2. Sprint 1

This section describes the work performed during Sprint 1. The objective of this first sprint was to start with the Ontology Generator.

4.2.1. Sprint Backlog

To achieve the objective of this sprint, the following user stories were planned.

4.2.1.1. US 1.1 Set up of the ontology project

Epic: Ontology generator.

Description: AS a developer, I WANT to create the ontology project with the basic structure IN ORDER TO store the generator.

Acceptance Criteria

- GIVEN the ontology project WHEN running the application, THEN the project should run without errors.

Story Points: 1

4.2.1.2. US 1.2 Define the specification of the ontology

Epic: Ontology generator.

Description: AS a developer, I WANT to define the specification of the ontology in a txt file IN ORDER TO specify the contents of the ontology.

Acceptance Criteria

- GIVEN the ontology specification txt file WHEN reading the file, THEN the name of the ontology should be easily identified.

- GIVEN the ontology specification txt file WHEN reading the file, THEN the concepts of the ontology should be easily identified.
- GIVEN the ontology specification txt file WHEN reading the file, THEN the actions of the ontology should be easily identified.
- GIVEN the ontology specification txt file WHEN reading the file, THEN the predicates of the ontology should be easily identified.

Story Points: 3

4.2.1.3. US 1.3 Parse the ontology specification file

Epic: Ontology generator.

Description: AS a developer, I WANT to parse the ontology specification txt file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN the ontology specification txt file WHEN parsing the file, THEN the name of the ontology should have been read.
- GIVEN the ontology specification txt file WHEN parsing the file, THEN the concepts of the ontology should have been read.
- GIVEN the ontology specification txt file WHEN parsing the file, THEN the actions of the ontology should have been read.
- GIVEN the ontology specification txt file WHEN parsing the file, THEN the predicates of the ontology should have been read.
- GIVEN the ontology specification txt file WHEN parsing the file, THEN the information of the ontology should be easily accessible.

Story Points: 5

4.2.1.4. US 1.4 Generate a concept

Epic: Ontology generator.

Description: AS a developer, I WANT to generate a concept from the ontology specification file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN the specification of a concept in the ontology specification file WHEN processing the file, THEN a JAVA file with the name of the specified concept should be created.
- GIVEN the specification of a concept in the ontology specification file WHEN processing the file, THEN the generated concept class should have the attributes indicated in the specification.
- GIVEN the specification of a concept in the ontology specification file WHEN processing the file, THEN the generated concept should compile.
- GIVEN the specification of some concepts in the ontology specification file WHEN processing the file, THEN the indicated concepts should be created.

Story Points: 5

4.2.1.5. US 1.5 Generate an action

Epic: Ontology generator.

Description: AS a developer, I WANT to generate an action from the ontology specification file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN the specification of an action in the ontology specification file WHEN processing the file, THEN a JAVA file with the name of the specified action should be created.
- GIVEN the specification of an action in the ontology specification file WHEN processing the file, THEN the generated action class should have the attributes indicated in the specification.
- GIVEN the specification of an action in the ontology specification file WHEN processing the file, THEN the generated action should compile.
- GIVEN the specification of some actions in the ontology specification file WHEN processing the file, THEN the indicated actions should be created.

Story Points: 3

4.2.1.6. US 1.6 Generate a predicate

Epic: Ontology generator.

Description: AS a developer, I WANT to generate a predicate from the ontology specification file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN the specification of a predicate in the ontology specification file WHEN processing the file, THEN a JAVA file with the name of the specified predicate should be created.
- GIVEN the specification of a predicate in the ontology specification file WHEN processing the file, THEN the generated predicate class should have the attributes indicated in the specification.
- GIVEN the specification of a predicate in the ontology specification file WHEN processing the file, THEN the generated predicate should compile.
- GIVEN the specification of some predicates in the ontology specification file WHEN processing the file, THEN the indicated predicates should be created.

Story Points: 3

4.2.1.7. US 1.7 Generate the ontology vocabulary file

Epic: Ontology generator.

Description: AS a developer, I WANT to generate the ontology vocabulary file from the ontology specification file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN an ontology specification file WHEN processing the file, THEN a JAVA file should be created.
- GIVEN an ontology specification file WHEN processing the file, THEN the JAVA file generated should have as name the name of the ontology concatenated with the word Vocabulary.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology vocabulary should contain the name of the ontology.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology vocabulary should contain all the names and attributes of the concepts.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology vocabulary should contain all the names and attributes of the actions.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology vocabulary should contain all the names and attributes of the predicates.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology vocabulary should compile.

Story Points: 3

4.2.1.8. US 1.8 Generate the ontology file

Epic: Ontology generator.

Description: AS a developer, I WANT to generate the ontology file from the ontology specification file IN ORDER TO build the ontology.

Acceptance Criteria

- GIVEN an ontology specification file WHEN processing the file, THEN a JAVA file should be created.
- GIVEN an ontology specification file WHEN processing the file, THEN the JAVA file generated should have as name the name of the ontology.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology file should add all the concepts.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology file should add all the actions.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology file should add all the predicates.
- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology file should use the vocabulary from the vocabulary ontology file.

- GIVEN an ontology specification file WHEN processing the file, THEN the generated ontology should compile.

Story Points: 8

4.2.1.9. US 1.9 Design a basic user interface for the Ontology Generator

Epic: Ontology generator.

Description: AS a developer, I WANT to design a basic user interface for the ontology IN ORDER TO build the ontology from user inputs.

Acceptance Criteria

- GIVEN a design of a basic user interface for the ontology WHEN the user wants to create a new concept, THEN the interface should have the necessary inputs.
- GIVEN a design of a basic user interface for the ontology WHEN the user wants to create a new action, THEN the interface should have the necessary inputs.
- GIVEN a design of a basic user interface for the ontology WHEN the user wants to create a new predicate, THEN the interface should have the necessary inputs.
- GIVEN a design of a basic user interface for the ontology WHEN the user wants to generate a new ontology from a specification file, THEN the interface should have the necessary buttons.

Story Points: 3

4.2.1.10. US 1.10 Generate the ontology from a specification file using the interface

Epic: Ontology generator.

Description: AS a developer, I WANT to generate the ontology from a specification file IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate the ontology from a specification file, THEN the specified ontology should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate the ontology from a specification file, THEN the generated ontology should have all the concepts, actions, and predicates indicated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate the ontology from a specification file, THEN the associated ontology vocabulary file should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate the ontology from a specification file, THEN the generated ontology should compile.

Story Points: 1

4.2.1.11. US 1.11 Generate a concept with the user interface

Epic: Ontology generator.

Description: AS a developer, I WANT to generate a concept from user inputs IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN a JAVA file with the name of the specified concept should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN the generated concept class should have the attributes indicated in the inputs.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN the generated concept should compile.

Story Points: 5

4.2.1.12. US 1.12 Generate an action with the user interface

Epic: Ontology generator.

Description: AS a developer, I WANT to generate an action from user inputs IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN a JAVA file with the name of the specified action should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN the generated action class should have the attributes indicated in the inputs.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN the generated action should compile.

Story Points: 3

4.2.1.13. US 1.13 Generate a predicate with the user interface

Epic: Ontology generator.

Description: AS a developer, I WANT to generate a predicate from user inputs IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN a JAVA file with the name of the specified predicate should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN the generated predicate class should have the attributes indicated in the inputs.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN the generated predicate should compile.

Story Points: 3

4.2.1.14. US 1.14 Update ontology vocabulary file after each modification of the ontology with the user interface

Epic: Ontology generator.

Description: AS a developer, I WANT to update the ontology vocabulary file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new concept, THEN the ontology vocabulary file should be created/updated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new action, THEN the ontology vocabulary file should be created/updated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new predicate, THEN the ontology vocabulary file should be created/updated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new concept, action or predicate, THEN updated/created ontology vocabulary should compile.

Story Points: 3

4.2.1.15. US 1.15 Update ontology file after each modification of the ontology with the user interface

Epic: Ontology generator.

Description: AS a developer, I WANT to update the ontology file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.

Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new concept, THEN the ontology file should be created/updated.

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new action, THEN the ontology file should be created/updated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new predicate, THEN the ontology file should be created/updated.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate a new concept, action or predicate, THEN updated/created ontology should compile.

Story Points: 3

4.2.1.16. US 1.16 Documentation of the Ontology Generator

Epic: Documentation.

Description: AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.

Acceptance Criteria

- GIVEN the 'User Manual: Ontology Generator' WHEN the user wants to generate an ontology from a specification file, THEN the document should specify the necessary steps.
- GIVEN the 'User Manual: Ontology Generator' WHEN the user wants to generate an ontology using the user interface, THEN the document should specify the necessary steps.

Story Points: 5

4.2.1.16.1. Task 1.16.1 Documentation of how to generate an ontology from a specification file

Description: Indicate the steps to generate an ontology from a specification file using the Ontology Generator.

4.2.1.16.2. Task 1.16.2 Documentation of how to generate an ontology from the user interface

Description: Indicate the steps to generate an ontology from the user interface using the Ontology Generator.

4.2.1.17. US 1.17 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.

- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.2.2. Stats of the sprint

As a summary of the work planned for Sprint 1, we can consider the following three metrics:

- **Number of epics addressed:** 2
- **Total number of user stories:** 17
- **Story points:** 62

4.3. Sprint 2

This section describes the work performed during Sprint 2. The objective of this sprint was to finish with the generation of an ontology from a specification file and to have a basic prototype of the Ontology Generator interface.

4.3.1. Sprint Backlog

4.3.1.1. US 2.1 Define the arguments of the Ontology Generator application

Epic: Ontology generator.

Description: AS a developer, I WANT to define the arguments of the ontology generator application IN ORDER TO pass the necessary information to the application.

Acceptance Criteria

- GIVEN the definition of the arguments of the ontology generator WHEN the user wants to process an ontology specification file, THEN he/she should know where he/she has to indicate the name of the file.
- GIVEN the definition of the arguments of the ontology generator WHEN the user wants to process an ontology specification file, THEN he/she should know where he/she has to indicate the destination path.
- GIVEN the definition of the arguments of the ontology generator WHEN the user wants to process an ontology specification file, THEN he/she should know where he/she has to indicate the package structure.

Story Points: 1

4.3.1.2. US 2.2 Generate a JAR file with the Ontology Generator application

Epic: Ontology generator.

Description: AS a developer, I WANT to generate a JAR file with the ontology generator application IN ORDER TO run the application easily.

Acceptance Criteria

- GIVEN the jar file of the ontology generator WHEN running it with the correct arguments, THEN the ontology should be generated without errors.

Story Points: 1

4.3.1.3. US 2.3 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.3.1.4. Bug 2.4 Fix constant names in ontology vocabulary file

Epic: Ontology generator.
Related with: US 1.7 Generate the ontology vocabulary file (4.2.1.7).

Description: AS a developer, I WANT to fix the construction of constant names IN ORDER TO have a more readable ontology vocabulary file.

Acceptance Criteria

- GIVEN a word to create a constant WHEN the generator creates the constant, THEN the constant name should use the symbol _ to separate uppercase and lowercase letters.

Story Points: 1

4.3.1.5. User stories modified

This section lists the user stories created in Sprint 1 that have changed their description, story points or acceptance criteria.

- **US 1.9 Design a basic user interface for the Ontology Generator (4.2.1.9)**
  Epic: Ontology generator.

  Description: AS a developer, I WANT to design a prototype of a user interface for the ontology IN ORDER TO build the ontology from user inputs.

  Acceptance Criteria

  - GIVEN a prototype of a user interface for the ontology WHEN the user wants to create a new concept, THEN the interface should have the necessary inputs.
  - GIVEN a prototype of a user interface for the ontology WHEN the user wants to create a new action, THEN the interface should have the necessary inputs.
  - GIVEN a prototype of a user interface for the ontology WHEN the user wants to create a new predicate, THEN the interface should have the necessary inputs.
  - GIVEN a prototype of a user interface for the ontology WHEN the user wants to generate a new ontology from a specification file, THEN the interface should have the necessary buttons.

  Story Points: 5

4.3.1.6. User stories from Sprint 1

- **US 1.8 Generate the ontology file (4.2.1.8):** AS a developer, I WANT to generate the ontology file from the ontology specification file IN ORDER TO build the ontology.
- **US 1.16 Documentation of the Ontology Generator (4.2.1.16):** AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.
  - **Task 1.16.1 Documentation of how to generate an ontology from a specification file (4.2.1.16.1):** Indicate the steps to generate an ontology from a specification file using the Ontology Generator.

4.3.2. Stats of the sprint

As a summary of the work planned for Sprint 2, we can consider the following three metrics:

- **Number of epics addressed:** 2
- **Total number of user stories:** 7
- **Story points:** 23,5

The number of story points planned for this sprint is less than the number of story points planned for Sprint 1 to limit the scope according to the actions agreed in the retrospective (8.1.4).

4.4. Sprint 3

This section describes the work performed during Sprint 3. The objective of this sprint was to start with the Team - Platform environment for the agents and have a basic definition of the API for the communication with the frontend part.

4.4.1. Sprint Backlog

4.4.1.1. US 3.1 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.4.1.2. Bug 3.2 Documentation response codes of how to generate an ontology from a specification file

Epic: Documentation.
Related with: Task 1.16.1 Documentation of how to generate an ontology from a specification file (4.2.1.16.1).

Description: AS a developer, I WANT to add to the Ontology Generator: User Manual the description of the response codes IN ORDER TO have a complete documentation.

Acceptance Criteria

- GIVEN the Ontology Generator: User Manual WHEN a person wants to know the description of a response code, THEN the document should have that information.

Story Points: 0,5

4.4.1.3. US 3.3 Documentation Ontology Generator from a specification file in final memory

Epic: Documentation.

Description: AS a developer, I WANT to document the progress in the Ontology Generator in the final memory IN ORDER TO start with the final document.

Acceptance Criteria

- GIVEN the final memory WHEN a person wants to know all the information of the Ontology Generator to generate an ontology from a specification file, THEN the document should have that information.

Story Points: 5

4.4.1.4. US 3.4 Set up of the agents project

Epic: Environment definition.

Description: AS a developer, I WANT to create the project IN ORDER TO store agents of the game.

Acceptance Criteria

- GIVEN the agents project WHEN running the application, THEN the project should run without errors.

Story Points: 1

4.4.1.5. US 3.5 Structure of the agents project

Epic: Environment definition.

Description: AS a developer, I WANT to decide and implement the organization of the agents project IN ORDER TO define the Team-Platform environment.

Acceptance Criteria

- GIVEN the agents project WHEN openning the project, THEN it should be clear where team agents are.
- GIVEN the agents project WHEN openning the project, THEN it should be clear where platform agents are.
- GIVEN the agents project WHEN openning the project, THEN it should be clear where the API client is.
- GIVEN the agents project WHEN openning the project, THEN it should be clear where the BaseAgent is.

Story Points: 1

4.4.1.6. US 3.6 Documentation of the agents environment

Epic: Documentation.

Description: AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.

Acceptance Criteria

- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to set up the project, THEN the document should specify the necessary steps.

- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to implement a new platform agent, THEN the document should specify how to do it.
- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to implement a new team agent, THEN the document should specify how to do it.
- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to modify the API client, THEN the document should specify how the client is implemented.
- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to modify the BaseAgent, THEN the document should specify how the BaseAgent is implemented.

Story Points: 8

4.4.1.6.1. Task 3.6.1 Documentation of the structure of the agents project

Description: Describe the organization of the agents project.

- Platform agents.
- Team agents.
- API client.
- BaseAgent.

4.4.1.6.2. Task 3.6.2 Documentation of implementation of BaseAgent

Description: Describe the organization and methods of the BaseAgent.

4.4.1.6.3. Task 3.6.3 Documentation of how to create an agent using the BaseAgent

Description: Indicate the steps to create a new agent using the functionalities of the BaseAgent.

4.4.1.6.4. Task 3.6.4 Documentation of implementation of API client

Note: This task is defined in Sprint 3 but it will be in the Product Backlog to be performed in future sprints.

Description: Describe the organization and methods of the API client.

4.4.1.6.5. Task 3.6.5 Documentation of how to use the API client

Note: This task is defined in Sprint 3 but it will be in the Product Backlog to be performed in future sprints.

Description: Indicate the necessary actions to use the API client.

4.4.1.6.6. Task 3.6.6 Documentation of how to set up the project

Note: This task is defined in Sprint 3 but it will be in the Product Backlog to be performed in future sprints.

Description: Indicate the necessary steps to set up the Team - Platform environment.

4.4.1.7. US 3.7 Implementation of the BaseAgent

Epic: Environment definition.

Description: AS a developer, I WANT to implement a base agent with message builders IN ORDER TO allocate the common code for agents.

Acceptance Criteria

- GIVEN the BaseAgent class WHEN extending it, THEN the agents should be able to send messages with each type of protocol using one of their parent methods.

Story Points: 8

4.4.1.8. US 3.8 Definition of API

Epic: API.

Description: AS a developer, I WANT to define the API endpoints, request bodies, responses, and errors IN ORDER TO establish the basis for its implementation.

Acceptance Criteria

- GIVEN the API definition WHEN a developer wants to know the endpoints to implement, THEN the definition should contain that information.
- GIVEN the API definition WHEN a developer wants to know the request bodies to implement, THEN the definition should contain that information.
- GIVEN the API definition WHEN a developer wants to know the responses to implement, THEN the definition should contain that information.

Story Points: 5

4.4.2. Stats of the sprint

As a summary of the work planned for Sprint 3, we can consider the following three metrics:

- **Number of epics addressed:** 3
- **Total number of user stories:** 8
- **Story points:** 29,5

4.5. Sprint 4

This section describes the work performed during Sprint 4. The objective of this sprint was to complete the implementation of the agents part of the environment and start with the API communications.

4.5.1. Sprint Backlog

4.5.1.1. US 4.1 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.5.1.2. US 4.2 Documentation of Sprint 3 in final memory

Epic: Documentation.

Description: AS a developer, I WANT to document the progress in Sprint 3 in the final memory IN ORDER TO have all information in the final memory.

Acceptance Criteria

- GIVEN the final memory WHEN a person wants to know the progress made during Sprint 3, THEN the document should have that information.

Story Points: 3

4.5.1.3. US 4.3 Creation of Team - Platform example agents

Epic: Environment definition.

Description: AS a developer, I WANT to create Team - Platform example agents IN ORDER TO illustrate Team - Platform agents.

Acceptance Criteria

- GIVEN a Team example agent WHEN a developer wants to create a new Team agent, THEN the process should be clear.
- GIVEN a Platform example agent WHEN a developer wants to create a new Platform agent, THEN the process should be clear.

Story Points: 3

4.5.1.4. US 4.4 Agent loading

Epic: Environment definition.

Description: AS a developer, I WANT to load team and platform agents IN ORDER TO initialize the game.

Acceptance Criteria

- GIVEN the agents project WHEN running the project, THEN the platform agents should be loaded.
- GIVEN the agents project WHEN running the project, THEN the team agents should be loaded.

Story Points: 3

4.5.1.5. US 4.5 Implementation of API client

Epic: API.

Description: AS a developer, I WANT to implement the API client following the definition IN ORDER TO establish the connection with the frontend.

Acceptance Criteria

- GIVEN the API client class WHEN using it, THEN the agents should be able to connect with the frontend.
- GIVEN the API client class WHEN an error occurs in the HTTP calls, THEN the API client should manage that error.
- GIVEN the API client class WHEN executing a HTTP request to the frontend, THEN the agents should receive the response of the call as specified in the definition.

Story Points: 8

4.5.1.6. User stories from Sprint 3

- **US 3.8 Definition of API (4.4.1.8):** AS a developer, I WANT to define the API endpoints, request bodies, responses, and errors IN ORDER TO establish the basis for its implementation.
- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.
    - **Task 3.6.3 Documentation of how to create an agent using the BaseAgent (4.4.1.6.3):** Indicate the steps to create a new agent using the functionalities of the BaseAgent.

4.5.1.7. User stories from Product Backlog

- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.

- **Task 3.6.6 Documentation of how to set up the project (4.4.1.6.6):** Indicate the necessary steps to set up the Team - Platform environment.

4.5.2. Stats of the sprint

As a summary of the work planned for Sprint 4, we can consider the following three metrics:

- **Number of epics addressed:** 3
- **Total number of user stories:** 7
- **Story points:** 29,6

4.6. Sprint 5

This section describes the work performed during Sprint 5. The objective of this sprint was to complete the documentation of the Team - Platform environment with the API client and start with the Ontology Generator interface.

4.6.1. Sprint Backlog

4.6.1.1. US 5.1 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.6.1.2. US 5.2 Modifications of the API

Epic: API.

Description: AS a developer, I WANT to implement the necessary changes IN ORDER TO align with the frontend endpoints.

Acceptance Criteria

- GIVEN the API definition and the GameHttpClient WHEN the agents want to end the game, THEN they should send the final results to the frontend in the request body.

- GIVEN the API definition and the GameHttpClient WHEN the agents want to kill a unit, THEN the API should allow that action.
- GIVEN the API definition and the GameHttpClient WHEN a unit wants to attack another unit, THEN it should pass the remaining health of the attacked unit.
- GIVEN the API definition and the GameHttpClient WHEN the agents want to reduce the health of a unit, THEN the API should allow that action.

Story Points: 3

4.6.1.3. US 5.3 Modifications of the Ontology Generator interface

Epic: Ontology Generator.

Description: AS a developer, I WANT to modify the design of the Ontology Generator interface IN ORDER TO add the changes proposed in the Sprint Review of Sprint 4.

Acceptance Criteria

- GIVEN the design of the Ontology Generator interface WHEN the main screen is displayed, THEN the button to generate an ontology from a specification file should have the label 'Generate all from specification file'.
- GIVEN the design of the Ontology Generator interface WHEN the main screen is displayed, THEN the options to generate a new concept, action, or predicate should be displayed using tabs.

Story Points: 1

4.6.1.4. US 5.4 Set up the Ontology Generator interface

Epic: Ontology Generator.

Description: AS a developer, I WANT to set up the Ontology Generator interface IN ORDER TO prepare the interface to add the screens.

Acceptance Criteria

- GIVEN the set up of the interface WHEN running the interface, THEN an empty screen should appear.
- GIVEN the set up of the interface WHEN running the interface and clicking the close button, THEN the screen should disappear.

Story Points: 3

4.6.1.5. US 5.5 Ontology Generator interface visual Main screen

Epic: Ontology generator.

Description: AS a developer, I WANT to implement the visual part of the Main screen IN ORDER TO develop the Ontology Generator interface.

Acceptance Criteria

- GIVEN Ontology Generator interface WHEN running the application, THEN the Main screen of the Ontology Generator should appear.
- GIVEN Ontology Generator interface WHEN running the application, THEN the Main screen should have four inputs (ontology specification file, ontology name, package declaration and destination path).
- GIVEN Ontology Generator interface WHEN running the application, THEN the Main screen should have a way to perform the following actions: Generate from specification file, Generate concept, Generate action and Generate predicate..
- GIVEN Ontology Generator interface WHEN running the application, THEN the Main screen should have the success and error banners to display the results of the operations.
- GIVEN Ontology Generator interface WHEN running the application, THEN the Main screen should be able to display the file not found error.

Story Points: 5

4.6.1.6. US 5.6 Ontology Generator interface visual New Concept screen

Epic: Ontology generator.

Description: AS a developer, I WANT to implement the visual part of the New concept screen IN ORDER TO add a new concept through the Ontology Generator interface.

Acceptance Criteria

- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Concept tab, THEN the New Concept screen of the Ontology Generator should appear.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Concept tab, THEN the New Concept screen should have three inputs (class name, first attribute name, and first attribute type).
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Concept tab, THEN the New Concept screen should have a button to generate the concept.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Concept tab, THEN the New Concept screen should have the error banner to display the result of the operation.

Story Points: 5

4.6.1.7. US 5.7 Generate the ontology specification file

Epic: Ontology generator.

Description: AS a developer, I WANT to implement a file specification service IN ORDER TO generate the ontology specification file after any change in the ontology through the Ontology Generator interface.

Acceptance Criteria

- GIVEN a new concept added through the Ontology Generator interface WHEN the change has been done in the interface, THEN an ontology specification file should be generated with the data of the new concept.
- GIVEN a new action added through the Ontology Generator interface WHEN the change has been done in the interface, THEN an ontology specification file should be generated with the data of the new action.
- GIVEN a new predicate added through the Ontology Generator interface WHEN the change has been done in the interface, THEN an ontology specification file should be generated with the data of the new predicate.
- GIVEN the ontology specification file generated WHEN a user wants to perform any change in that ontology, THEN file should contain the name of the ontology.

Story Points: 8

4.6.1.8. User stories from Product Backlog

- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.
  - **Task 3.6.4 Documentation of implementation of API client (4.4.1.6.4):** Describe the organization and methods of the API client.
  - **Task 3.6.5 Documentation of how to use the API client (4.4.1.6.5):** Indicate the necessary actions to use the API client.
- **US 1.10 Generate the ontology from a specification file using the interface (4.2.1.10):** AS a developer, I WANT to generate the ontology from a specification file IN ORDER TO build the ontology from the user interface.

4.6.1.9. User stories canceled

As a result of the user flow agreed in Sprint Review 4 for the Ontology Generator interface (5.4.3). These user stories were canceled.

- **US 1.14 Update ontology vocabulary file after each modification of the ontology with the user interface (4.2.1.14):** AS a developer, I WANT to update the ontology vocabulary file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.
- **US 1.15 Update ontology file after each modification of the ontology with the user interface (4.2.1.15):** AS a developer, I WANT to update the ontology file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.

4.6.2. Stats of the sprint

As a summary of the work planned for Sprint 5, we can consider the following three metrics:

- **Number of epics addressed:** 3
- **Total number of user stories:** 9
- **Story points:** 33,6

4.7. Sprint 6

This section describes the work performed during Sprint 6. The objective of this sprint was to finish the API definition and to complete the visual part of the Ontology Generator interface.

4.7.1. Sprint Backlog

4.7.1.1. US 6.1 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.7.1.2. US 6.2 Ontology Generator interface visual New Action screen

Epic: Ontology generator.

Description: AS a developer, I WANT to implement the visual part of the New action screen IN ORDER TO add a new action through the Ontology Generator interface.

Acceptance Criteria

- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Action tab, THEN the New Action screen of the Ontology Generator should appear.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Action tab,

THEN the New Action screen should have three inputs (class name, first attribute name and first attribute type).

- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Action tab, THEN the New Action screen should have a button to generate the action.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Action tab, THEN the New Action screen should have the success and error banners to display the results of the operations.

Story Points: 1

4.7.1.3. US 6.3 Ontology Generator interface visual New Predicate screen

Epic: Ontology generator.

Description: AS a developer, I WANT to implement the visual part of the New predicate screen IN ORDER TO add a new predicate through the Ontology Generator interface.

Acceptance Criteria

- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Predicate tab, THEN the New Predicate screen of the Ontology Generator should appear.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Predicate tab, THEN the New Predicate screen should have three inputs (class name, first attribute name and first attribute type).
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Predicate tab, THEN the New Predicate screen should have a button to generate the predicate.
- GIVEN Ontology Generator interface WHEN running the application, entering the required data of the home screen and going to the Predicate tab, THEN the New Predicate screen should have the success and error banners to display the results of the operations.

Story Points: 1

4.7.1.4. Bug 6.4 The inputs modify its width depending on its content

Epic: Ontology Generator.
Related with: US 5.5 Ontology Generator interface visual Main screen (4.6.1.5).

Description: AS a developer, I WANT to change the distribution of the Main screen to one column IN ORDER TO avoid the changes in width depending on user input.

Acceptance Criteria

- GIVEN the Ontology Generator interface design and prototype WHEN the user enters a value in the Main screen, THEN the Main screen should have one column.
- GIVEN the Main screen WHEN the user enters a value greater than the TextField width, THEN the TextField should not change its width.

Story Points: 3

4.7.1.5. Bug 6.5 Open user interface with double click on JAR file

Epic: Ontology Generator.
Related with: US 2.2 Generate a JAR file with the Ontology Generator application (4.3.1.2).

Description: AS a developer, I WANT to open the Ontology Generator user interface with a double click on the JAR file IN ORDER TO make the Ontology Generator more accessible to the users.

Acceptance Criteria

- GIVEN the Ontology Generator JAR file WHEN double clicking on the file, THEN the Ontology Generator interface should be displayed.

Story Points: 1

4.7.1.6. US 6.6 Ontology Generator interface Selector of types in New Concept screen

Epic: Ontology generator.

Description: AS a developer, I WANT to add a selector of types to the New Concept screen IN ORDER TO make it easier to insert attributes.

Acceptance Criteria

- GIVEN Ontology Generator interface design and prototype of the New Concept/Action/Predicate screen WHEN a user is adding a new attribute, THEN the New Concept/Action/Predicate screens should have a selector with all possible values.
- GIVEN Ontology Generator interface New Concept screen WHEN a user is adding a new attribute, THEN the New Concept screen should have a selector with all possible values.
- GIVEN Ontology Generator interface New Concept screen WHEN a user is adding a new attribute, THEN the types selector should contain basic type and the concepts of the ontology.
- GIVEN Ontology Generator interface New Concept screen WHEN a user is typing the type of a new attribute, THEN the types selector should filter the possible values with the input data.

Story Points: 5

4.7.1.7. User stories from Sprint 5

- **US 5.2 Modifications of the API (4.6.1.2):** AS a developer, I WANT to implement the necessary changes IN ORDER TO align with the frontend endpoints.

4.7.1.8. User stories from Product Backlog

- **US 1.16 Documentation of the Ontology Generator (4.2.1.16):** AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.
  - **Task 1.16.2 Documentation of how to generate an ontology from the user interface (4.2.1.16.2):** Indicate the steps to generate an ontology from the user interface using the Ontology Generator.

4.7.1.9. User stories modified

This section lists the user stories created in previous sprints that have changed their description, story points or acceptance criteria.

- **US 1.11 Generate a concept with the user interface (4.2.1.11)**
  Epic: Ontology generator.

  Description: AS a developer, I WANT to generate a concept from user inputs IN ORDER TO build the ontology from the user interface.

  Acceptance Criteria
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN a JAVA file with the name of the specified concept should be created.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN the generated concept class should have the attributes indicated in the inputs.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN the generated concept should compile.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a concept, THEN an ontology specification file with the data should be generated.

  Story Points: 5

- **US 1.12 Generate an action with the user interface (4.2.1.12)**
  Epic: Ontology generator.

  Description: AS a developer, I WANT to generate an action from user inputs IN ORDER TO build the ontology from the user interface.

  Acceptance Criteria

- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN a JAVA file with the name of the specified action should be created.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN the generated action class should have the attributes indicated in the inputs.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN the generated action should compile.
- GIVEN the user interface WHEN the user performs the necessary user inputs to generate an action, THEN an ontology specification file with the data should be generated.

Story Points: 3

- **US 1.13 Generate a predicate with the user interface (4.2.1.13)**
  Epic: Ontology generator.

  Description: AS a developer, I WANT to generate a predicate from user inputs IN ORDER TO build the ontology from the user interface.

  Acceptance Criteria

  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN a JAVA file with the name of the specified predicate should be created.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN the generated predicate class should have the attributes indicated in the inputs.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN the generated predicate should compile.
  - GIVEN the user interface WHEN the user performs the necessary user inputs to generate a predicate, THEN an ontology specification file with the data should be generated.

  Story Points: 3

4.7.2. Stats of the sprint

As a summary of the work planned for Sprint 6, we can consider the following three metrics:

- **Number of epics addressed:** 3
- **Total number of user stories:** 11
- **Story points:** 32,5

4.8. Sprint 7

This section describes the work performed during Sprint 7. The objective of this sprint was to write some chapters of the final memory and to implement the Log standardization epic.

4.8.1. Sprint Backlog

4.8.1.1. US 7.1 Documentation of the sprint

Epic: Documentation.

Description: AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

Acceptance Criteria

- GIVEN the documentation of the sprint WHEN a person wants to know the user stories of the current sprint, THEN the document should have that information.
- GIVEN the documentation of the sprint WHEN a person wants to know the results of the current sprint, THEN the document should have that information.

Story Points: 5

4.8.1.2. US 7.2 Analysis of log standardization

Epic: Log standardization.

Description: AS a developer, I WANT to analyze the possibilities to standardize the logs in the agents project IN ORDER TO implement the best solution.

Acceptance Criteria

- GIVEN an agents project WHEN a developer wants to implement log standardization in the project, THEN all possibilities has to be contemplated.

Story Points: 3

4.8.1.3. US 7.3 Implementation of log standardization

Epic: Log standardization.

Description: AS a developer, I WANT to implement the best log standardization solution in the agents project IN ORDER TO recognize each agent and team.

Acceptance Criteria

- GIVEN an agent of a team in the agents project WHEN a user wants to know the decisions that the agent takes, THEN the project should provide an easy way to localize the logs of an agent from a team.

Story Points: 5

4.8.1.4. US 7.4 Documentation of log standardization

Epic: Documentation.

Description: AS a developer, I WANT to document the log standardization implementation IN ORDER TO use it in the agents of the agents project.

Acceptance Criteria

- GIVEN the 'Team - Platform Environment: Developer Manual' WHEN the developer wants to use the log standardization, THEN the document should specify how to do it.

Story Points: 3

4.8.1.5. US 7.5 Introduction of the memory

Epic: Documentation.

Description: AS a researcher, I WANT to write the introduction chapter of the final memory IN ORDER TO offer the readers of the document the purpose, scope, and context of the project.

Acceptance Criteria

- GIVEN the final memory document WHEN a person wants to understand the purpose, scope, and context of the project, THEN the final memory document should have that information.

Story Points: 5

4.8.1.6. US 7.6 Global results of the memory

Epic: Documentation.

Description: AS a researcher, I WANT to write the global results section of the final memory IN ORDER TO offer the readers the overall outcomes, key findings, or trends derived from the analysis presented.

Acceptance Criteria

- GIVEN the final memory document WHEN a person wants to know the overall outcomes, key findings, or trends derived from the analysis presented, THEN the final memory document should have that information.

Story Points: 5

4.8.1.7. US 7.7 Abstract of the memory

Epic: Documentation.

Description: AS a researcher, I WANT to write the Abstract section (Spanish and English version) of the final memory IN ORDER TO offer the readers the main objectives, key findings, and conclusions without reading the entire text.

Acceptance Criteria

- GIVEN the final memory document WHEN a person wants to quickly know the main objectives, key findings, and conclusions, THEN the final memory document should have that information.

Story Points: 5

4.8.1.8. US 7.8 Game description section

Epic: Documentation.

Description: AS a researcher, I WANT to write a game description chapter in the final memory IN ORDER TO give the readers an overview of the basis of the new game.

Acceptance Criteria

- GIVEN the final memory document WHEN a person wants to know the basis of the agents game in which this work will be used, THEN the final memory document should have that information.

Story Points: 3

4.8.2. Stats of the sprint

As a summary of the work planned for Sprint 7, we can consider the following three metrics:

- **Number of epics addressed:** 2
- **Total number of user stories:** 8
- **Story points:** 34

5 Ontology Generator

Just as people use languages to communicate with each other, in the world of agents the communications between the different agents are based on ontologies. An ontology in the agents world is a formal and shared knowledge structure that defines the concepts, relationships, and rules relevant to a domain, enabling agents to interpret, reason about, and communicate knowledge consistently.

Sometimes, these ontologies take a lot of effort to create because they are composed of a large number of concepts, actions, and predicates.

The Ontology Generator reduces that effort, offering a tool to build ontologies from a simple text file or through an interactive interface. This application takes the user's specifications and creates the concepts, actions, predicates, the vocabulary, and groups all in an ontology.

This section explains the Ontology Generator in detail, and there is also a user manual appended to this document (12.1).

5.1. Class diagram

This class design represents the logic layer of the Ontology Generator. It is composed of four main services (OntologyService, ParseService, GeneratorService and FileService), five generators (ConceptGenerator, ActionGenerator, PredicateGenerator, VocabularyGenerator and OntologyGenerator), and one manager (OntologyGui). The functionality of each component is explained in this section.

5.1.1. Diagram

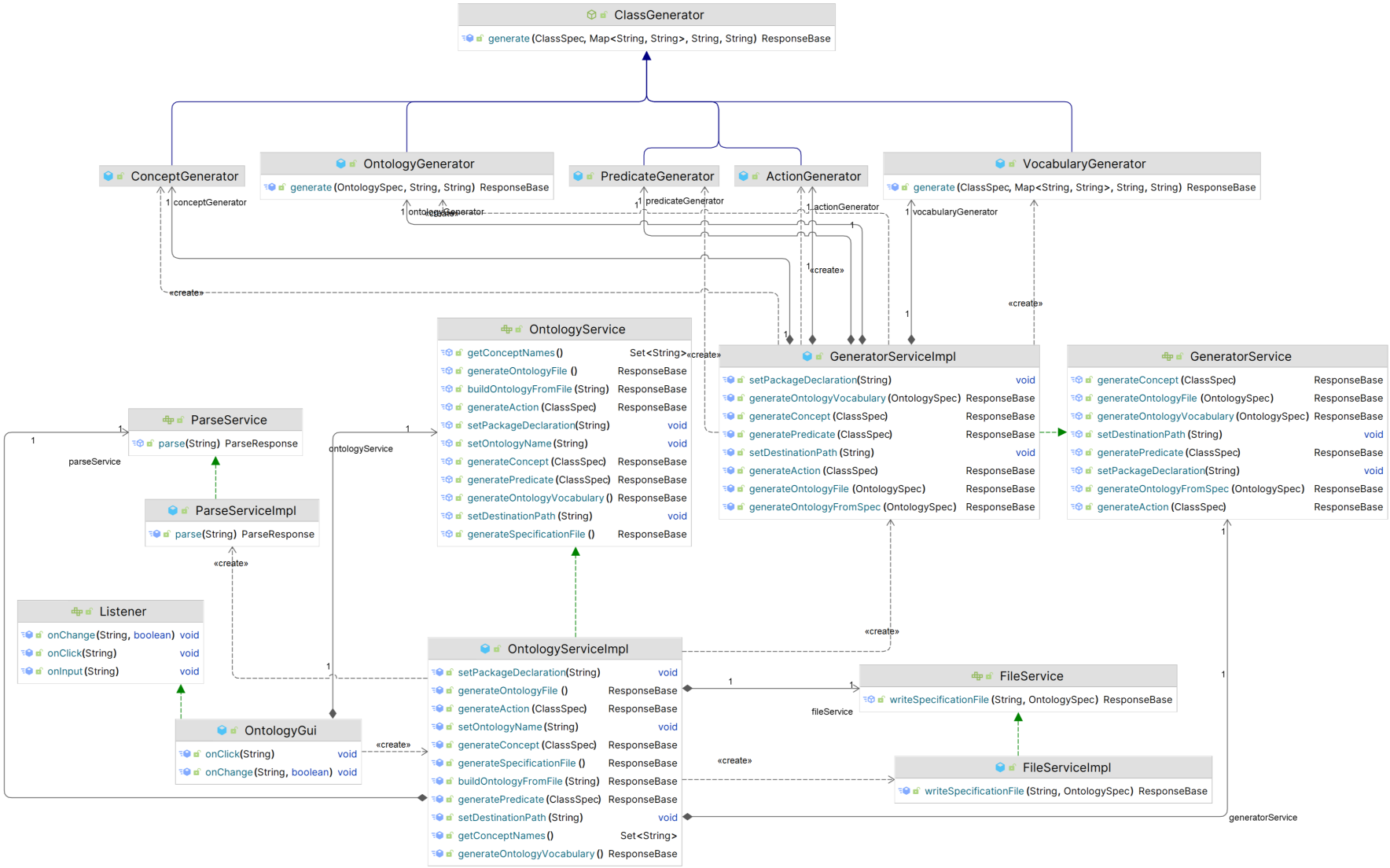


Figure 5.1: Class design of Ontology Generator

5.1.2. Services

- **OntologyService:** This is the main service of the application. Manages the generation of the ontology through:

  a) A **specification file**: It receives the path of the specification file, calls the ParseService to extract the data of the file, and sends that data to the GeneratorService to build the ontology.

  b) The **user interface**: It has the necessary methods to order to the GeneratorService the generation of concepts, actions, predicates, the vocabulary ontology file, and the ontology file. In addition, it has a method to order the FileService the writing of the ontology specification file with the items created during the execution of the program.

  The service exposes the following methods:

  - ***setPackageDeclaration(String packageDeclaration)***: This method allows to establish the desired package structure for the ontology.
  - ***setDestinationPath(String destinationPath)***: This method allows to establish the desired destination path to save the ontology files.
  - ***setOntologyName(String ontologyName)***: This method allows to establish the ontology name.
  - ***getConceptNames()*** → ***Set<String>***: Method to get the names of the concepts of the ontology.
  - ***buildOntologyFromFile(String pathSpecificationFile)*** → ***ResponseBase***: Method with the path of the ontology specification file as parameter. Returns the result of the operation.
  - ***generateConcept(ClassSpec conceptSpecification)*** → ***ResponseBase***: Method to generate a concept file of the ontology from a concept specification. Returns the result of the operation.
  - ***generateAction(ClassSpec actionSpecification)*** → ***ResponseBase***: Method to generate an action file of the ontology from an action specification. Returns the result of the operation.
  - ***generatePredicate(ClassSpec predicateSpecification)*** → ***ResponseBase***: Method to generate a predicate file of the ontology from a predicate specification. Returns the result of the operation.
  - ***generateOntologyVocabulary()*** → ***ResponseBase***: Method to generate the vocabulary file of the ontology using the items created previously. Returns the result of the operation.
  - ***generateOntologyFile()*** → ***ResponseBase***: Method to generate the main ontology file using the items created previously. Returns the result of the operation.

- **ParseService:** This service is in charge of processing the specification file to obtain a representation of the ontology understandable by the generators.

  - ***parse(String pathSpecificationFile)*** → ***ParseResponse***: Method to process the specification file indicated and returns the result of the operation and, if it does not have any error, an ontology specification

with a list of concepts, a list of actions, a list of predicates and the name of the ontology.

- **GeneratorService:** This service manages the generation of the different files that compose an ontology. Receives the ontology data and distributes the work between the generators.
  - ***setPackageDeclaration(String packageDeclaration)***: This method allows to establish the desired package structure for the ontology.
  - ***setDestinationPath(String destinationPath)***: This method allows to establish the desired destination path to save the ontology files.
  - ***generateOntologyFromSpec(OntologySpec ontologySpecification)*** → ***ResponseBase***: Method to generate ontology files from an ontology specification. Returns the result of the operation.
  - ***generateConcept(ClassSpec conceptSpecification)*** → ***ResponseBase***: Method to generate a concept file of the ontology from a concept specification. Returns the result of the operation.
  - ***generateAction(ClassSpec actionSpecification)*** → ***ResponseBase***: Method to generate an action file of the ontology from an action specification. Returns the result of the operation.
  - ***generatePredicate(ClassSpec predicateSpecification)*** → ***ResponseBase***: Method to generate a predicate file of the ontology from a predicate specification. Returns the result of the operation.
  - ***generateOntologyVocabulary(OntologySpec ontologySpec)*** → ***ResponseBase***: Method to generate the vocabulary file of the ontology from a ontology specification. Returns the result of the operation.
  - ***generateOntologyFile(OntologySpec ontologySpecification)*** → ***ResponseBase***: Method to generate the main ontology file from an ontology specification. Returns the result of the operation.
- **FileService:** This is the service responsible for writing the ontology specification file after any change made through the user interface.
  - ***writeSpecificationFile(String destinationPath, OntologySpec spec)*** → ***ResponseBase***: Method to generate the ontology specification file (with the structure described in 5.3) in the destination path indicated as a parameter. It receives the ontology specification with the data of the ontology to write and returns the result of the operation.

5.1.3. Generators

These are the classes that write the ontology files in the specified destination path. All of them extend the abstract class ClassGenerator that has the shared logic with a public generate method.

- ***generate(ClassSpec classSpecification, Map<String, String> imports, String packageDeclaration, String destinationPath)*** → ***ResponseBase***: Method to write a JAVA class file. It receives the class

specification, a map with the imports of other classes, the package declaration of the class and the destination path to save the file, and it returns the result of the operation.

- **ConceptGenerator:** Create a concept class of the ontology with the indicated name and attributes.
- **ActionGenerator:** Create an action class of the ontology with the indicated name and attributes.
- **PredicateGenerator:** Create a predicate class of the ontology with the indicated name and attributes.
- **VocabularyGenerator:** Create the ontology vocabulary class with the necessary constants.
- **OntologyGenerator:** Create the ontology file that adds all the concepts, actions, and predicates to the ontology using the constants of the vocabulary class.

5.1.4. Managers

- **OntologyGui**: This is the manager of the Ontology Generator interface. Creates the interface frame with the different panels (screens) and is in charge of the communications between the interface screens and the services/generators of the Ontology Generator. The OntologyGui and some components of the interface implement the Listener interface that defines methods for the communication between GUI components. The **Listener** interface defines the following methods:
  - ***onInput(String source)***: Notify a new user input on a child component (source).
  - ***onClick(String source)***: Notify a click on a child component (source).
  - ***onChange(String source, boolean state)***: Notify a change on a child component (source) with the new state (true indicating enabled or false meaning disabled).

5.2. Structure of the source code

In this section we are going to explain the structure of the Ontology Generator.

First of all, it was decided to use Git [7] as a version control tool for the code, for this, a private repository was created for the project code in GitHub [8].

Git is a version control tool that allows the creation of branches among many other things. A branch is a division of the code that allows to implement a new functionality without modifying the main code of the project, and once the implementation is finished to incorporate that branch to the main project.

Secondly, a structure had to be decided to develop the source code using the programming language Java [9]. The internal structure of the Ontology Generator is:

- lib: Folder to add external libraries that cannot be downloaded via Maven.
  - jade-4.6.0.jar: JADE library [3].
- src.main.java.org.ontology.generator: Main folder of the project.
  - exception: Folder with the exceptions used in the services.
    - OntologyException: Internal exception used in the services to notify specific errors.
  - gui: Folder with the source code of the visual part of the interface.
    - components: Folder with the components used in the different panels (screens).
      - ActionsButtons: Component that implements the buttons to add and remove attributes inputs from the AddItemPanel (used for the New concept, New action, and New predicate screens).
      - Banner: Component that represents the Success and Error banners used to communicate the result of the operation.
      - SearchComboBox: Custom combobox with search functionality.
      - TextField: Custom form input with labelled text and an error state.
    - dto: Folder that contains the models to store the user input data.
      - OntologyConfigDTO: Model to store the data of the HomePanel (name of the specification file, ontology name, package declaration, and destination path).
    - panels: Folder with the screens (panels) of the interface (5.4.1).
      - AddItemPanel: Panel that represents New concept, New action, and New predicate screens.
      - HomePanel: Main screen of the interface.
    - utils: Folder with auxiliary data.
      - Colors: Colors used in the interface.
      - Messages: Success and error messages used in the interface.
    - Listener: Interface that defines communication between GUI items.
    - OntologyGui: Main class of the interface. Controls the screens and handles the communication with the OntologyService to perform the user-required actions.
  - model: Folder with the models used by the Ontology Generator to build the ontology.
    - Attribute: Model that represents an attribute of a Java class. It is used by the ClassSpec model.
    - BuildClassModel: Model used in the generators to store the data of a class that is going to be written in the file.

- ClassSpec: Model that represents a Java class. It contains all the data to create a specific class, and it is used by the OntologySpec model.
- OntologySpec: Model that represents an ontology. It contains all the necessary data to build the ontology (concepts, actions and predicates classes). It is used in the ParseResponse model and as argument in several services.
- ParseResponse: Model that represents the response of the ParseService with the result of parsing an ontology specification file.
- ResponseBase: Model that represents the responses of the services.
- ResponseCodes: Enum with the definition of the response codes used in the responses (5.5).
- SpecificClassData: Model to store the specific data of each ontology class type (declaration import and class declaration).

- service: Folder that contains the logic of the Ontology Generator. A complete description of the services are in 5.1.2.
  - generators: Folder that contains the classes in charge of writing the ontology files. They are described in detail above in the document (5.1.3).
    - ActionGenerator: Class that extends ClassGenerator to generate ontology actions.
    - ClassGenerator: Abstract class that has the common logic for the generation.
    - ConceptGenerator: Class that extends ClassGenerator to generate ontology concepts.
    - OntologyGenerator: Class that extends ClassGenerator to generate the ontology file.
    - PredicateGenerator: Class that extends ClassGenerator to generate ontology predicates.
    - VocabularyGenerator: Class that extends ClassGenerator to generate the ontology vocabulary file.
  - FileService: Interface that defines the methods to write the ontology specification file.
  - FileServiceImpl: Service that implements the FileService interface.
  - GeneratorService: Interface that defines the methods to generate ontology items.
  - GeneratorServiceImpl: Service that implements the GeneratorService interface.
  - OntologyService: Interface that defines the main Ontology Generator service with methods to set input arguments and generates the ontology

- OntologyServiceImpl: Service that implements the OntologyService interface.
- ParseService: Interface that defines a parse method to obtain the ontology data from a file.
- ParseServiceImpl: Service that implements the ParseService interface.

utils: Folder to store auxiliary classes.

- Utils: Class that declares constants used along the project, and contains common and re-used methods.

OntologyGeneratorApplication: File that has the main method of the Ontology Generator. This is the file that must be executed to create an ontology.

pom.xml: File with the project dependencies downloaded via Maven.

5.3. Specification file

5.3.1. How to write the ontology specification file?

This section explains how to write the ontology specification file to allow the software to correctly generate the ontology.

The user must provide a text file following the structure and rules described in the next sections.

5.3.1.1. Structure of the file

This section specifies the requirements that the text file must comply for the software to work correctly.

The structure of the file must be the following:

```
OntologyName
concepts
    -ConceptName1
        attributeName1: TypeAttribute1
        attributeName2: TypeAttribute2

actions
    -ActionName1
    -ActionName2
        attributeName1: TypeAttribute1

predicates
    -PredicateName1
        attributeName1: TypeAttribute1
        attributeName2: TypeAttribute2
```

The file must start with the name of the ontology (**OntologyName**) and must have three reserved words that indicate the three different groups of classes (described in detail in chapter 5.3.2):

- **concepts**: In this group must be specified the concepts of the ontology.

- **actions**: In this group must be specified the actions of the ontology.
- **predicates**: In this group must be specified the predicates of the ontology.

Each group can be empty or can contain one or more class specifications. Each class specification must follow the structure described in the class specification subsection 5.3.1.1.1.

5.3.1.1.1. Class specification

A class specification is the scheme to define the properties of a class (name and attributes). It must have the following structure:

```
-Name
    attributeName_1: TypeAttribute_1
    attributeName_2: TypeAttribute_2
    ...
```

Where:

- **Name** is the name of the class.
- **attributeName**$_i$ is the name of the attribute i of the class.
- **TypeAttribute**$_i$ is the type of the attribute i of the class.

To define a class without attributes, it is only necessary to specify the name of the class.

Examples of class specifications:

- Car class with two attributes: model of type String and price of type double.

```
-Car
    model: String
    price: double
```

- Utils class with no attributes.

```
-Utils
```

5.3.1.2. Rules

This section enumerates the rules that must be followed for the correct processing of the file with the ontology specification.

1. The **name of the ontology** must be in the first line of the file and it can not be omitted.
2. The three **reserved words** (concepts, actions and predicates) must appear in the file whether or not they contain specifications.
3. If a group does not contain any class specification, leave it empty.
4. The different groups of class specifications defined by the reserved words (concepts, actions and predicates) must be separated by a blank line.

5. The **class specifications** inside each group must follow the structure described in subsection 5.3.1.1.1:
   5.1. The class specification must start with the symbol - following by the name of the class.
   5.2. All attributes of a class specification must have a name and a type.
   5.3. The names and types of the attributes of a class specification must be separated by the symbol :.
   5.4. If a class does not contain any attribute, indicate only the name of the class.
6. If a concept (X) has any attribute whose type is another concept (Y), the concept Y **must be declared before** the concept X.

5.3.2. Ontology classes

This chapter describes the different ontology classes generated by the Ontology Generator.

The Ontology Generator generates five different types of classes (concepts, actions, predicates, ontology vocabulary and ontology), but the user only need to specify three: concepts, actions, and predicates. The following sections explain how to specify them in the specification file.

5.3.2.1. Concepts

The concepts of the ontology are the classes that implement the Concept interface of the JADE library.

The specifications of the concept classes of the ontology must be included in the **concepts group** described in the section 5.3.1.1 and following the rules of section 5.3.1.2.

Good example of concepts group with three concepts:

```
    concepts
        -Building
            type: String
        -Coordinate
            xValue: int
            yValue: int
        -Cell
            coord: Coordinate
            content: String
```

Bad example of concepts group:

```
    concepts
        -Cell
            coord: Coordinate
            content: String
        -Building
            type: String
        -Coordinate
            xValue: int
            yValue: int
```

The above example is not applying rule 5 of section 5.3.1.2.

5.3.2.2. Actions

The actions of the ontology are the classes that implement the AgentAction interface of the JADE library.

The specifications of the action classes of the ontology must be included in the **actions group** described in section 5.3.1.1 and following the rules in section 5.3.1.2.

Example of actions group with two actions:

```
    actions
        -ConstructBuilding
            building: Building
        -DistributeMap
            mapSize: Coordinate
```

5.3.2.3. Predicates

The predicates of the ontology are the classes that implement the Predicate interface of the JADE library.

The specifications of the predicate classes of the ontology must be included in the **predicates group** described in section 5.3.1.1 and following the rules in section 5.3.1.2.

Example of predicates group with one predicate:

```
    predicates
        -Proposes
            estimation: Estimation
```

5.3.3. Example of the specification file

This chapter contains a complete example of an ontology described using the specification file explained in this document.

```
    WoAOntology
    concepts
        -Building
            type: String
        -Coordinate
            xValue: int
            yValue: int
        -Cell
            coord: Coordinate
            content: String
        -Resource
            typeRes: String
            amount: float
        -CurrentResources
            gold: Resource
            stone: Resource
            wood: Resource
        -Destination
            newDest: Coordinate
```

```
        -Direction
            dir: int
        -NewPhase
            phase: int
        -StorageCapacity
            size: float

    actions
        -AllocateUnit
            initialPosition: Coordinate
        -AssignNewUnit
            unitID: AID
        -ChangePhase
        -CollectResource
        -ConstructBuilding
            building: Building
        -CreateUnit
        -DistributeMap
            mapSize: Coordinate
        -InformCurrentResources
            tribeResources: CurrentResources
            storage: StorageCapacity
        -InformNewBuilding
            cell: Cell
        -Move
            dest: Direction
        -Register
        -RevealCell
            cellContent: Cell

    predicates
```

5.4. Interface

5.4.1. Design

Before starting the implementation of the Ontology Generator interface, a design was developed using Figma [10] to structure the different user inputs and clarify the user flows. The final design can be viewed at:

`https://www.figma.com/design/sJme3sDNZ0XsaDTyzdkGcq/Ontology-Generator?node-id=1-2&t=uaWtXEPQHdZdjUCM-1`.

The description of each screen is the following:

- **Main screen:** First screen that appears in the interface, it has the general inputs for the ontology (the package declaration and the destination path). Also, it has an input to specify the ontology specification file and other for the ontology name. In the first version of the design, this screen had four buttons to start the four functionalities of the Ontology Generator (Generate from a specification file, Generate concept, Generate action and Generate predicate). But as result of the Sprint Review of Sprint 4 (US 5.3 Modifications of the Ontology Generator interface (4.6.1.3)), we decided to change the buttons for tabs in the top of the screen and maintain only the Generate all from specification file button. This change implies a second version of all screens of the design. Finally, we found a bug in the width of the inputs in the Sprint Review of Sprint 5 and we decided to change the distribution of this screen to one column (Bug 6.4 The inputs modify its width depending on its content (4.7.1.4)).

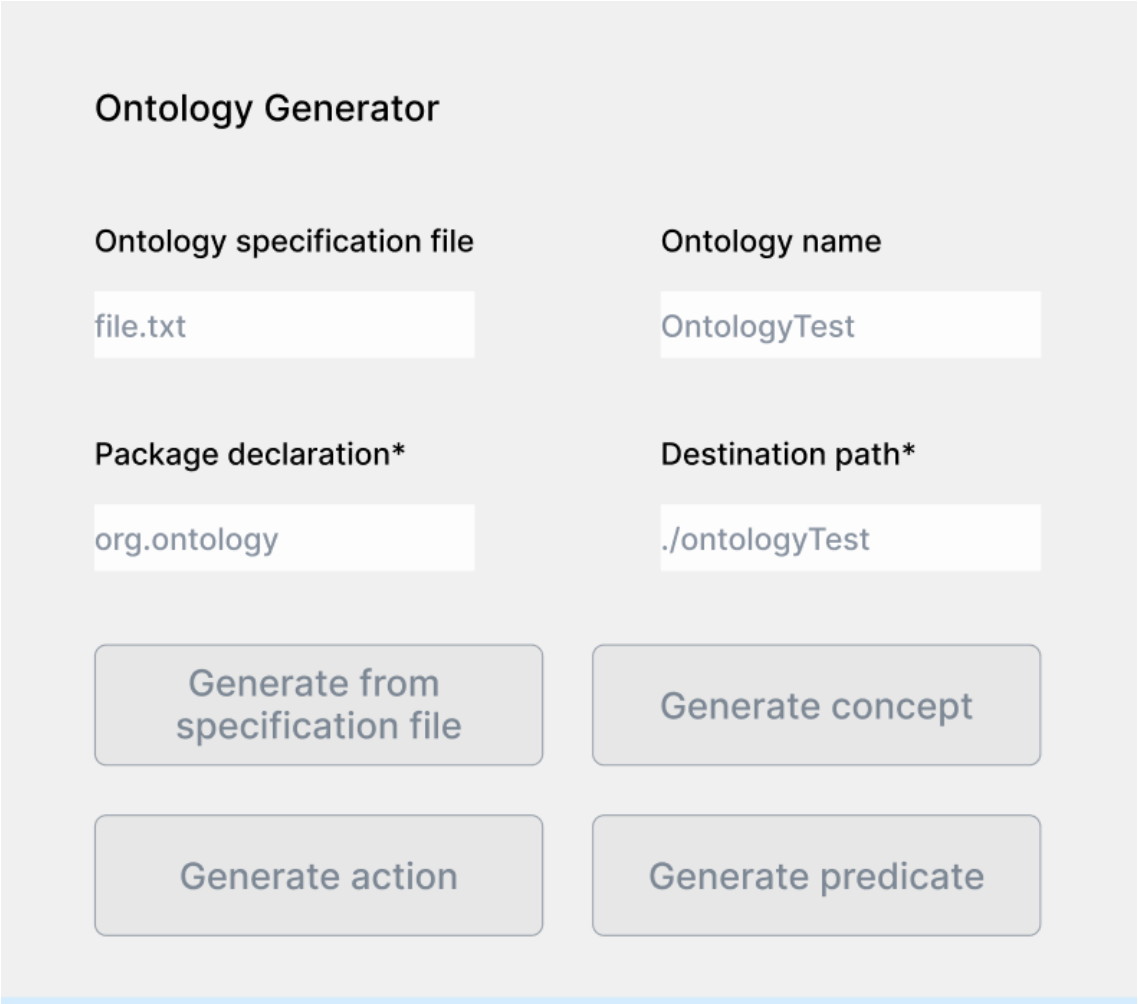


Figure 5.2: Ontology Generator interface - Main screen first version

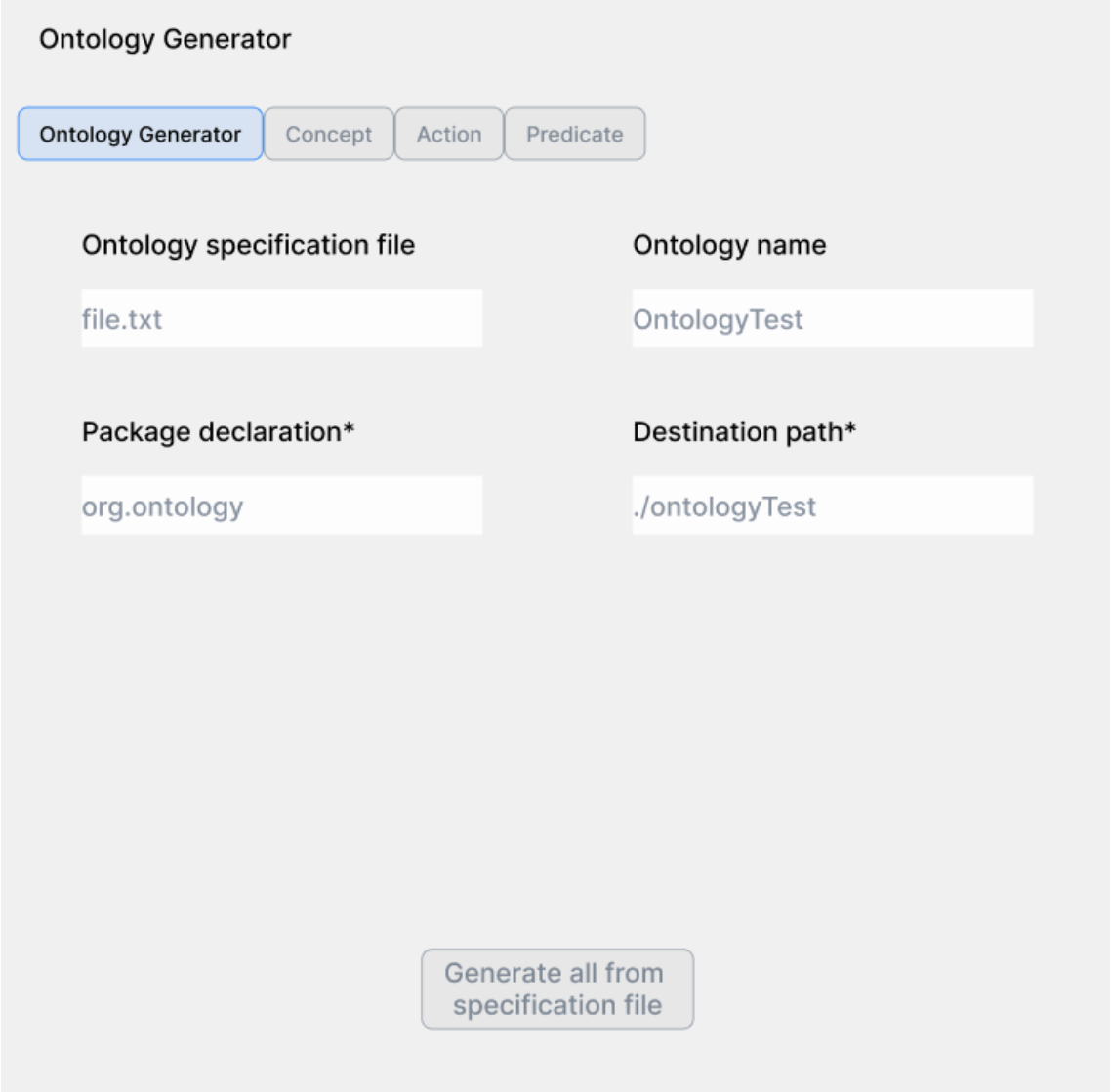


Figure 5.3: Ontology Generator interface - Main screen second version

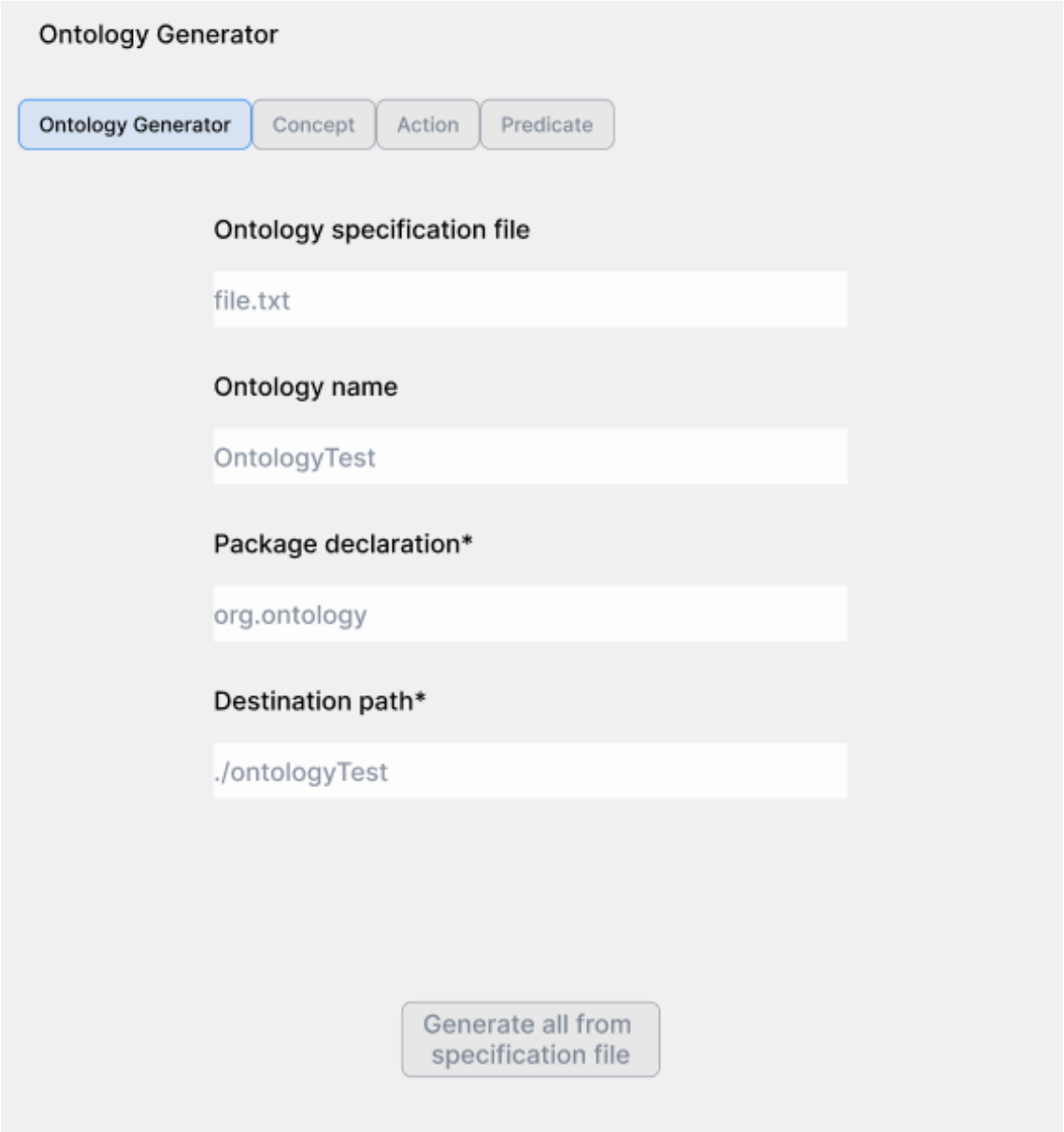


Figure 5.4: Ontology Generator interface - Main screen third version

- **New Concept screen:** Screen that contains the form to add a new concept to the ontology. It has one input for the name of the concept and a variable number of inputs to specify the attributes of the concept (name and type of each attribute).

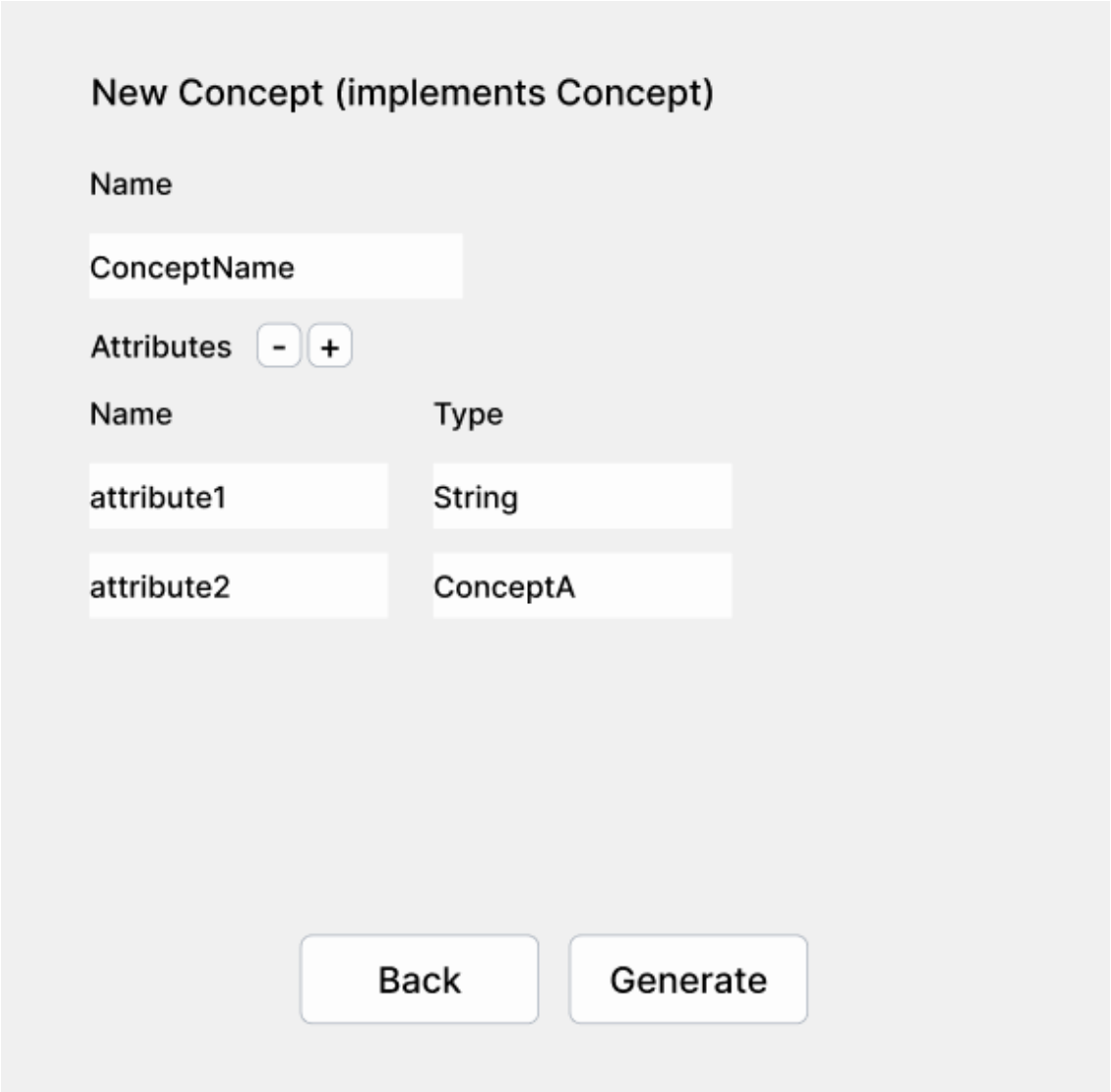


Figure 5.5: Ontology Generator interface - New Concept screen first version

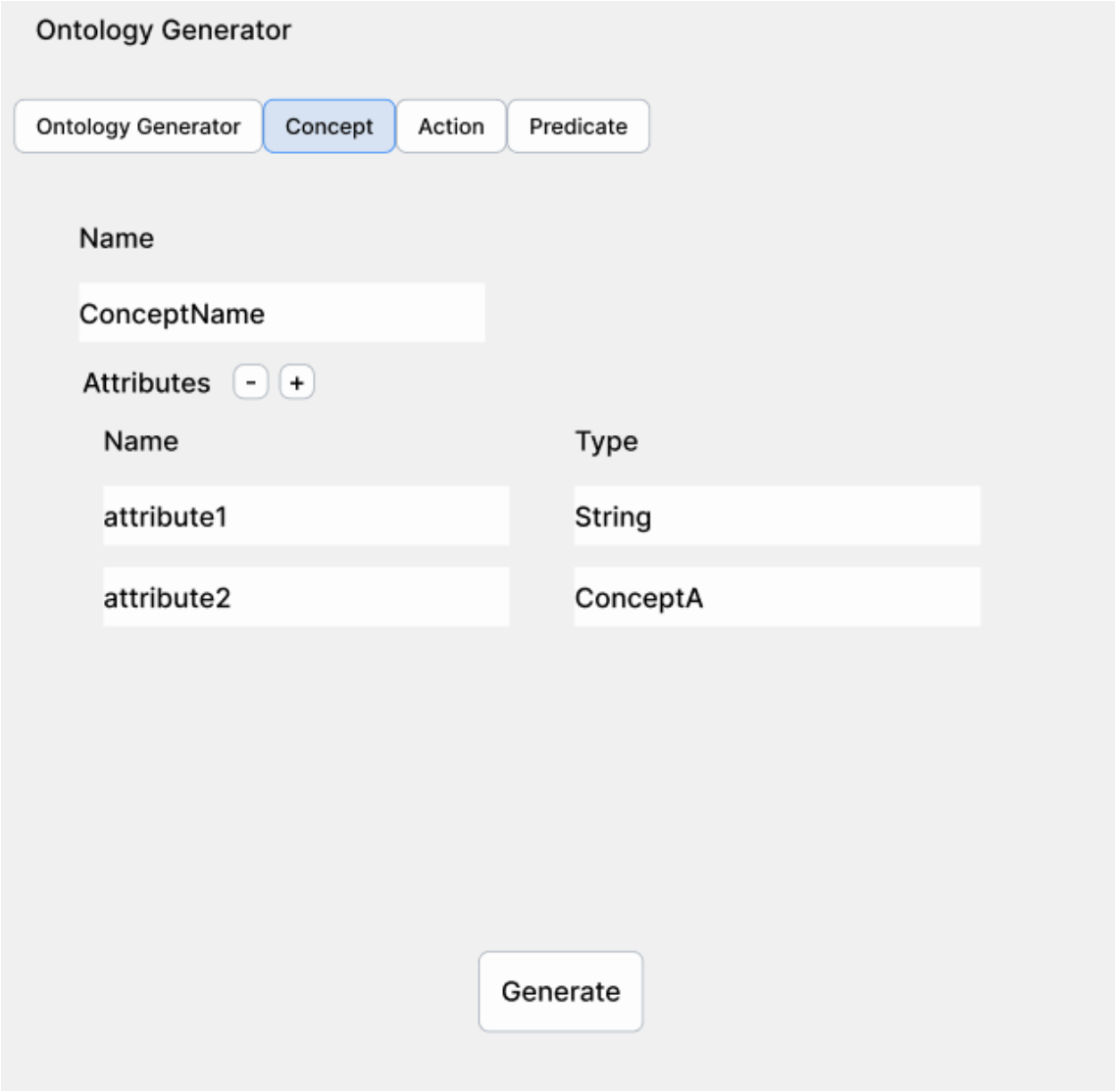


Figure 5.6: Ontology Generator interface - New Concept screen second version

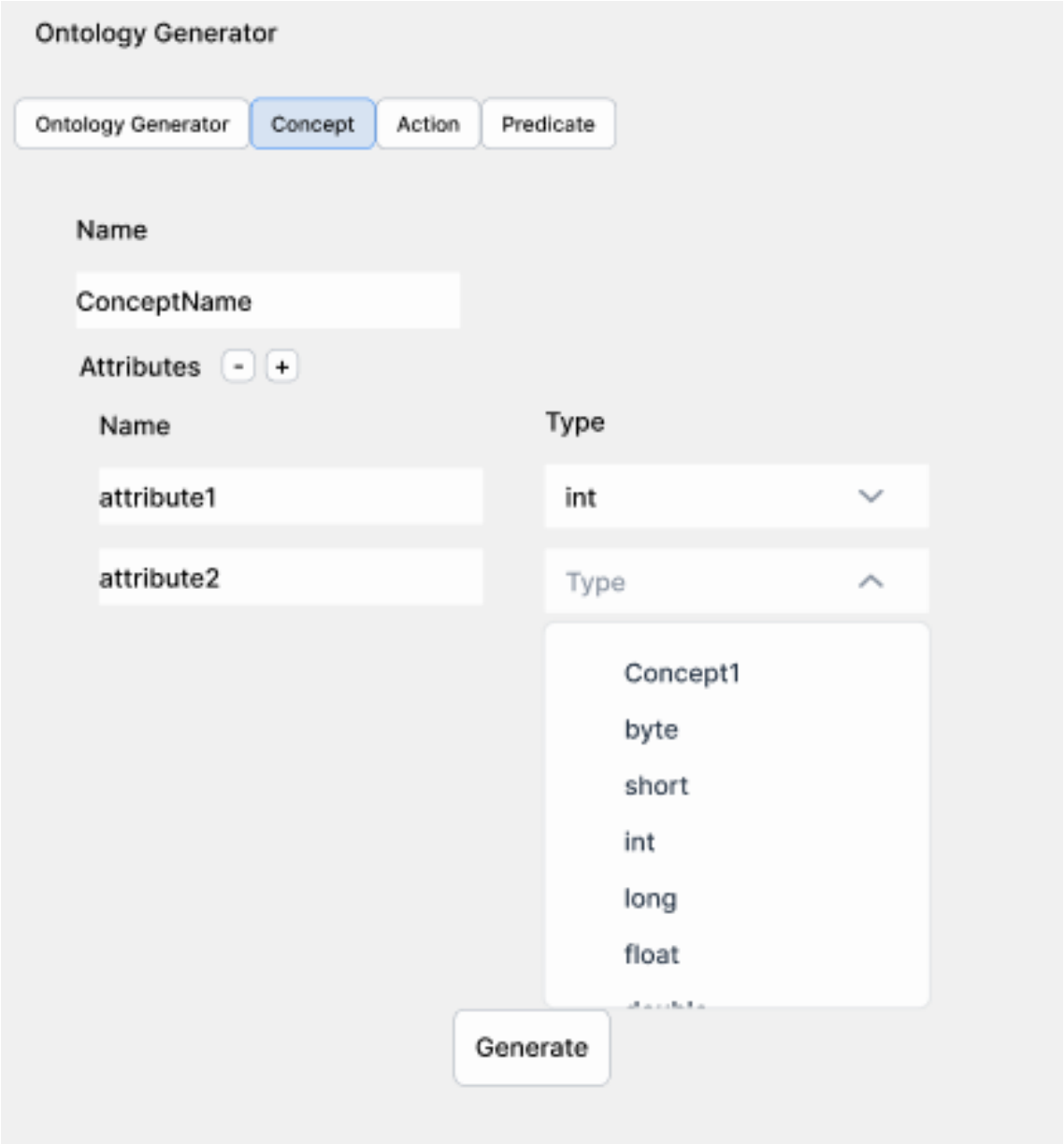


Figure 5.7: Ontology Generator interface - New Concept screen third version

- **New Action screen:** Screen that contains the form to add a new action to the ontology. It has the same structure as the previous one.

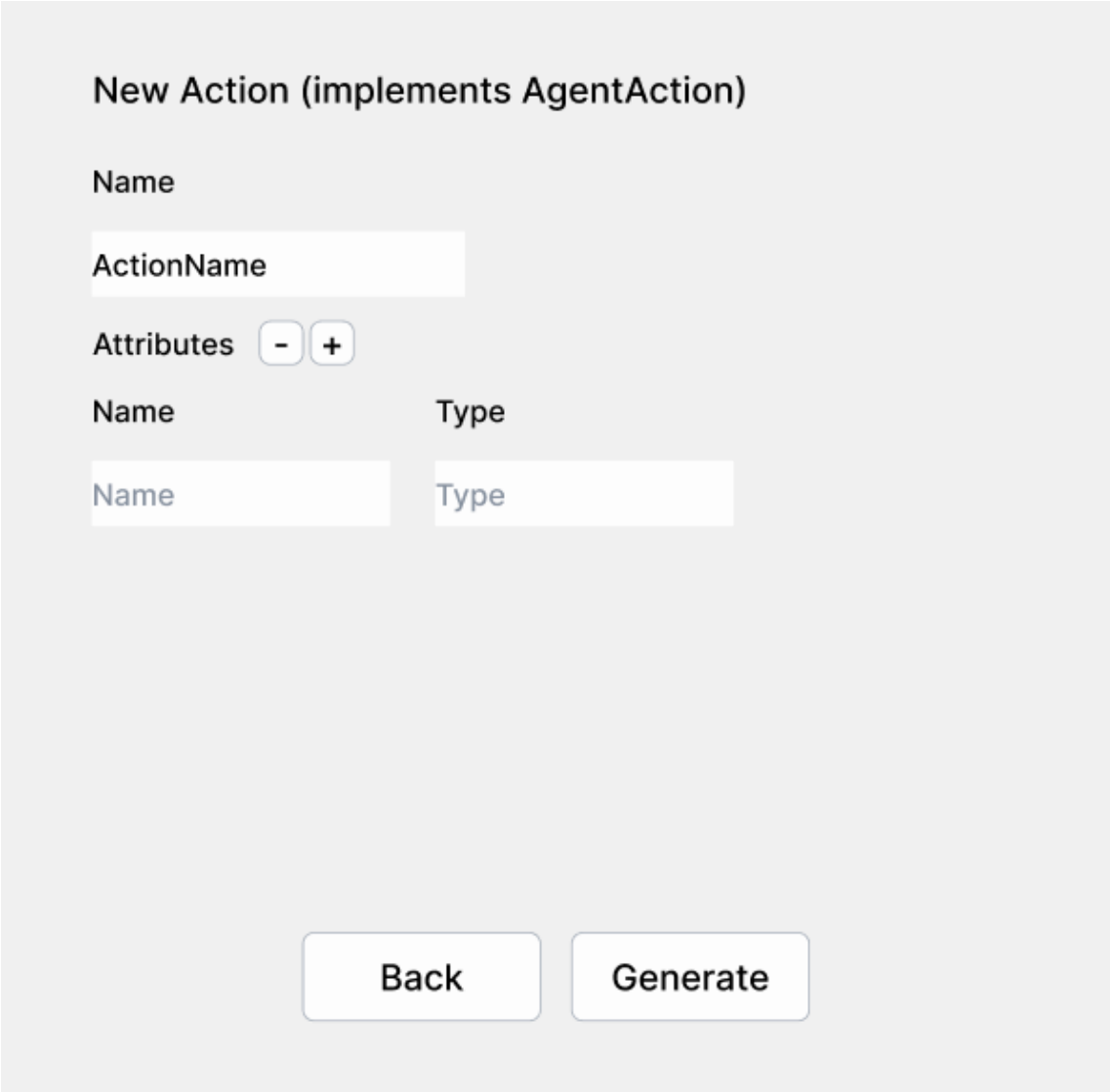


Figure 5.8: Ontology Generator interface - New Action screen first version

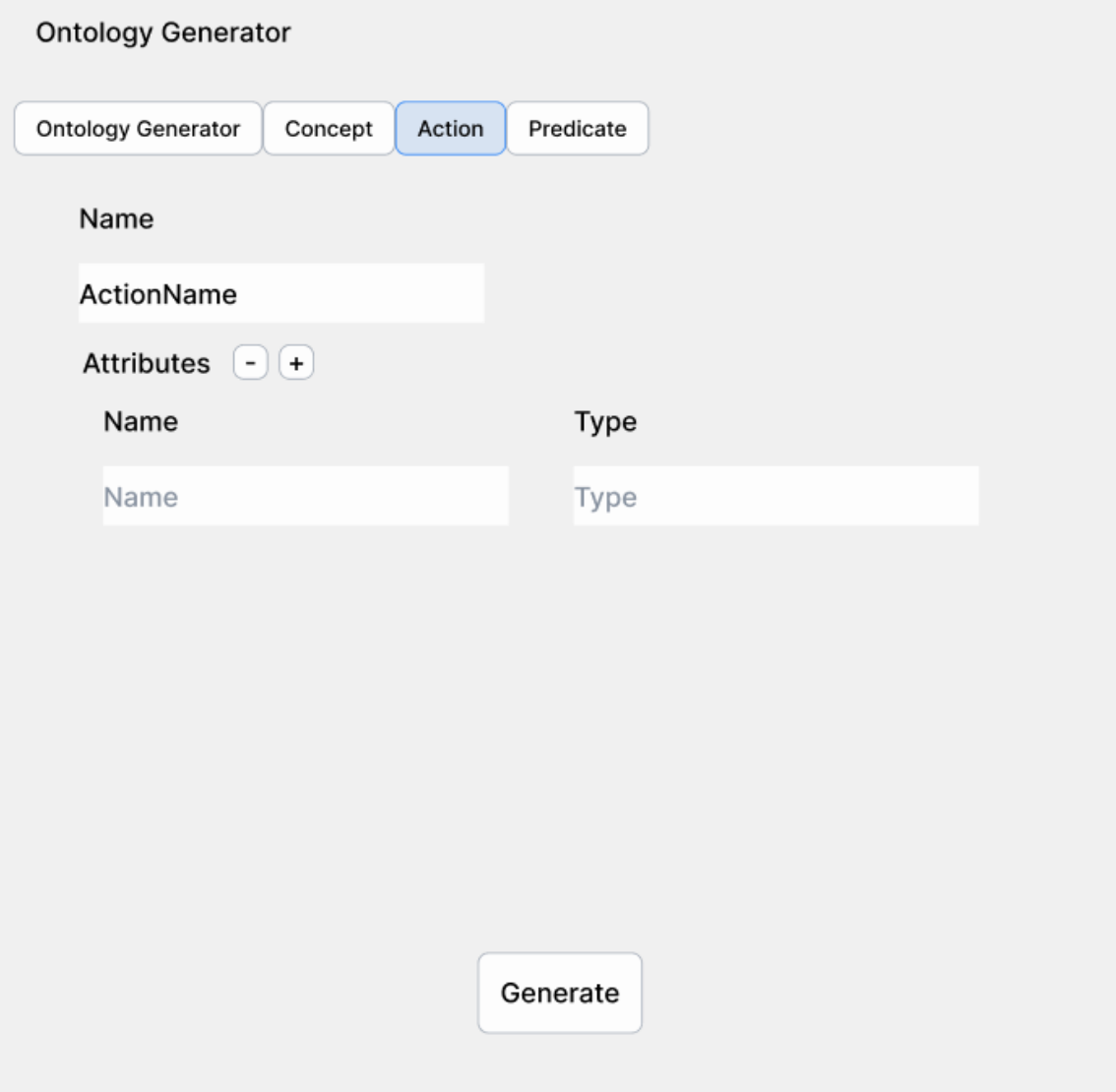


Figure 5.9: Ontology Generator interface - New Action screen second version

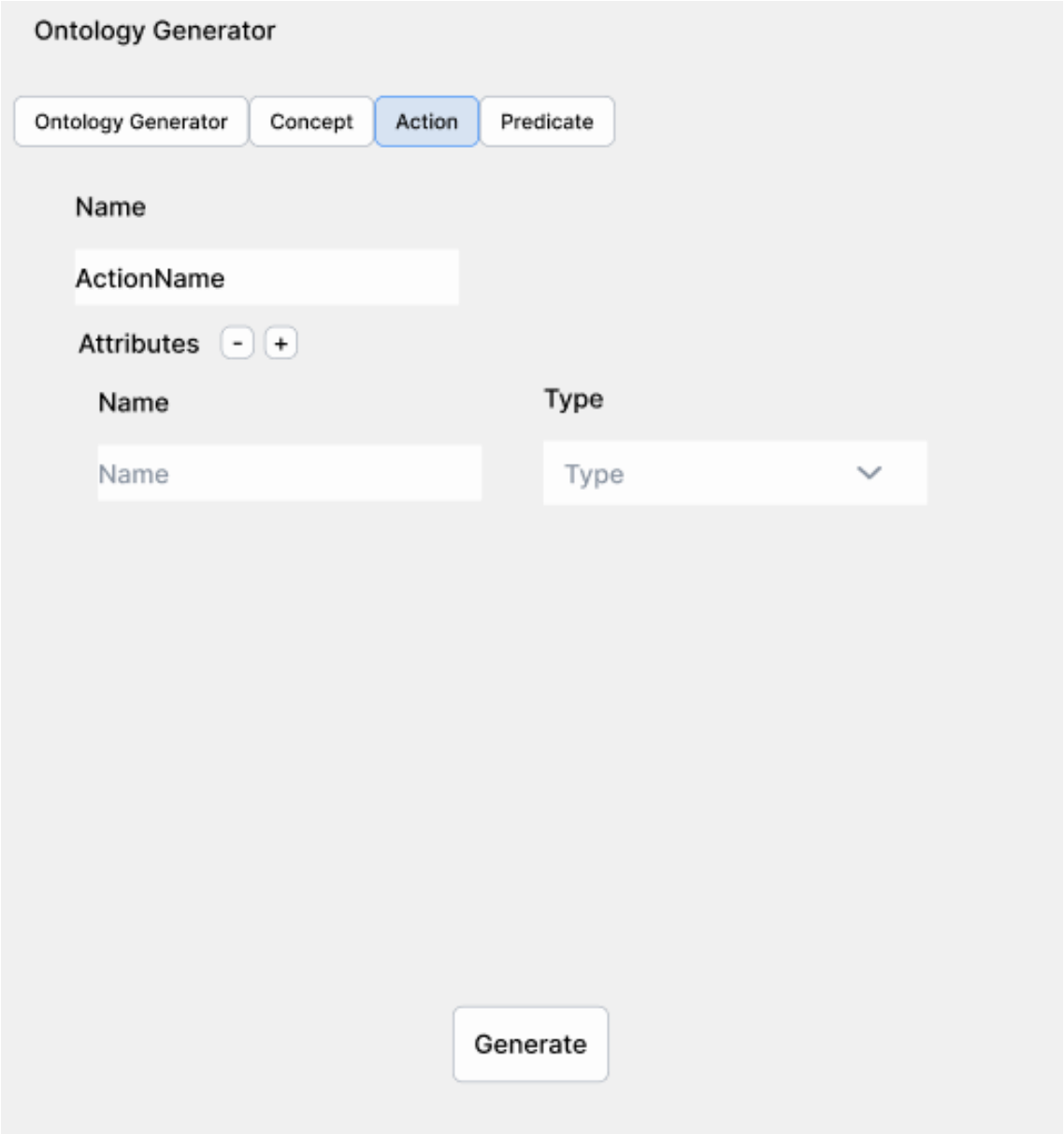


Figure 5.10: Ontology Generator interface - New Action screen third version

- **New Predicate screen:** Screen that contains the form to add a new predicate to the ontology. It has the same structure as the previous one.

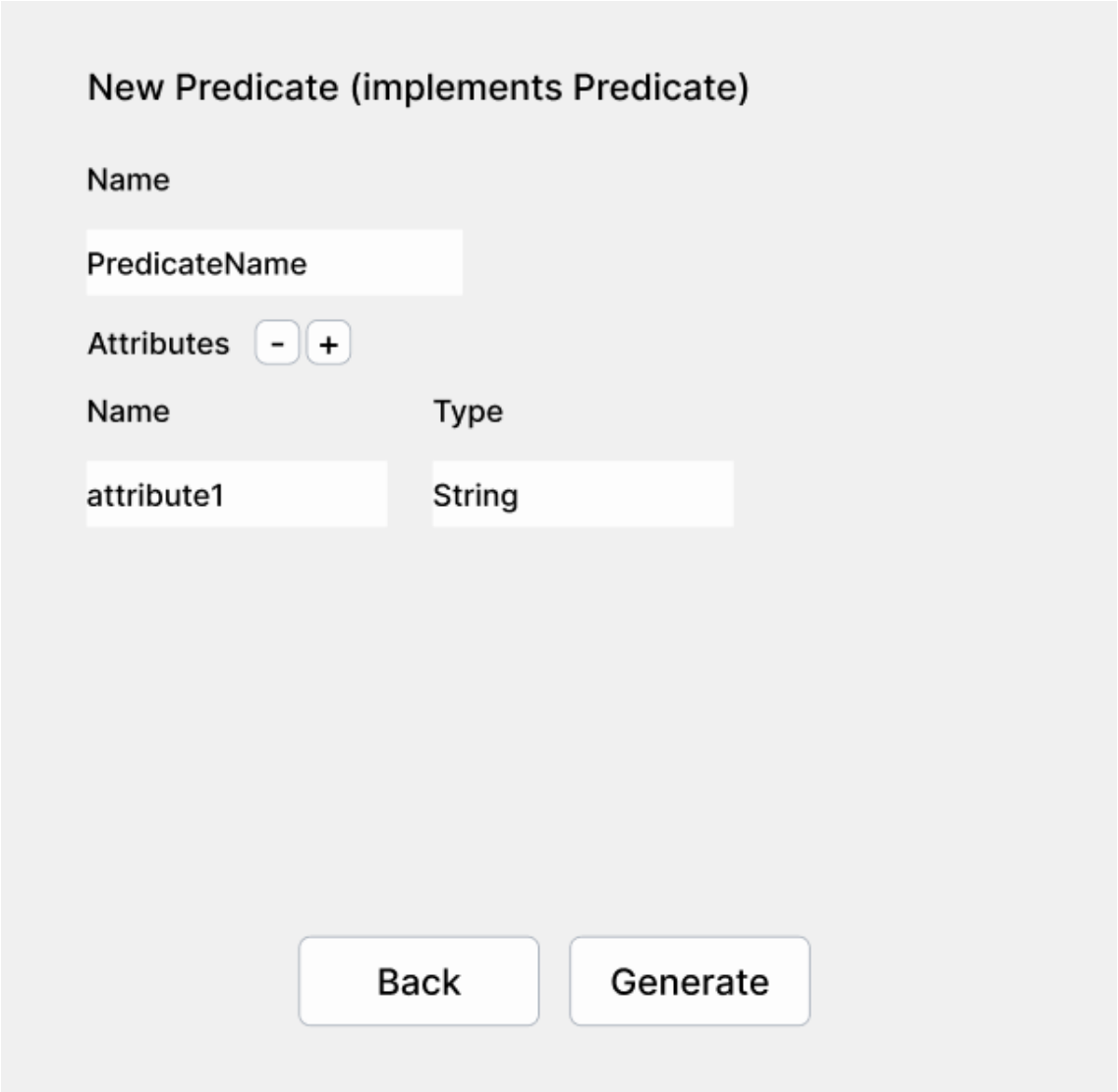


Figure 5.11: Ontology Generator interface - New Predicate screen first version

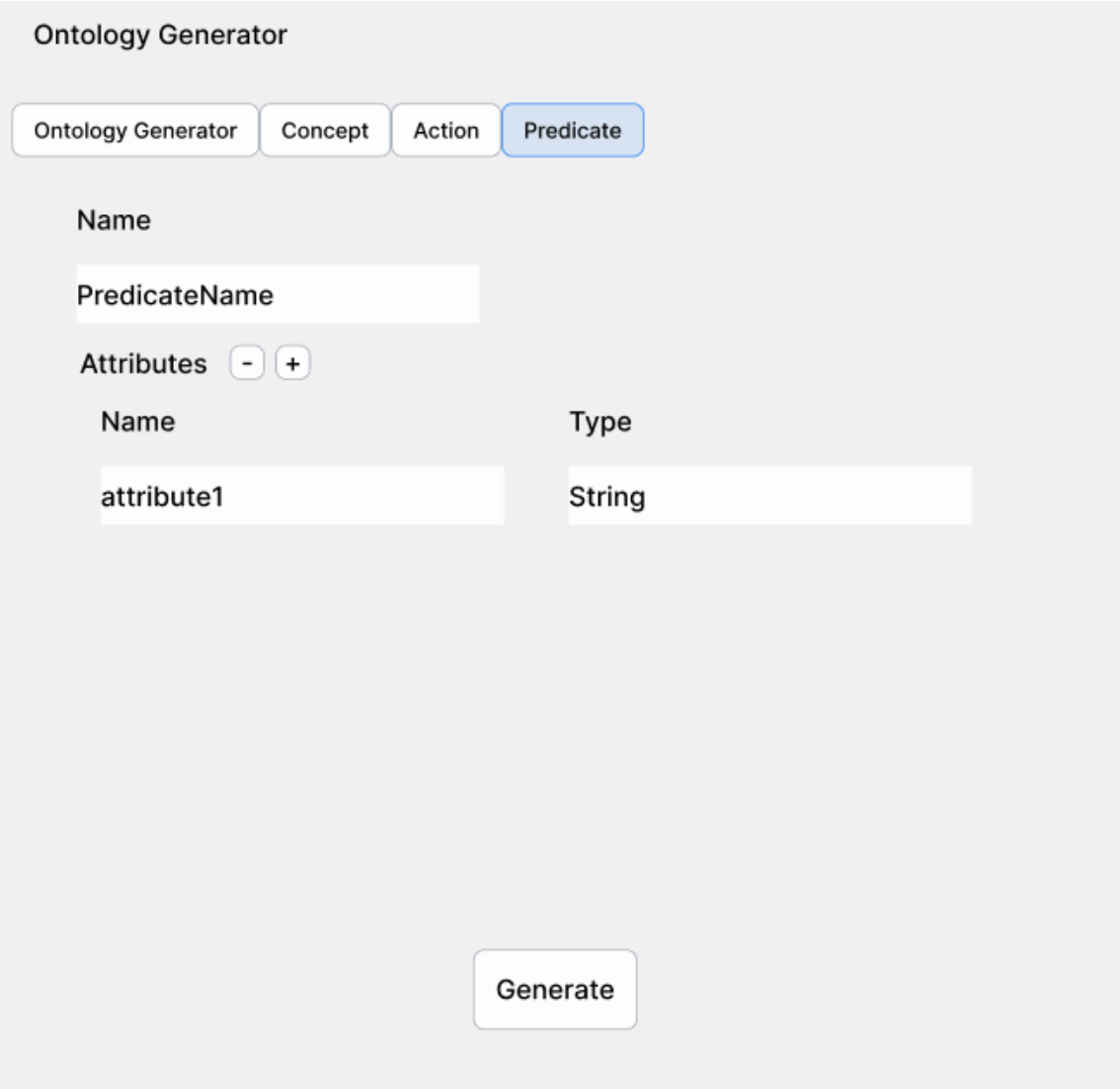


Figure 5.12: Ontology Generator interface - New Predicate screen second version

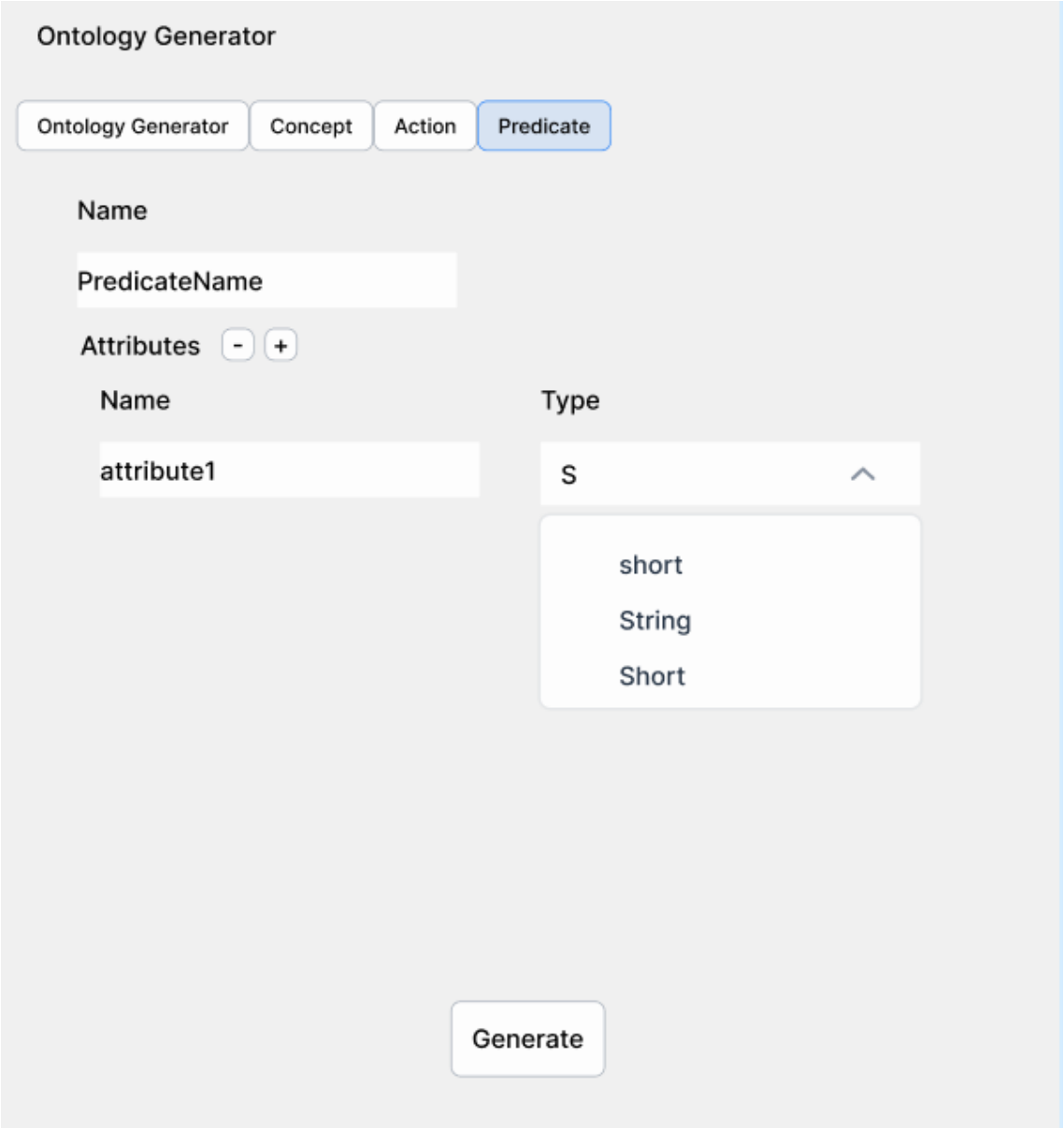


Figure 5.13: Ontology Generator interface - New Predicate screen third version

- **Success main screen:** This screen is the Main screen with a banner that indicates the result of the operations in case of success.

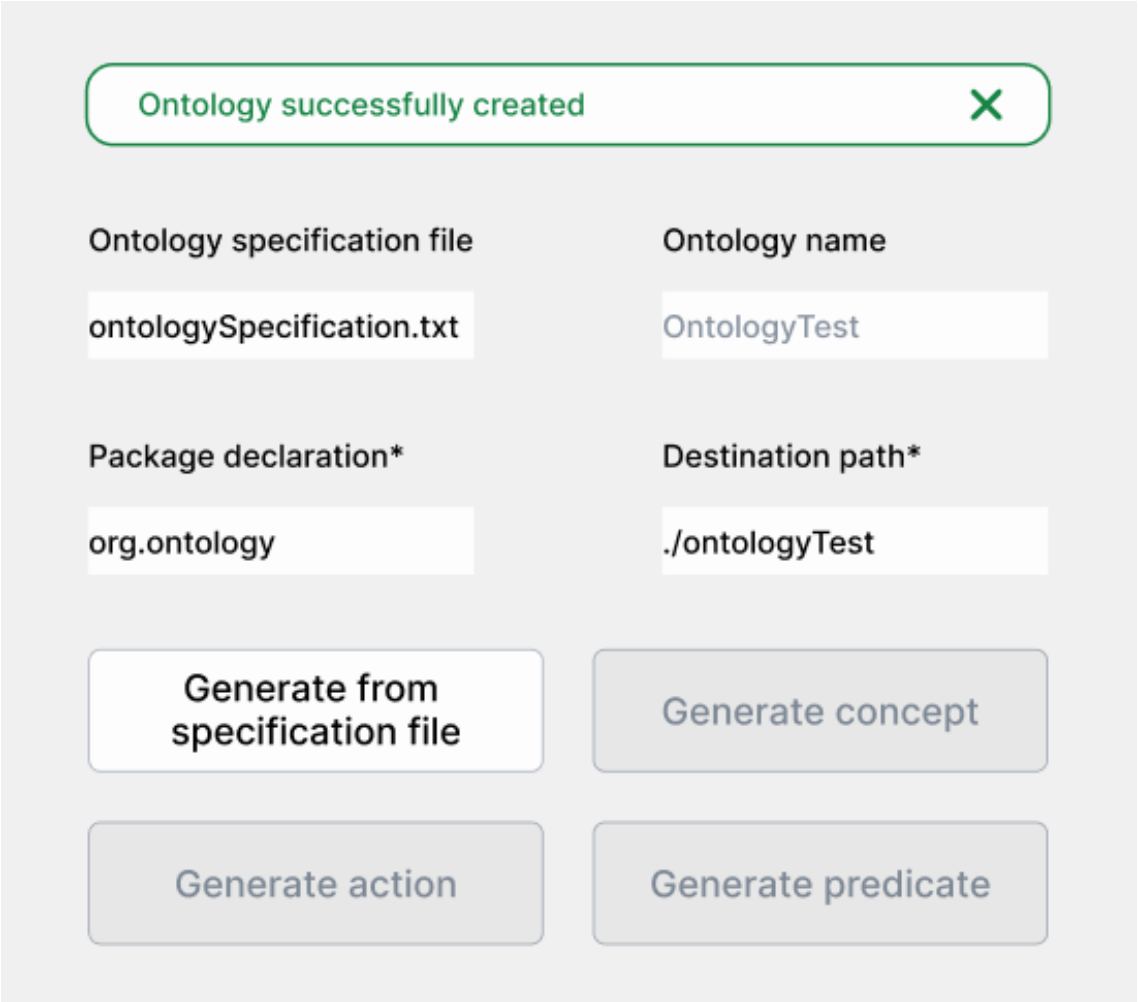


Figure 5.14: Ontology Generator interface - Success main screen first version

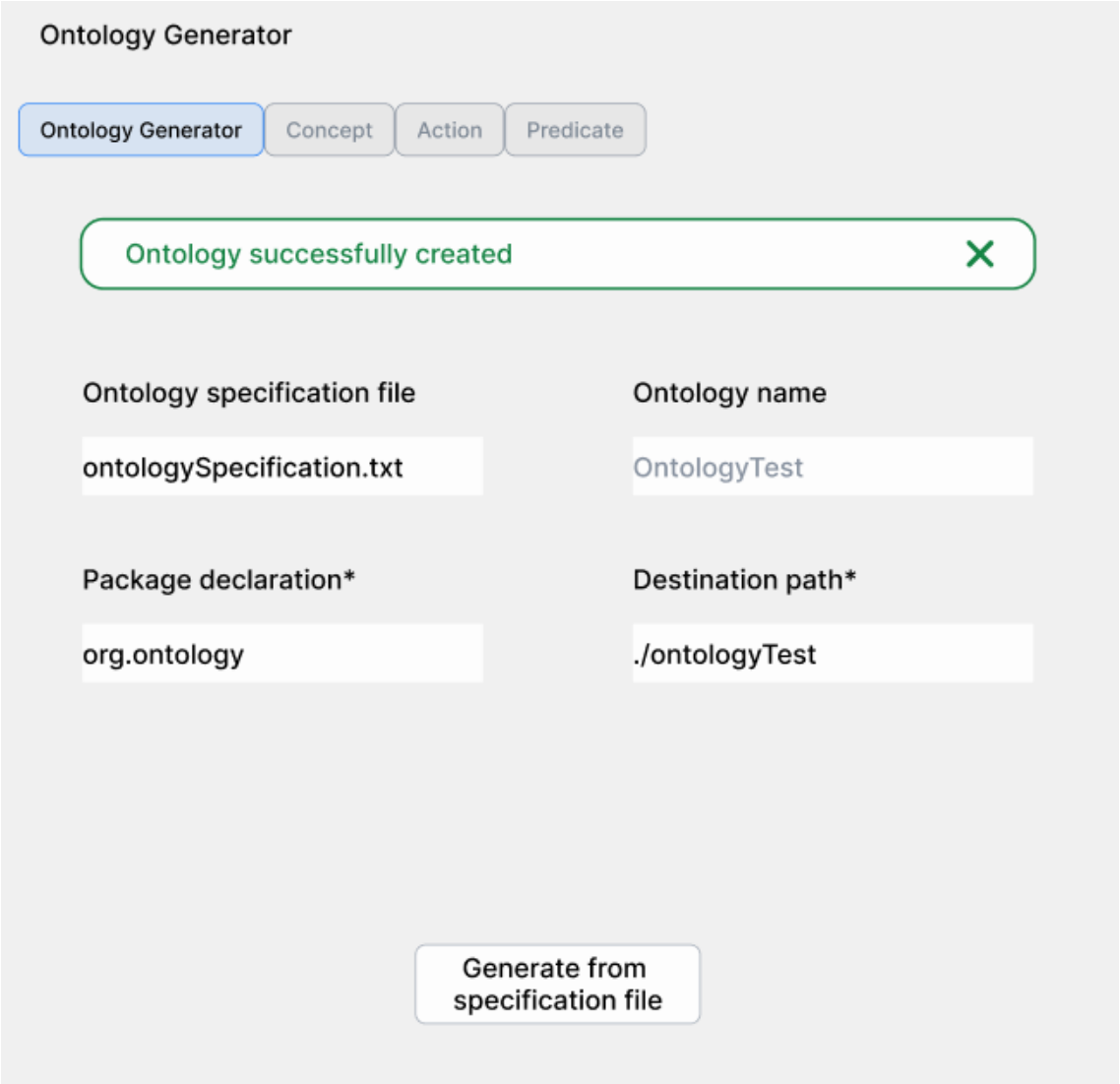


Figure 5.15: Ontology Generator interface - Success main screen second version

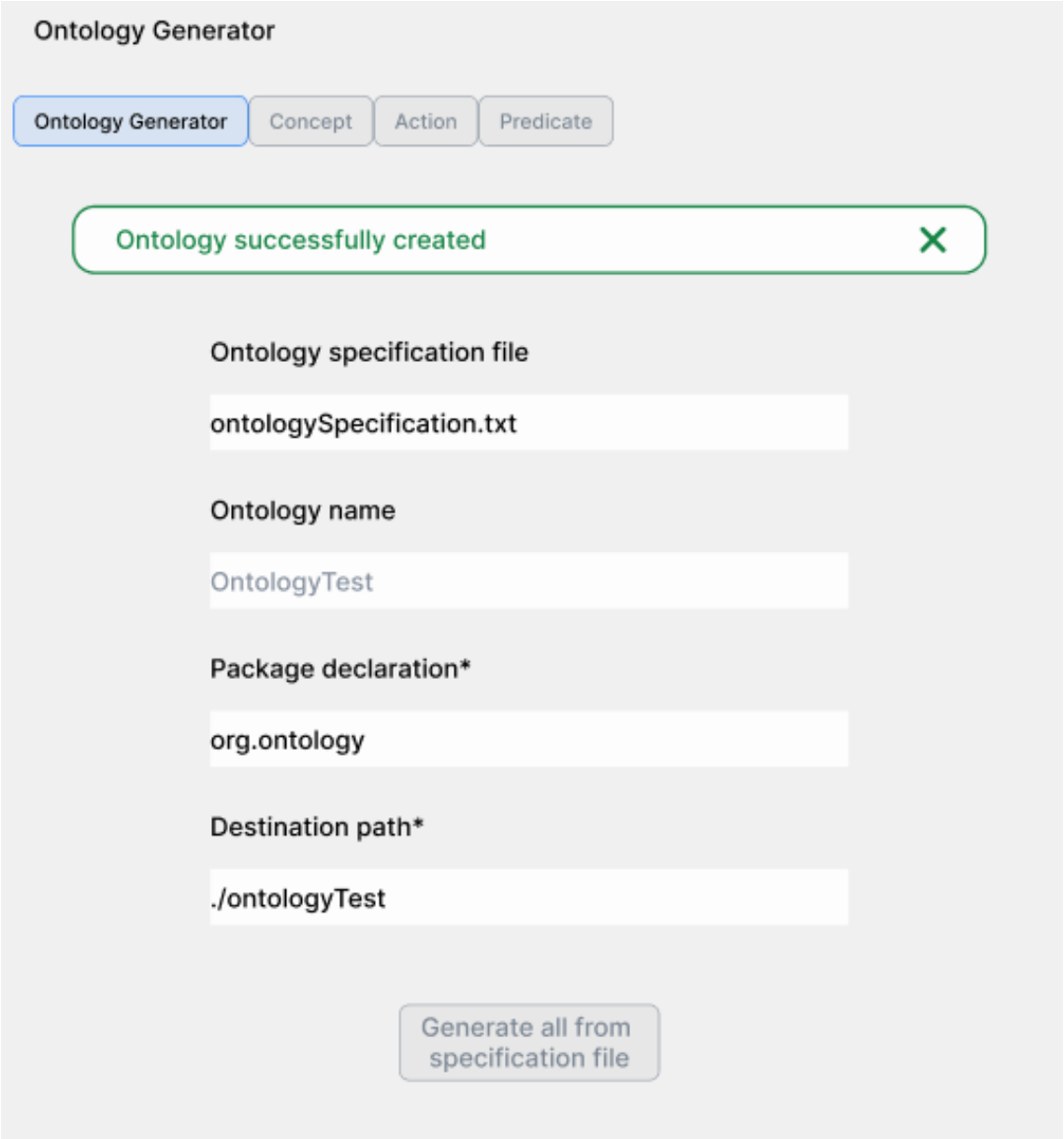


Figure 5.16: Ontology Generator interface - Success main screen third version

- **Success New Concept, Action and Predicate screen:** This screen is the New Concept/Action/Predicate screen with a banner that indicates the result of the operations in case of success (it changes the message and the tab according to the operation). This screen did not exists in the first version because the success of these operations was represented with the previous screen.

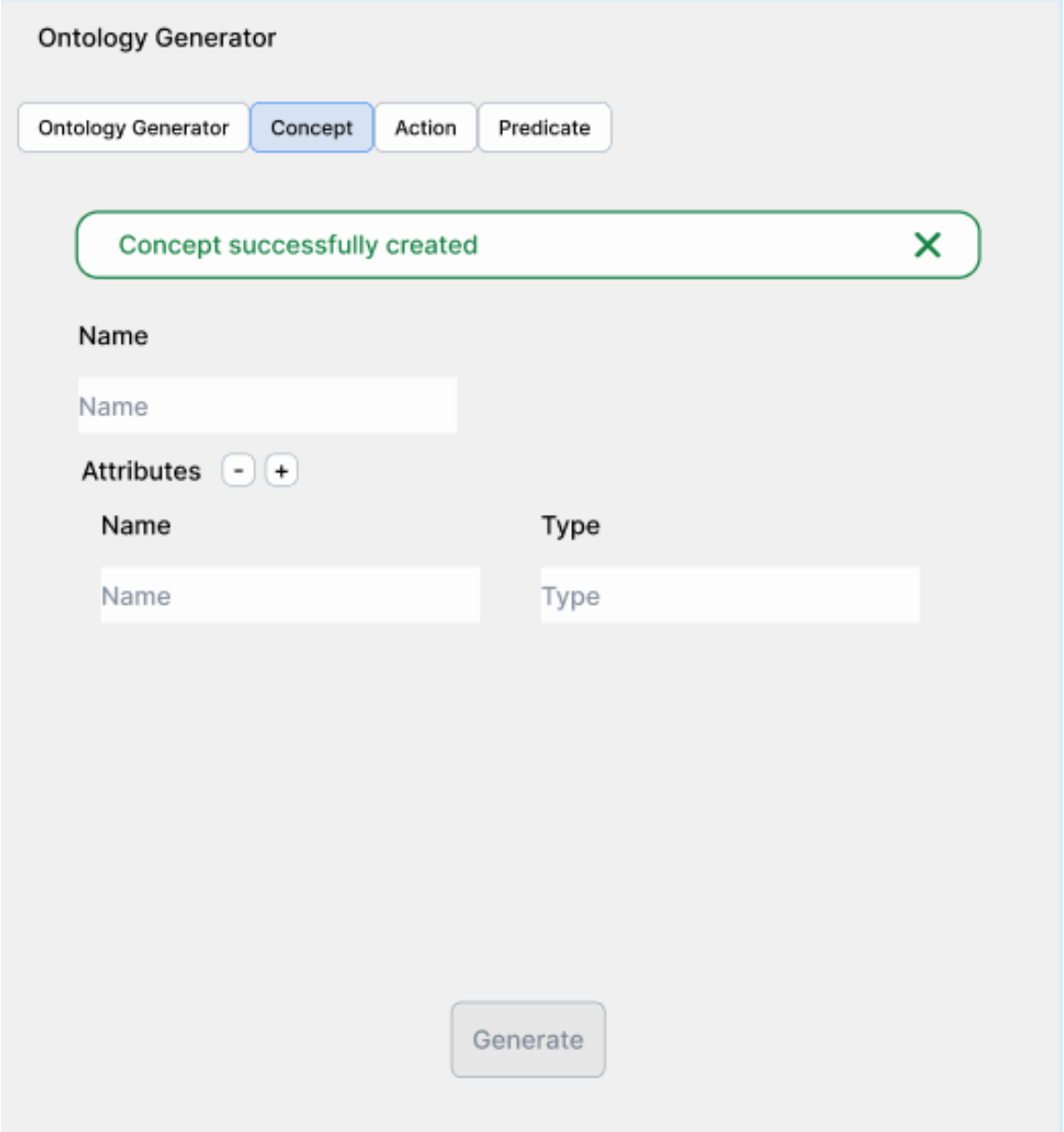


Figure 5.17: Ontology Generator interface - Success New Concept, Action and Predicate second version

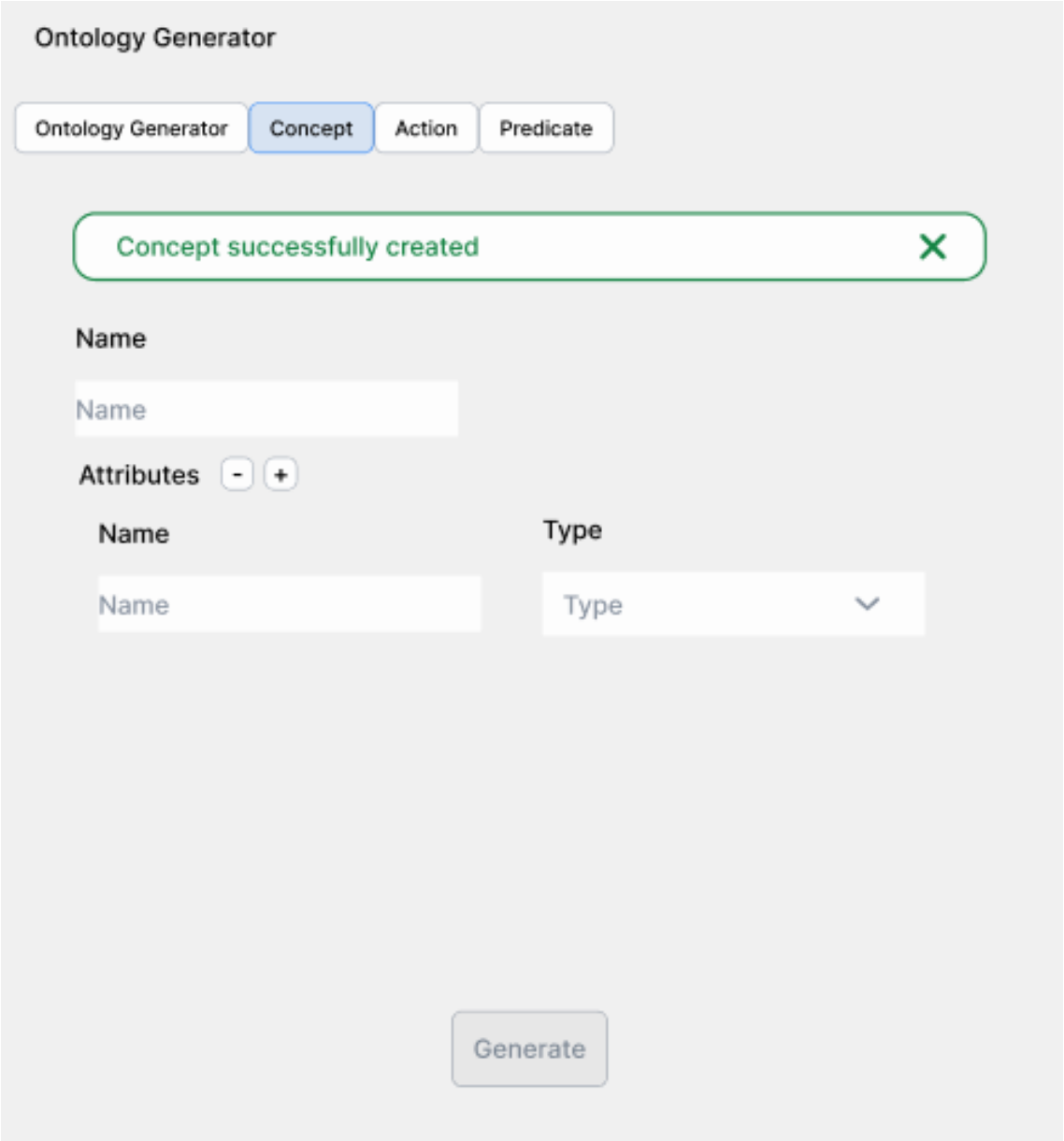


Figure 5.18: Ontology Generator interface - Success New Concept, Action and Predicate third version

- **Error in specification file:** This screen is the Main screen with a banner that indicates the result of the operations in case of error.

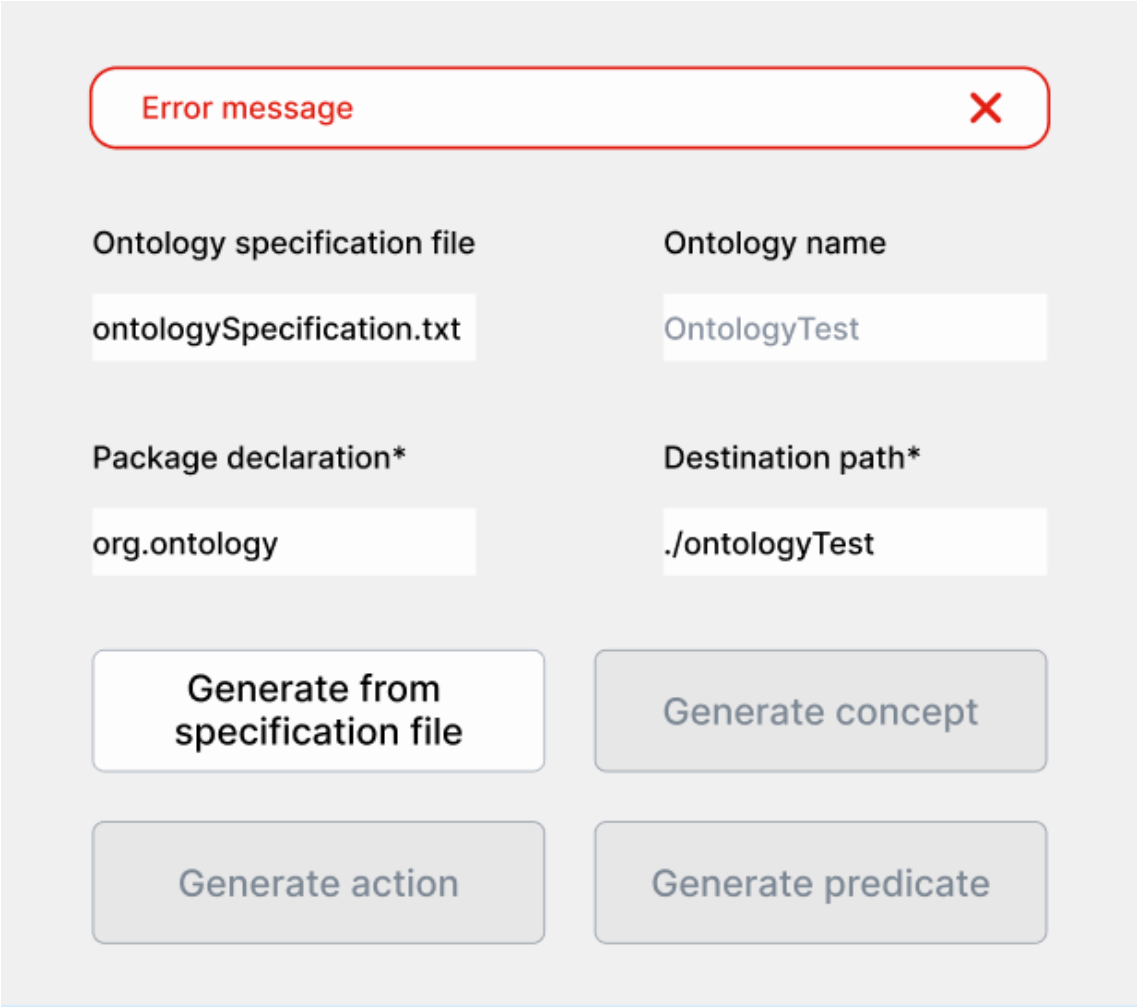


Figure 5.19: Ontology Generator interface - Error in specification file first version

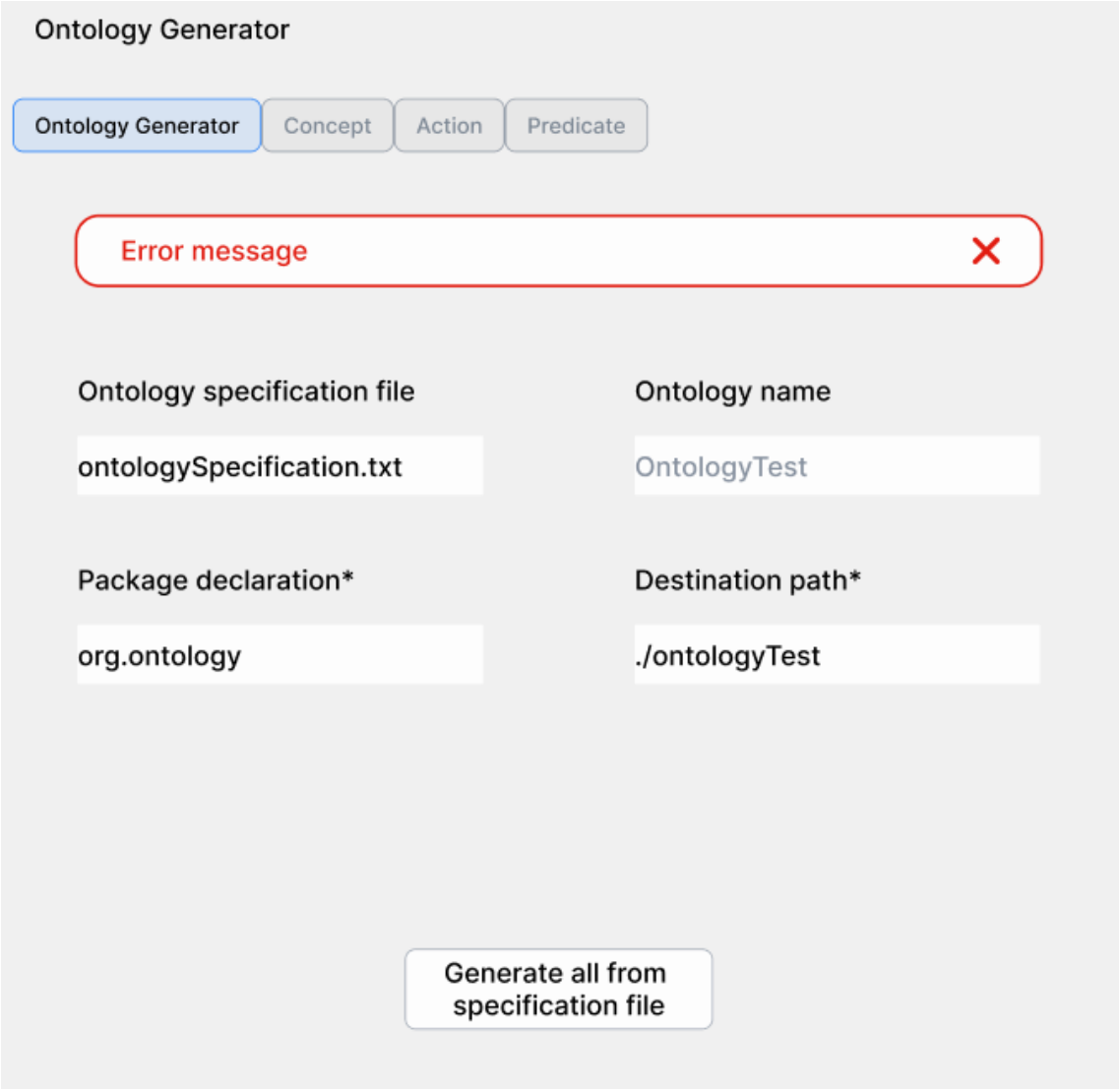


Figure 5.20: Ontology Generator interface - Error in specification file second version

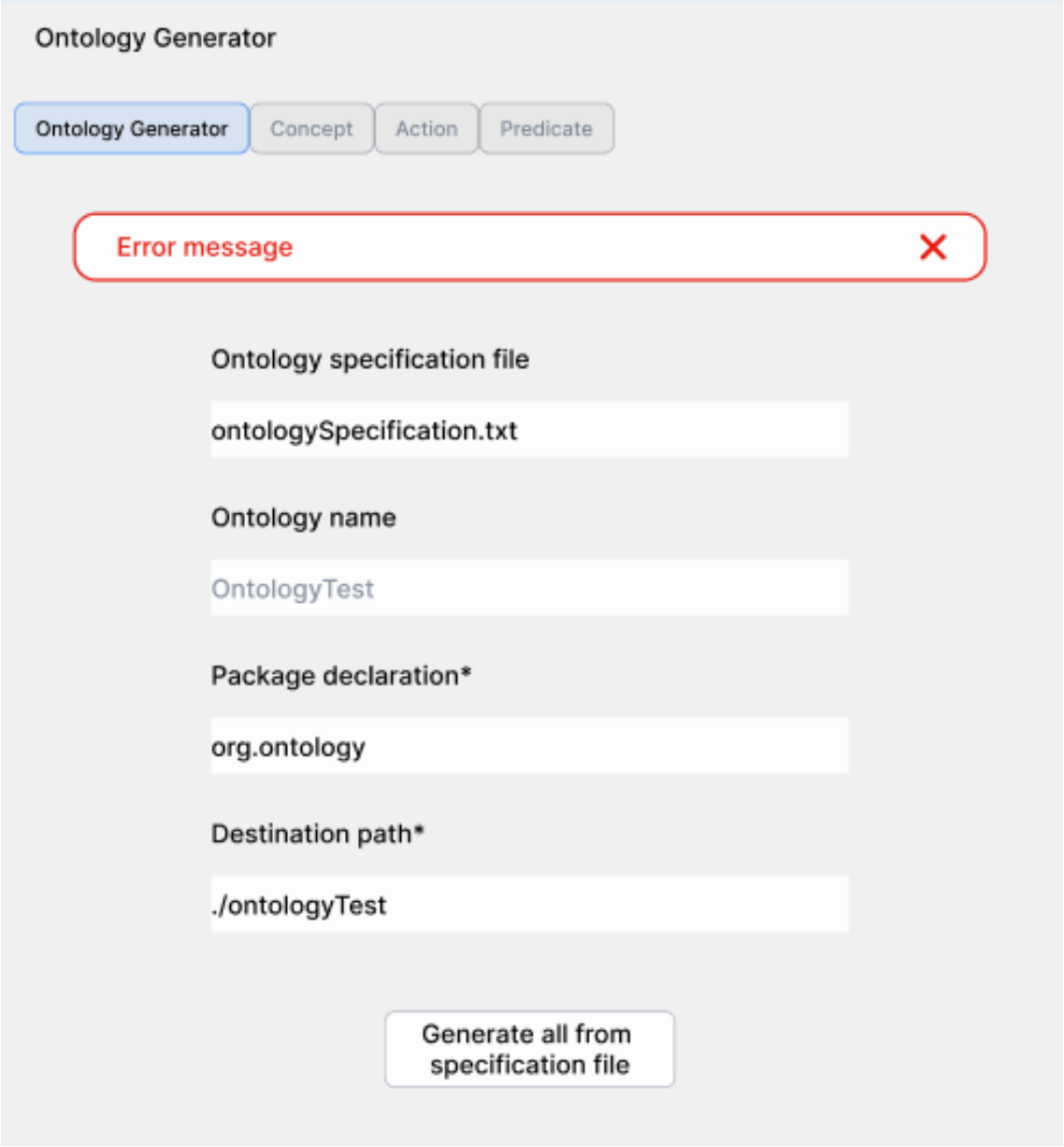


Figure 5.21: Ontology Generator interface - Error in specification file third version

- **Error file not found:** This screen is the Main screen when the Ontology Generator cannot find the specification file indicated.

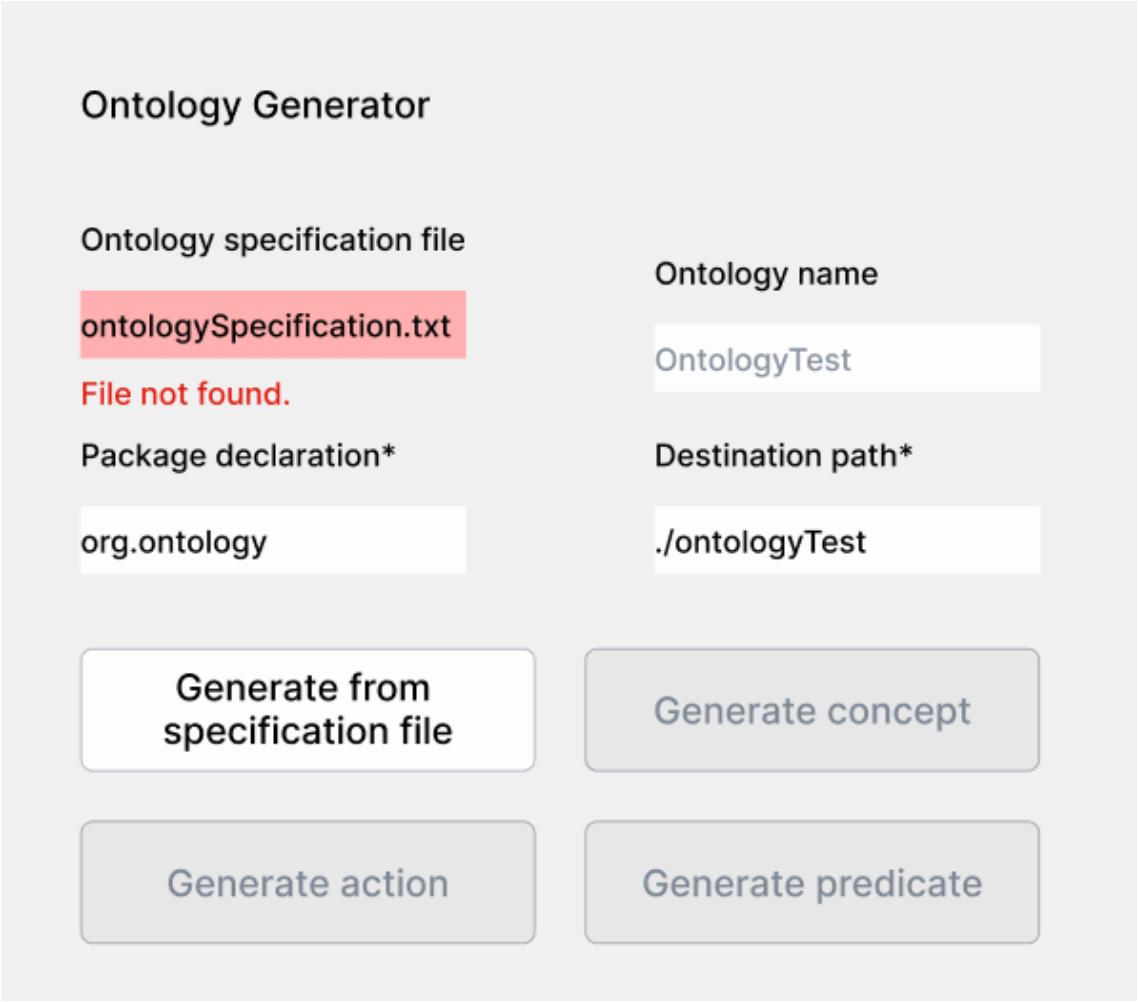


Figure 5.22: Ontology Generator interface - Error file not found first version

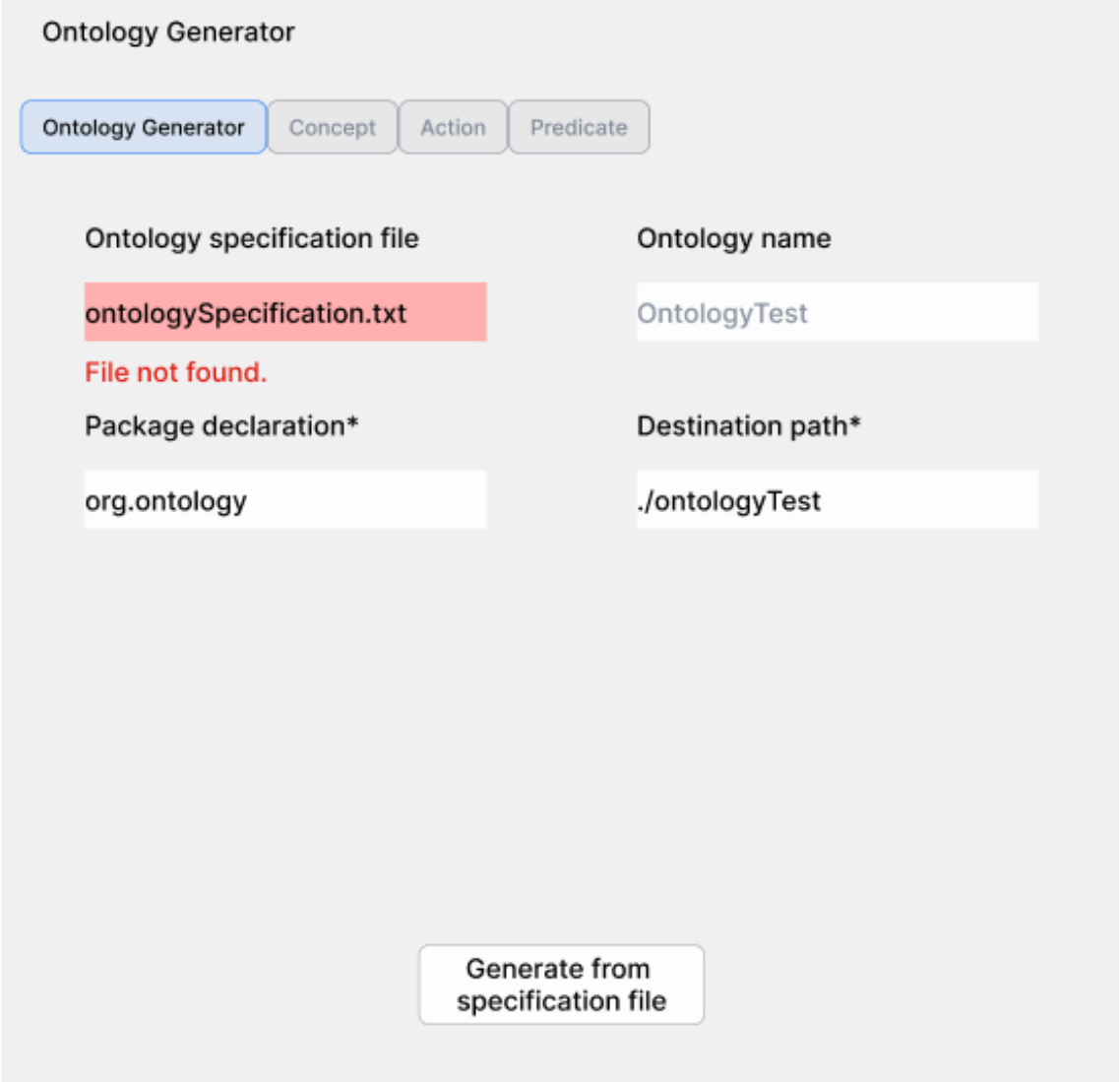


Figure 5.23: Ontology Generator interface - Error file not found second version

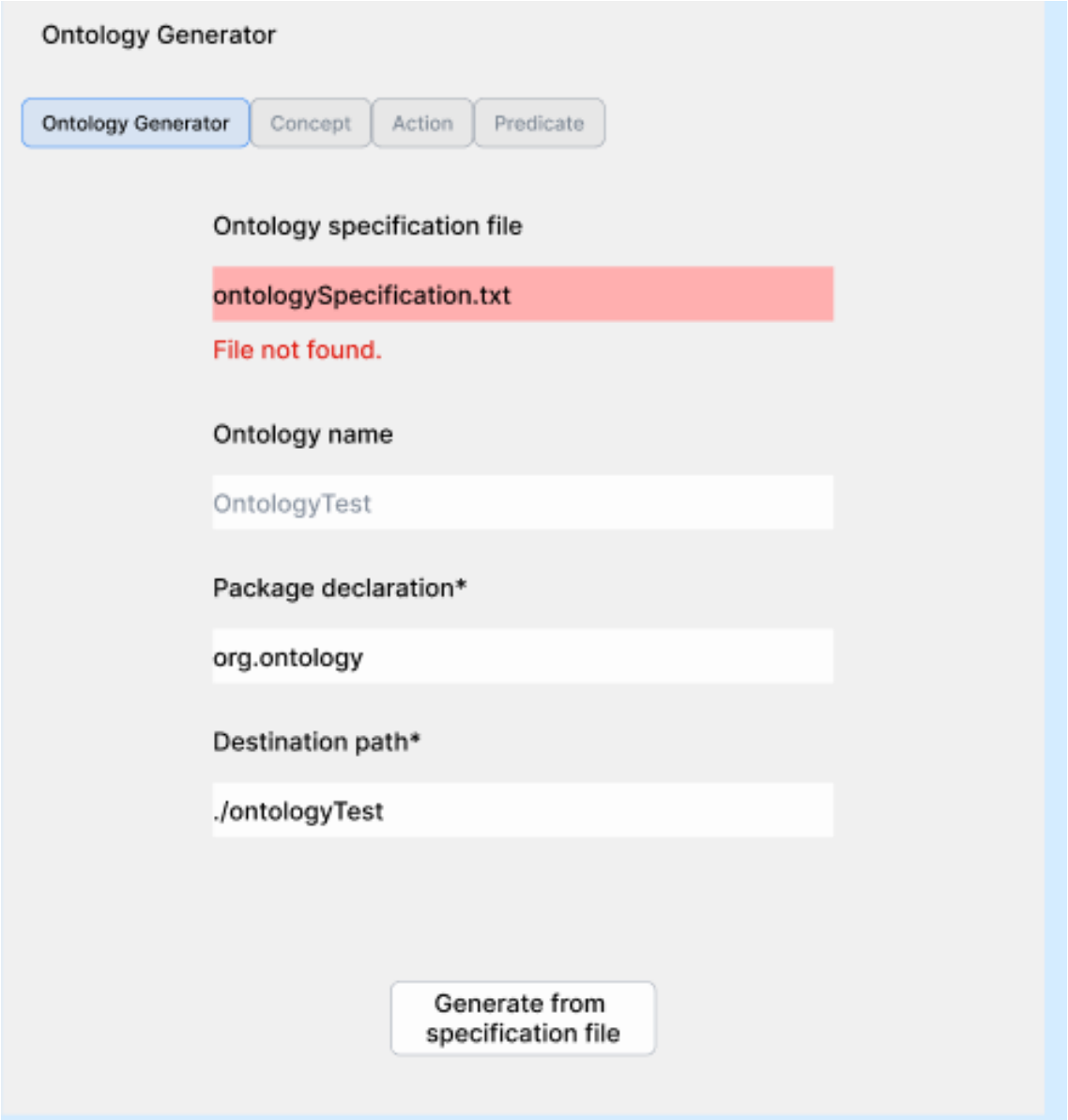


Figure 5.24: Ontology Generator interface - Error file not found third version

- **Error generating concept/action/predicate:** This screen is the New concept/action/predicate screen with a banner that indicates the result of the generation in case of error.

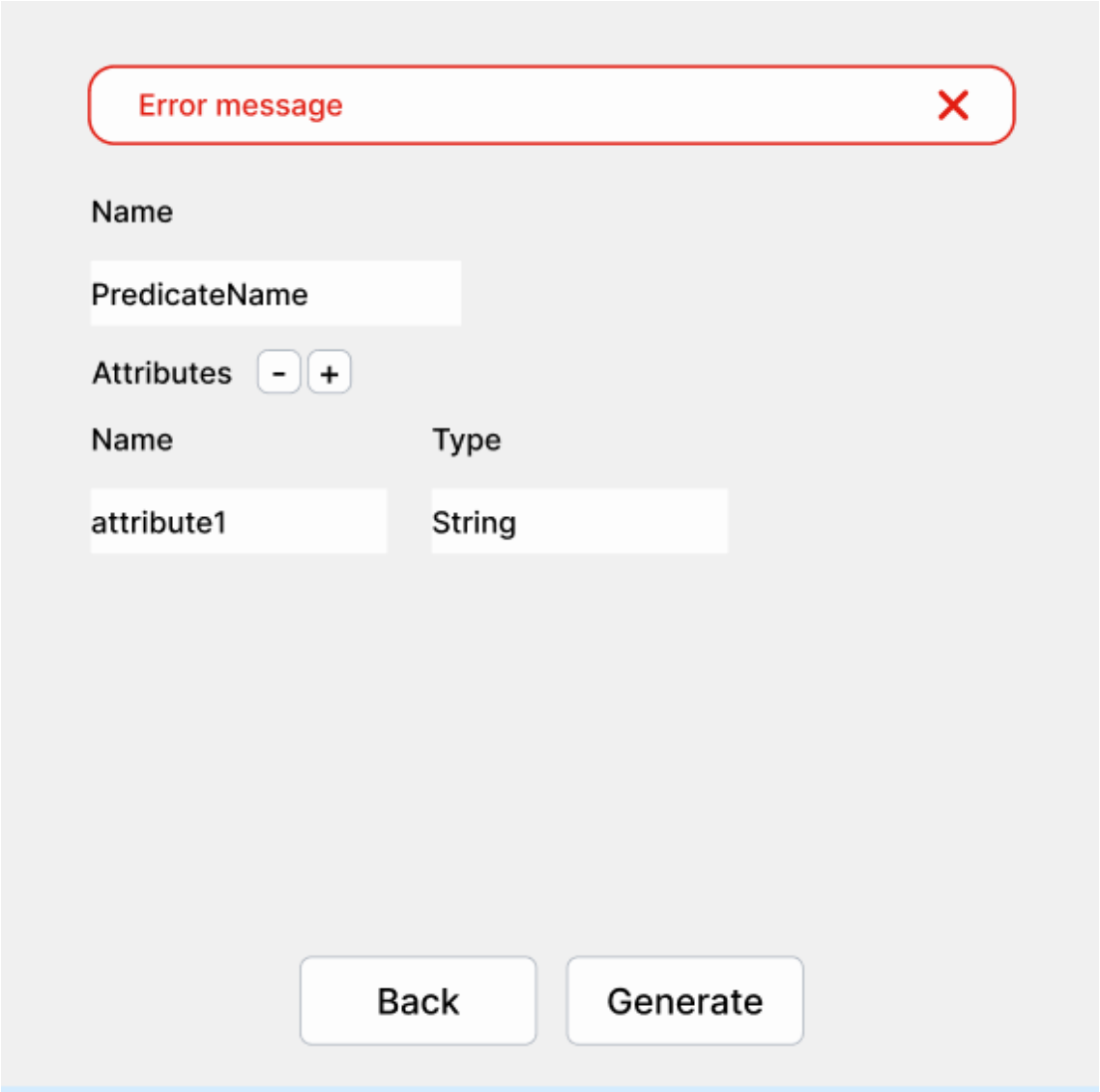


Figure 5.25: Ontology Generator interface - Error generating concept/action/-predicate first version

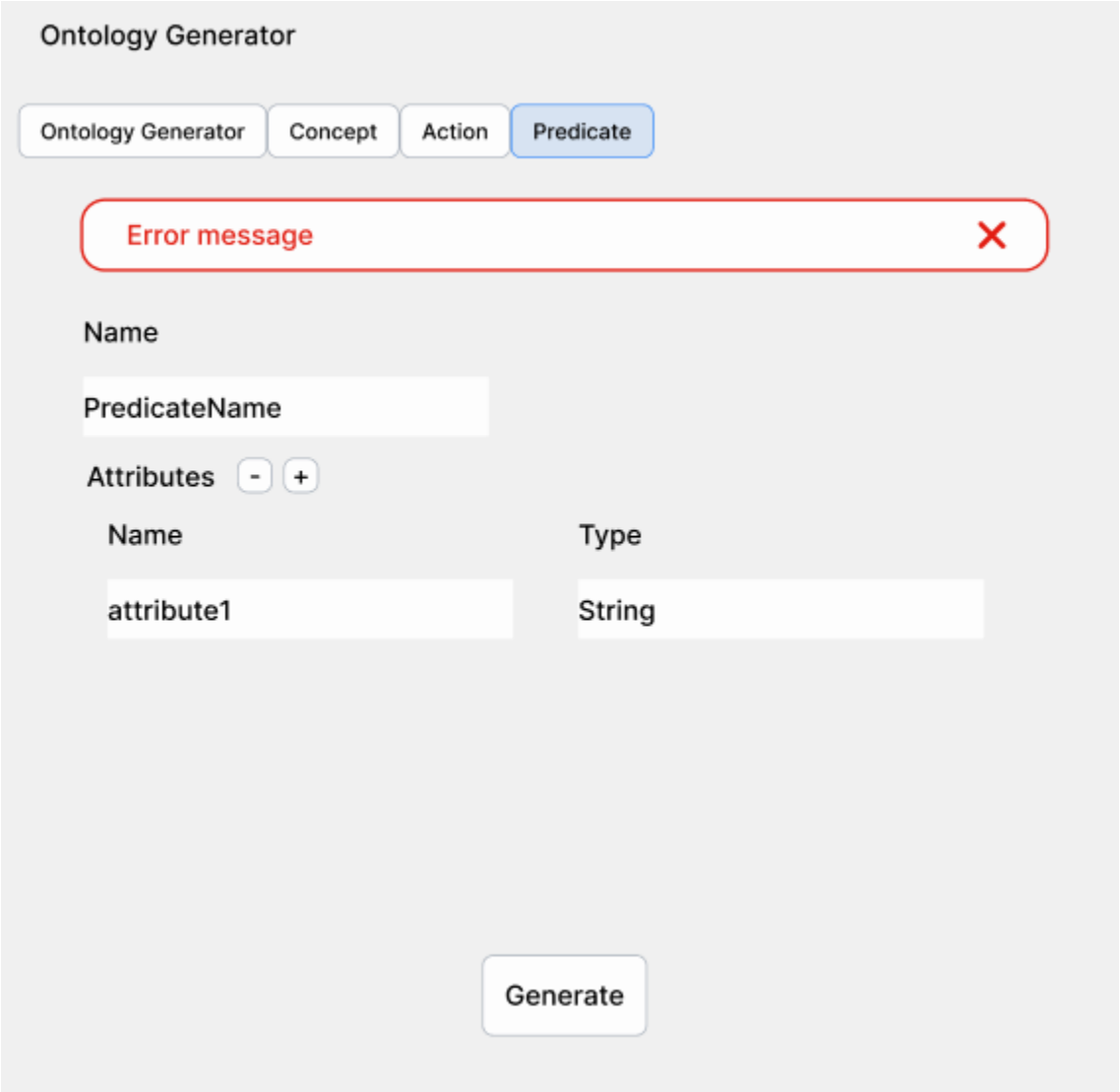


Figure 5.26: Ontology Generator interface - Error generating concept/action/-predicate second version

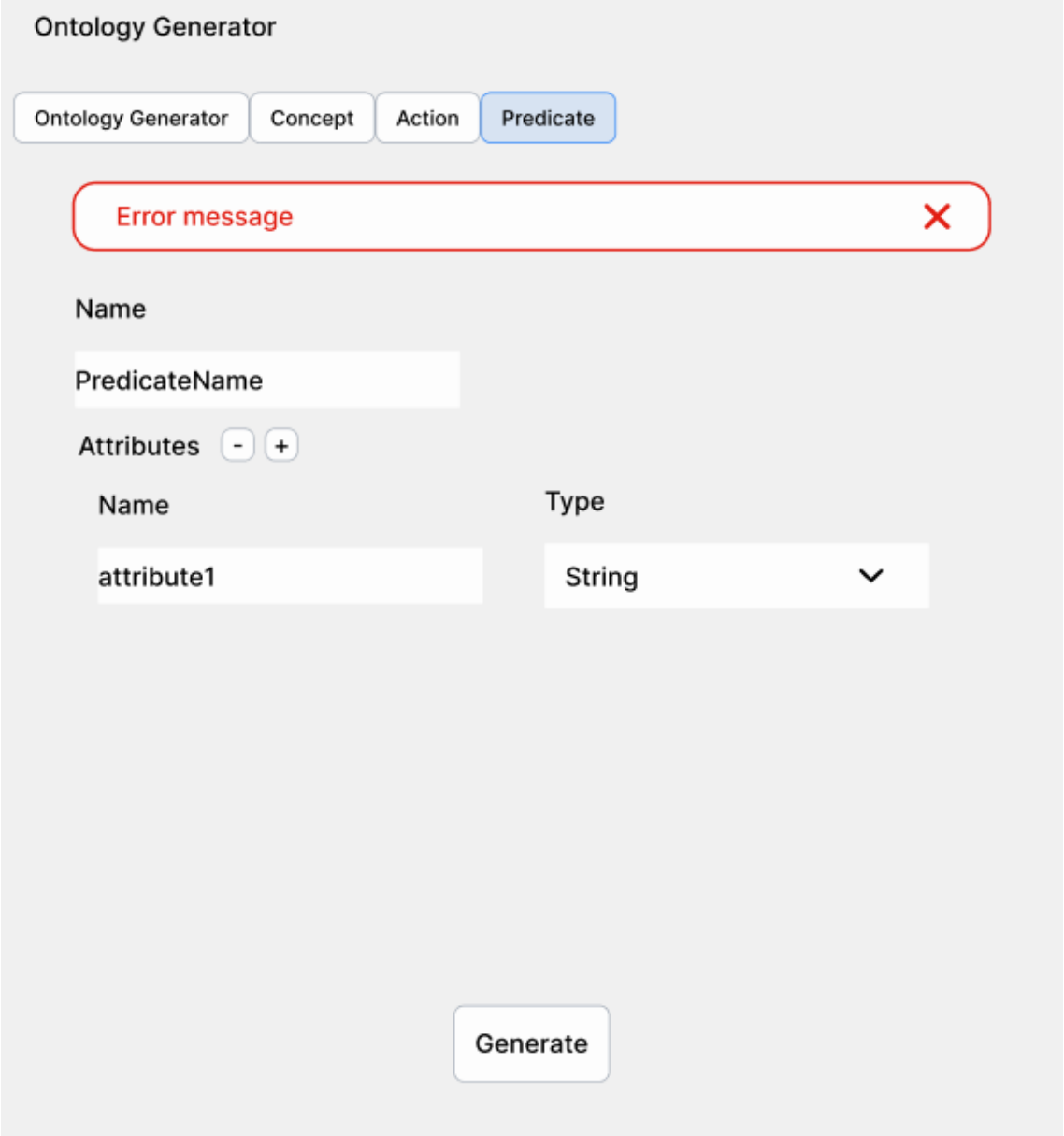


Figure 5.27: Ontology Generator interface - Error generating concept/action/-predicate third version

5.4.2. Prototype

The Ontology Generator interface can have different flows depending on the user inputs. There are four possibilities:

- **Success in the generation of the ontology:** Flow that simulates the process of generating an ontology with a specification file or entering the data in a manual way on the screens without any error.
- **Specification file not found:** Flow that simulates the process of entering a specification file that does not exist.
- **Error specification file:** Flow that simulates the process of entering a specification file that does not have the structure specified (5.3.1).

- **Error in generation of a concept, action or predicate:** Flow that simulates the process of entering invalid data when generating a new concept, action or predicate.

These four flows have been included in a prototype that can be viewed at:

`https://www.figma.com/proto/sJme3sDNZ0XsaDTyzdkGcq/Ontology-Generator?node-id=9-759&p=f&t=La5GntsNOAWS32ar-1&scaling=scale-down&content-scaling=fixed&page-id=1%3A2&starting-point-node-id=9%3A759&show-proto-sidebar=1`

They use the screens explained in the previous section (5.4.1).

5.4.3. User flow

This section describes the flow that a user must follow to use the interface. The chosen user flow conditions the implementation of the interface, so an analysis of the existing possibilities was made in order to choose the best one in terms of both implementation and usability of the Ontology Generator. There are three possible types of user flow:

1. **Generate all each time**: When the user wants to make a change in the ontology, the user has to load the entire ontology. This means that the user has to add the ontology items without changes again and then add the items with changes or the new items.

2. **Generate an intermediate specification file**: When the user wants to make a change in the ontology, the user first has to load the ontology specification file generated with the previous execution of the Ontology Generator interface (with the ontology items without changes) and, after loading the file, the user can make the changes adding new ontology items or the items with changes.

3. **Detect current ontology items**: When the user wants to make a change in the ontology, the user only has to add the items with changes or new items. In this case the Ontology Generator interface has to read and load current ontology items.

To choose the best solution, an analysis was made with the advantages and disadvantages of each option. The following table shows the results of the analysis.

Option	Advantages	Disadvantages
1. Generate all each time	▪ Less complexity ▪ Shorter development time	▪ Tedious process for the user
2. Generate an intermediate specification file	▪ The process generates a specification file that offers the possibility for the user to quickly generate the ontology.	▪ More complexity than option 1 (generate the specification file) but less complexity than option 3. ▪ More tedious for the user than option 3 but less tedious than option 1. ▪ Longer development and processing time (load file - make changes - generate file)
3. Detect current ontology items	▪ Very usable, the user does not have to worry about what is currently.	▪ More complexity ▪ Longer development and processing time

Table 5.1: Analysis of user flows

Finally, after analyzing the options and discussing the solution in the Sprint Review of Sprint 4, we have decided that the best solution for this work is option 2: **Generate an intermediate specification file**.

5.4.4. Visual Outcome

This section shows the result of implementing the interface designed in section 5.4.1 with all flows described in 5.4.2 taking into account the user flow decided (5.4.3).

For the implementation of the interface, I decided to use Java swing package [11].

The first screen that appears when running the Ontology Generator interface is the Main screen and it has two versions that look as follows:

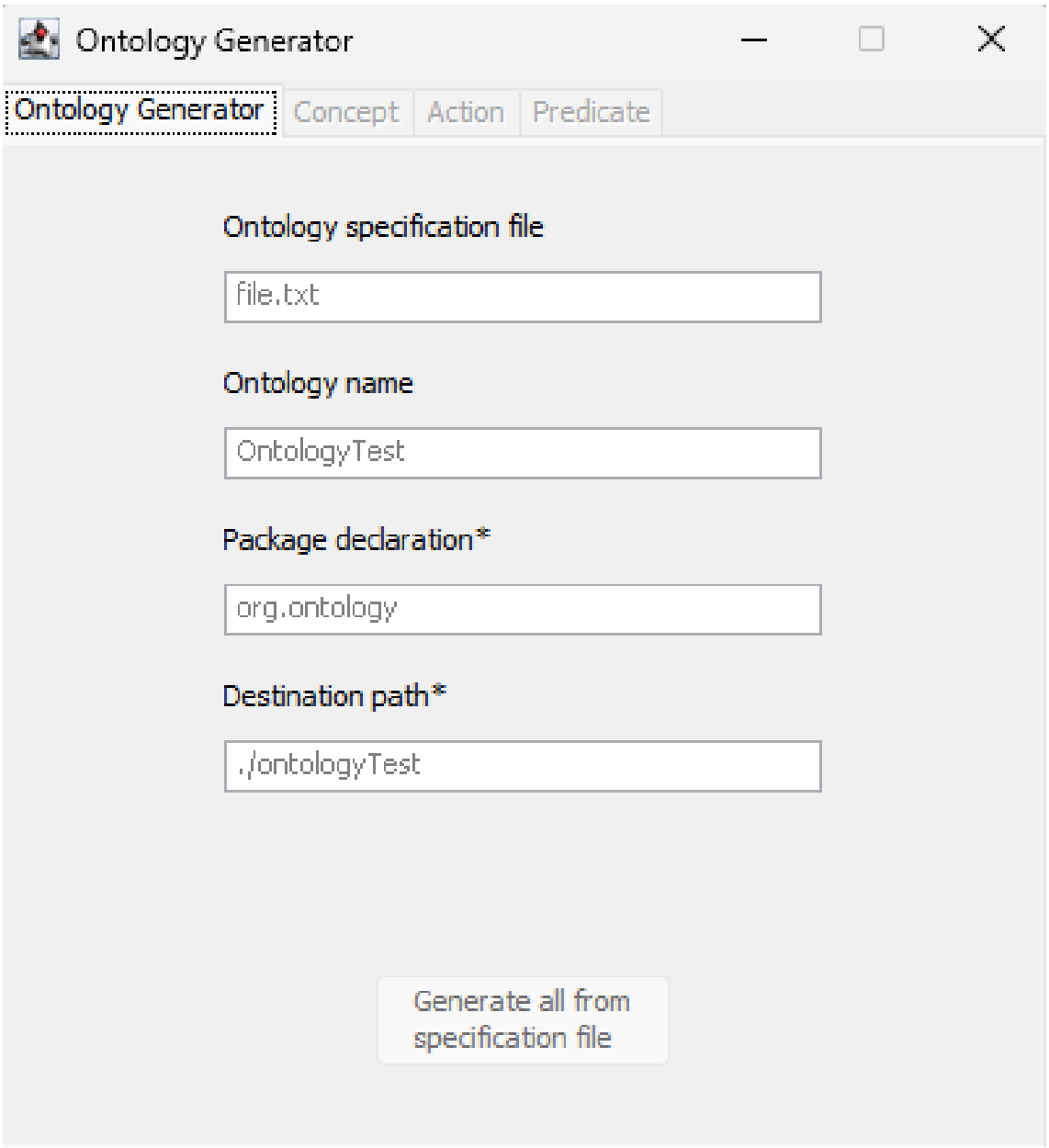


Figure 5.28: Ontology Generator interface - Main screen result

We can see that the Concept, Action and Predicate tabs and the Generate all from specification file button are disabled because we have not entered the required values on the Main screen. If we enter the ontology specification file, the package declaration, and the destination path, then the button becomes enabled. And if we also add the ontology name, the tabs become enabled.

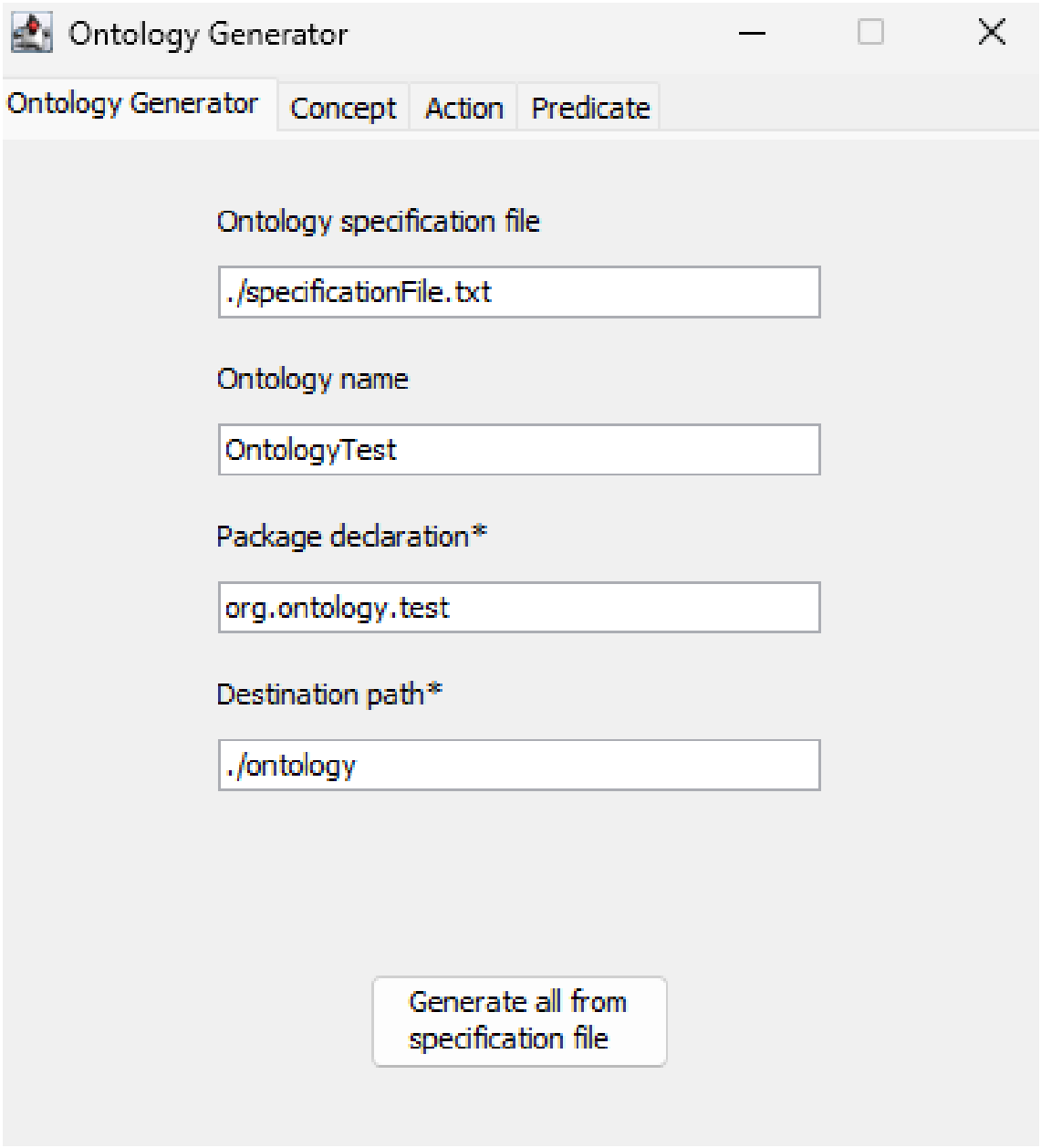


Figure 5.29: Ontology Generator interface - Main screen with data

By clicking on the "Generate all from specification file"button, we can obtain different variants of this screen depending on the result of the operation: file not found, success banner, or error banner.

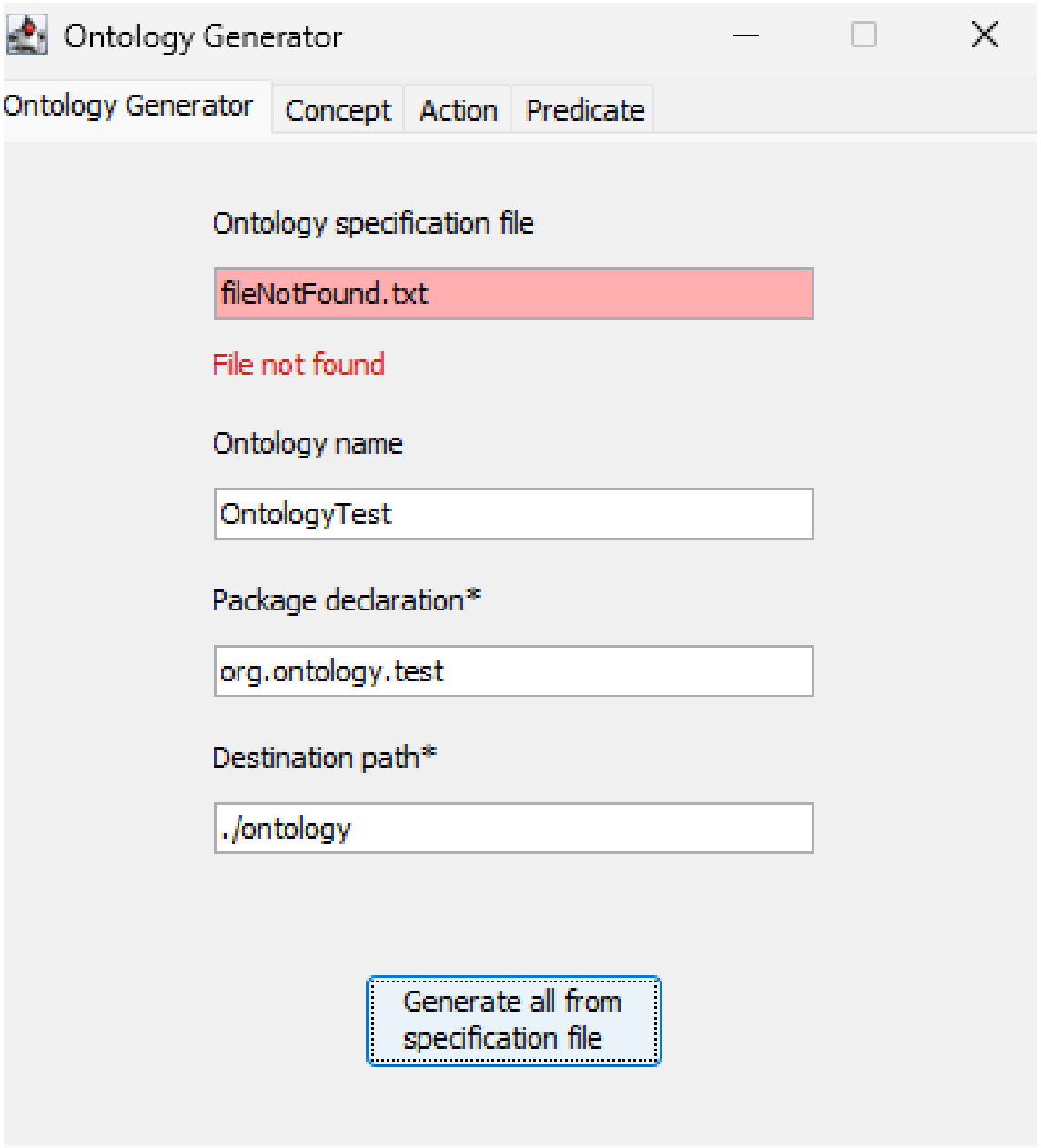


Figure 5.30: Ontology Generator interface - Main screen with file not found error

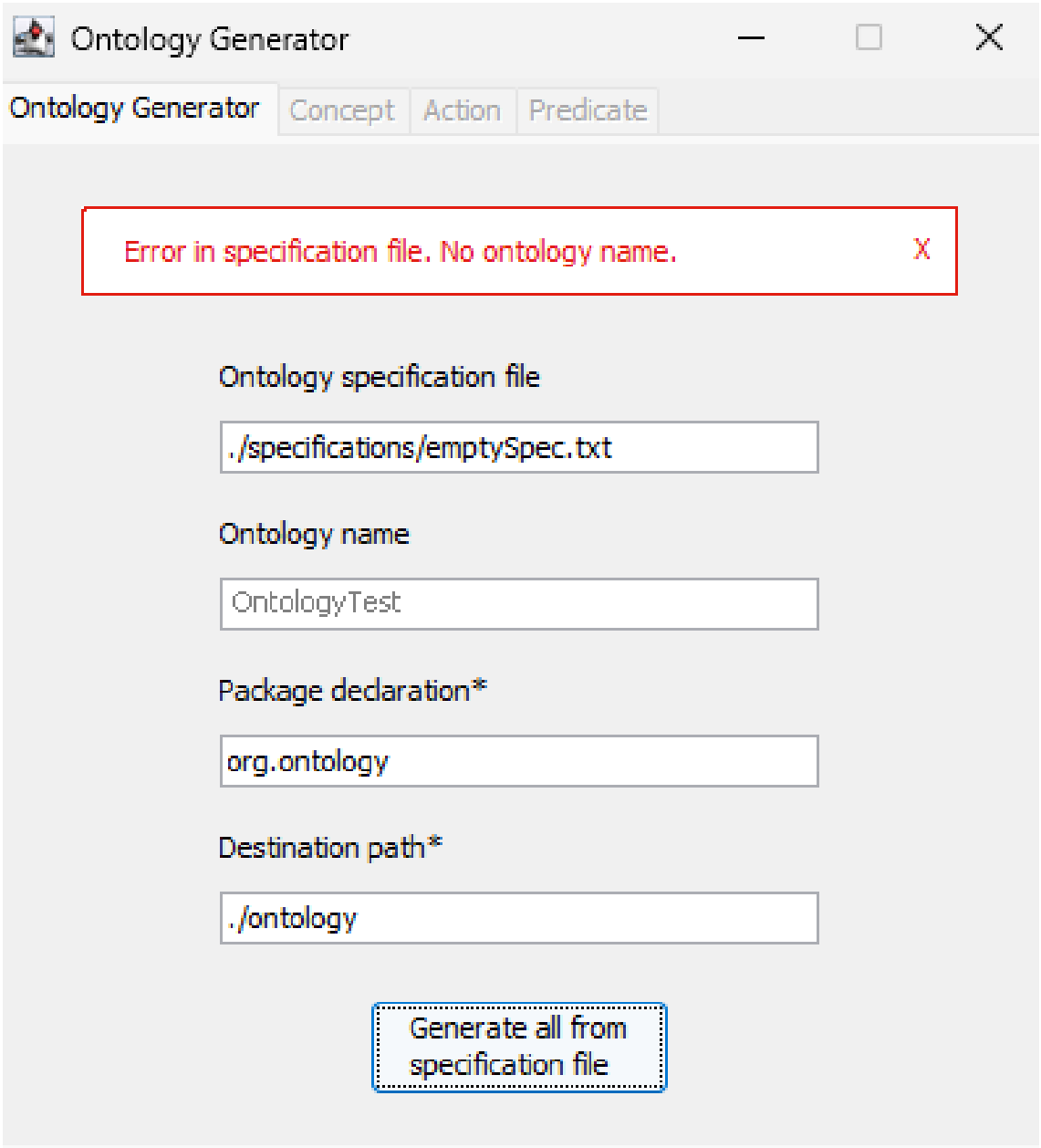


Figure 5.31: Ontology Generator interface - Main screen with error banner

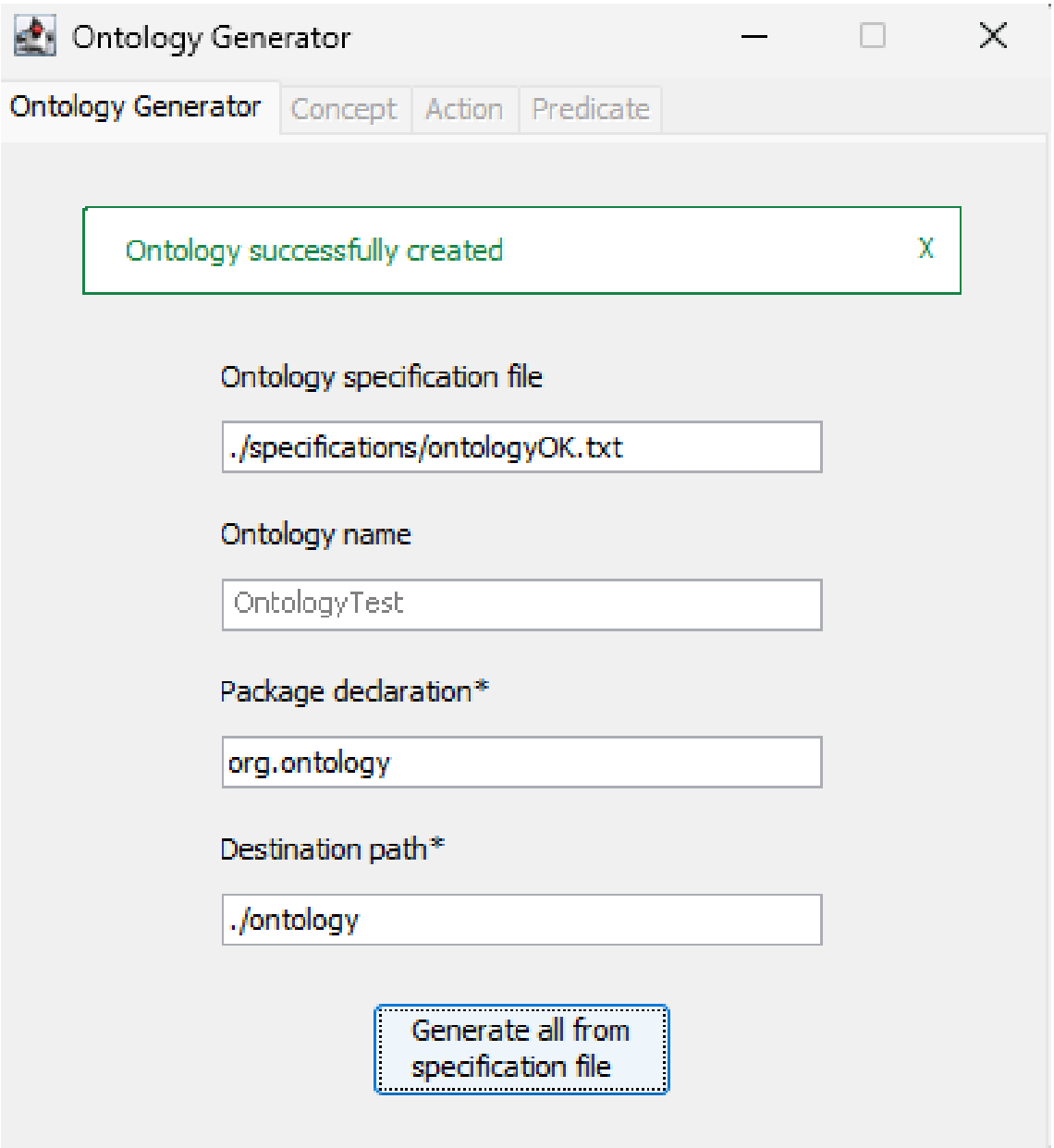


Figure 5.32: Ontology Generator interface - Main screen with success banner

On the other hand, clicking on the Concept tab, the interface shows the form to add a new concept.

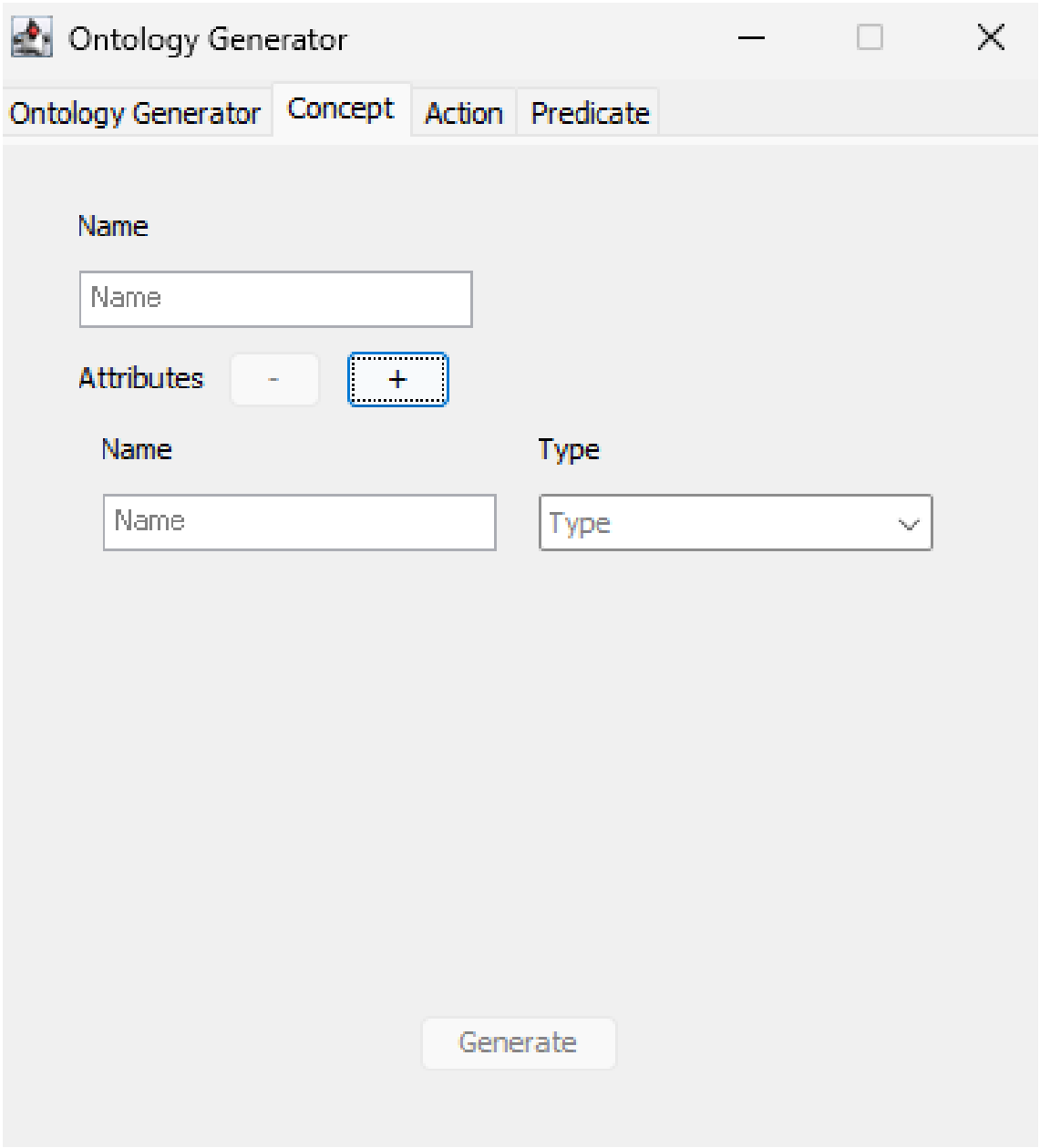


Figure 5.33: Ontology Generator interface - New Concept screen

The Generate button appears disabled because it is necessary to enter at least the name of the new concept.

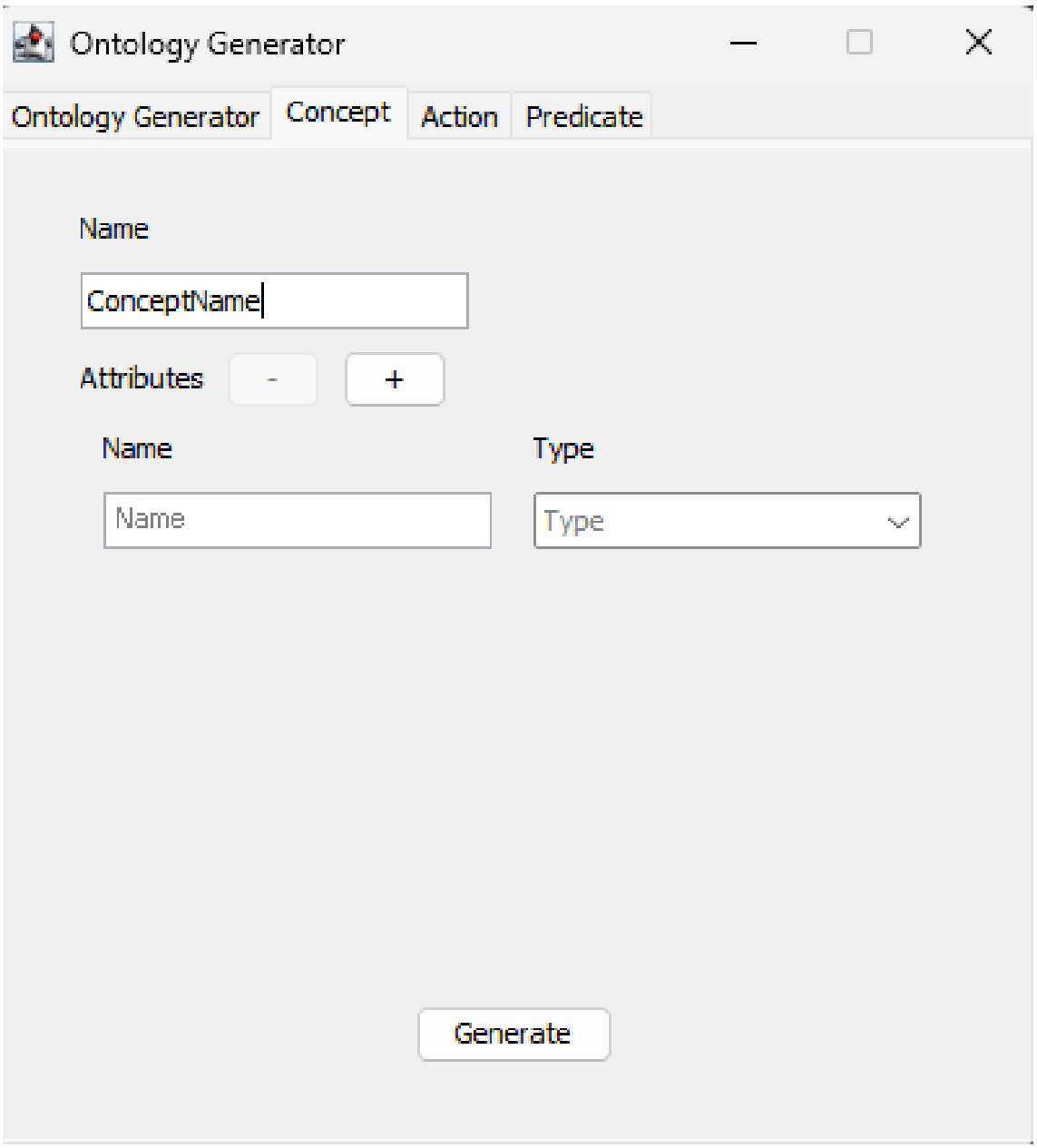


Figure 5.34: Ontology Generator interface - New Concept screen with minimal data

The attribute type inputs are selectors that provide the user with a list of possible types (primitive types and concepts already created in the ontology) in order to select a valid type.

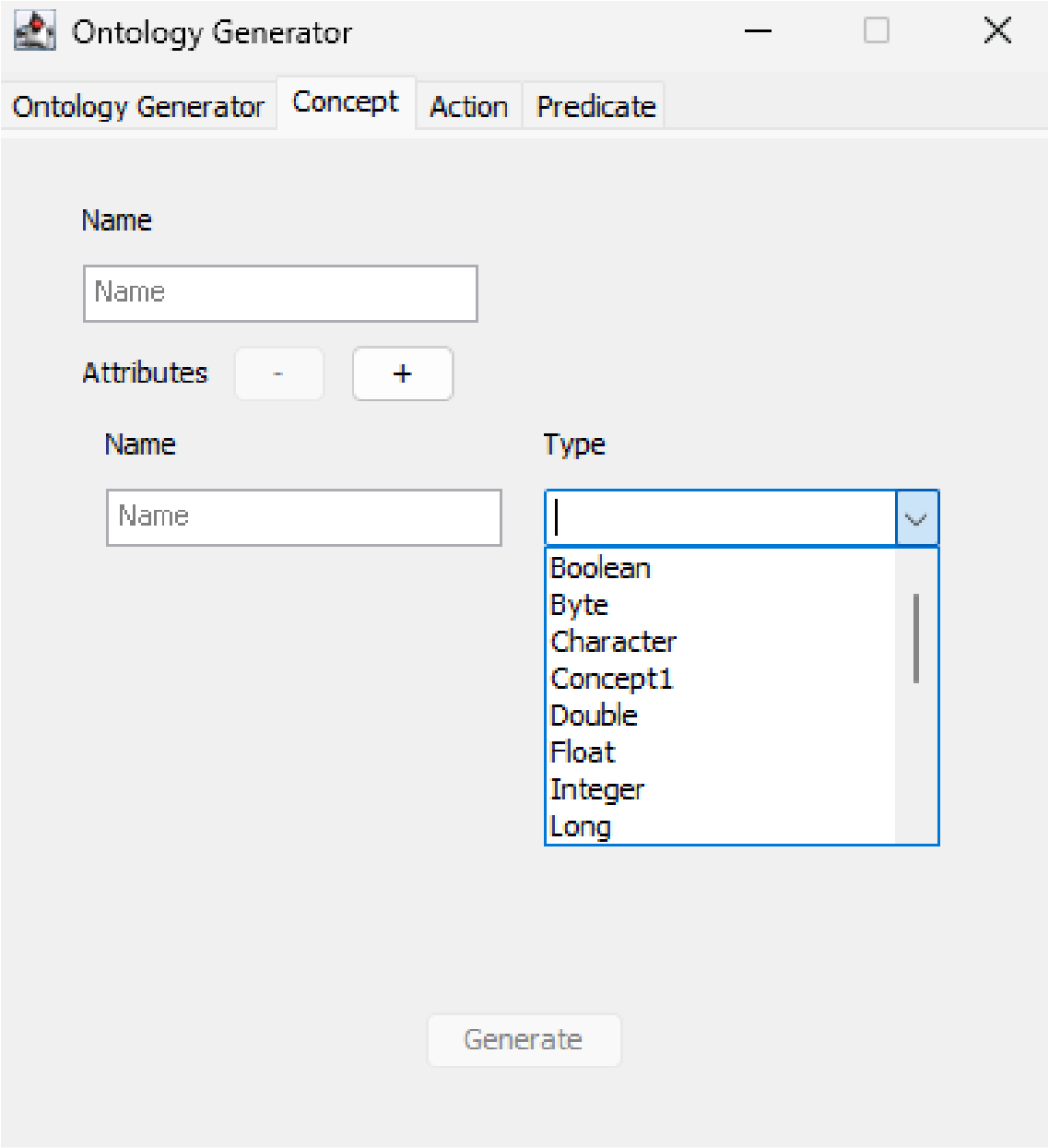


Figure 5.35: Ontology Generator interface - New Concept screen with type selector opened

The type selector has search functionality that allows the user to filter the list of types.

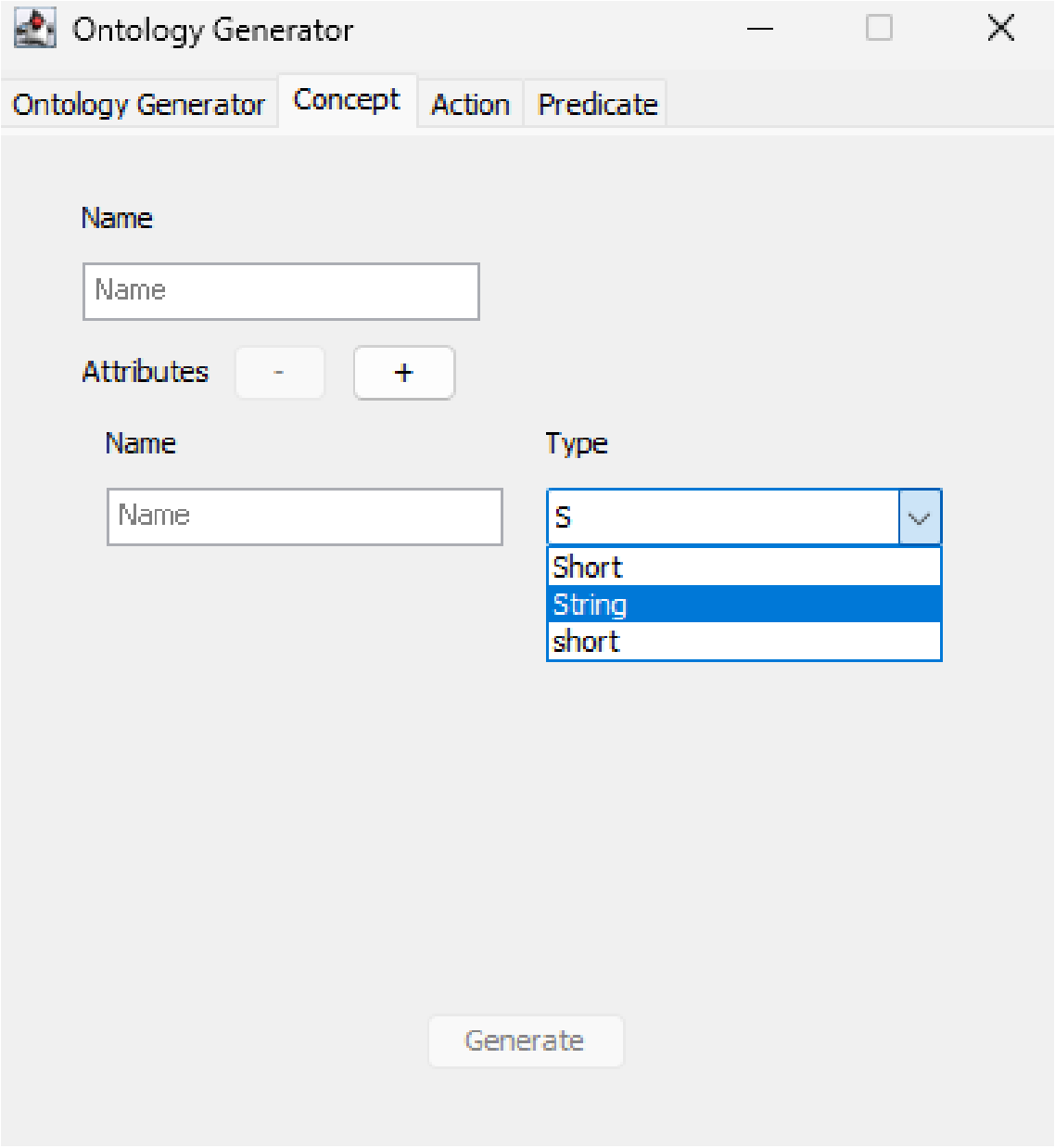


Figure 5.36: Ontology Generator interface - New Concept screen with types filtered

We can add as many attributes as we want to the concept and, from the sixth attribute onwards, a scroll bar appears on the right so that we can display all of them in the screen space of the interface.

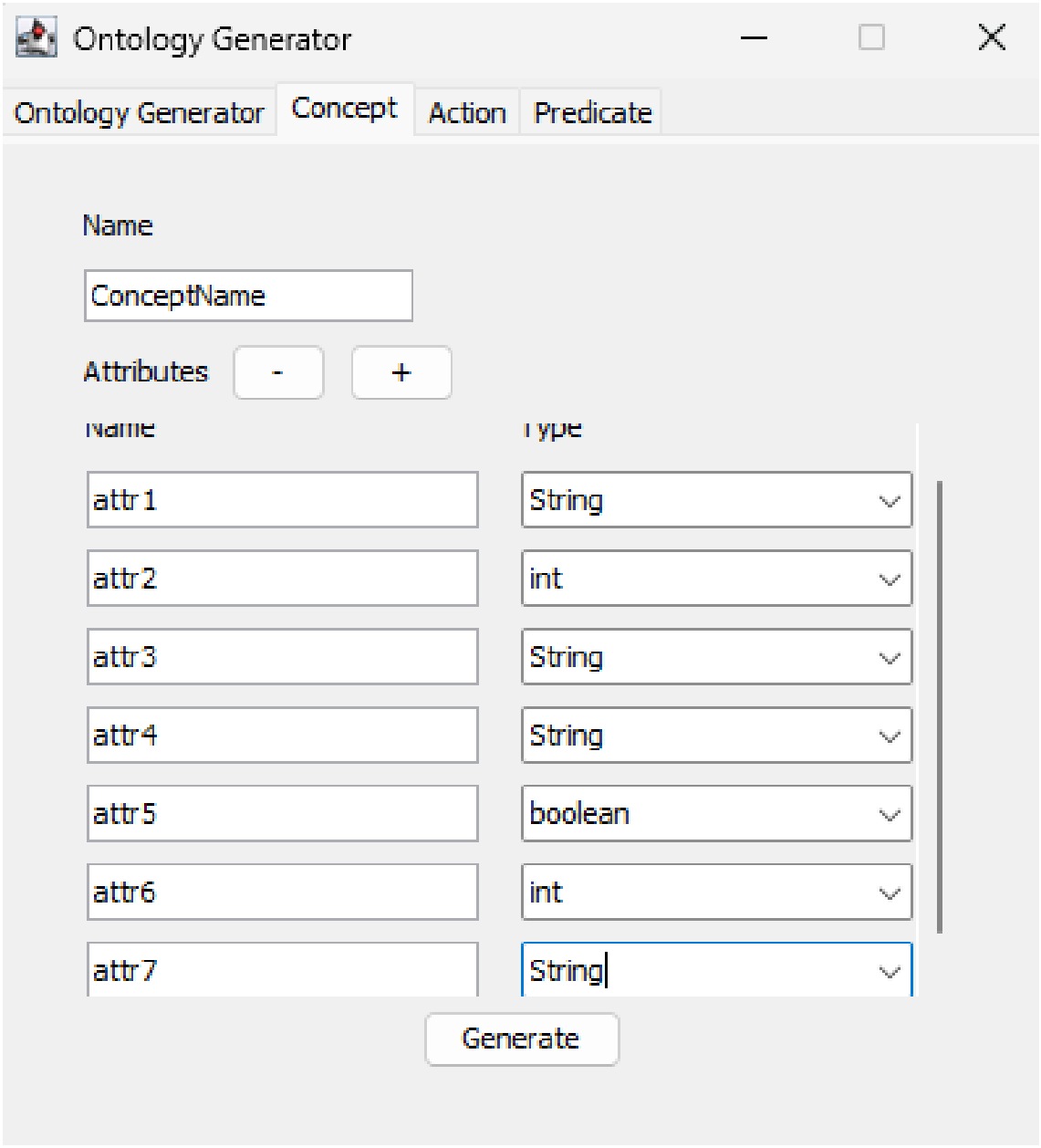


Figure 5.37: Ontology Generator interface - New Concept screen with more than five attributes first version

Also, the new concept screen has banners to indicate the results of the operations.

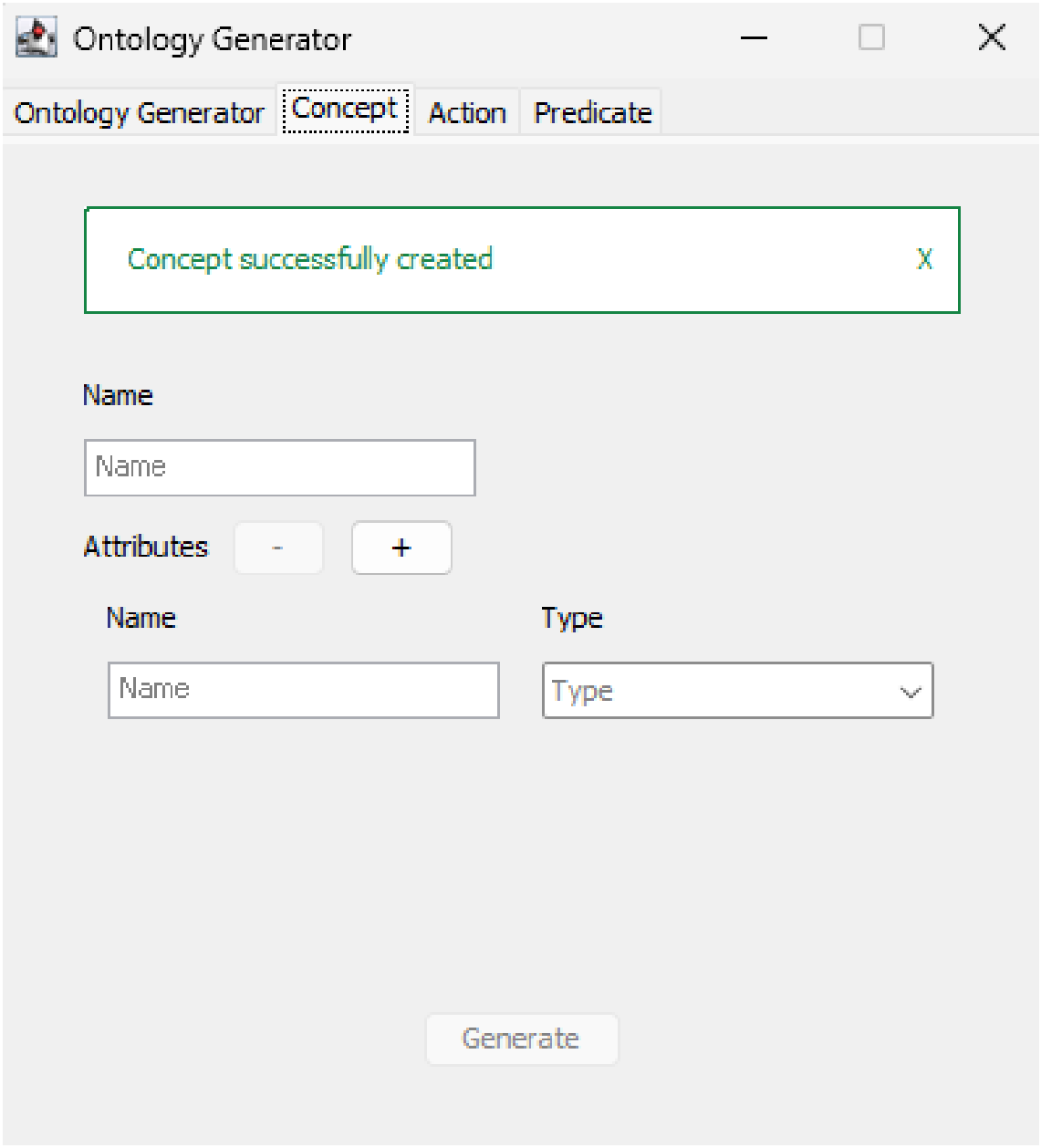


Figure 5.38: Ontology Generator interface - New Concept screen success banner

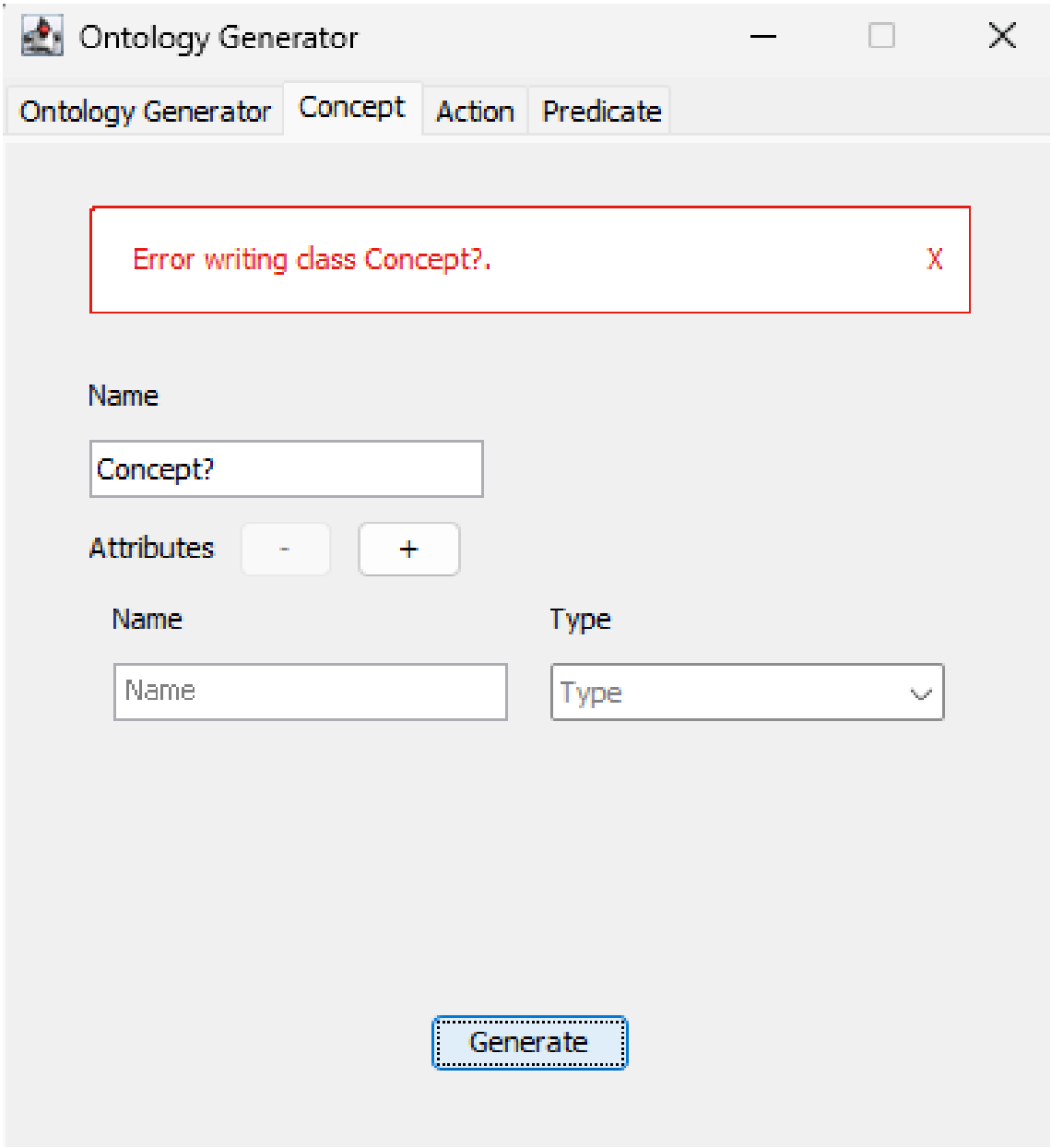


Figure 5.39: Ontology Generator interface - New Concept screen error banner

Clicking on the Action tab, the interface shows the form to add a new action.

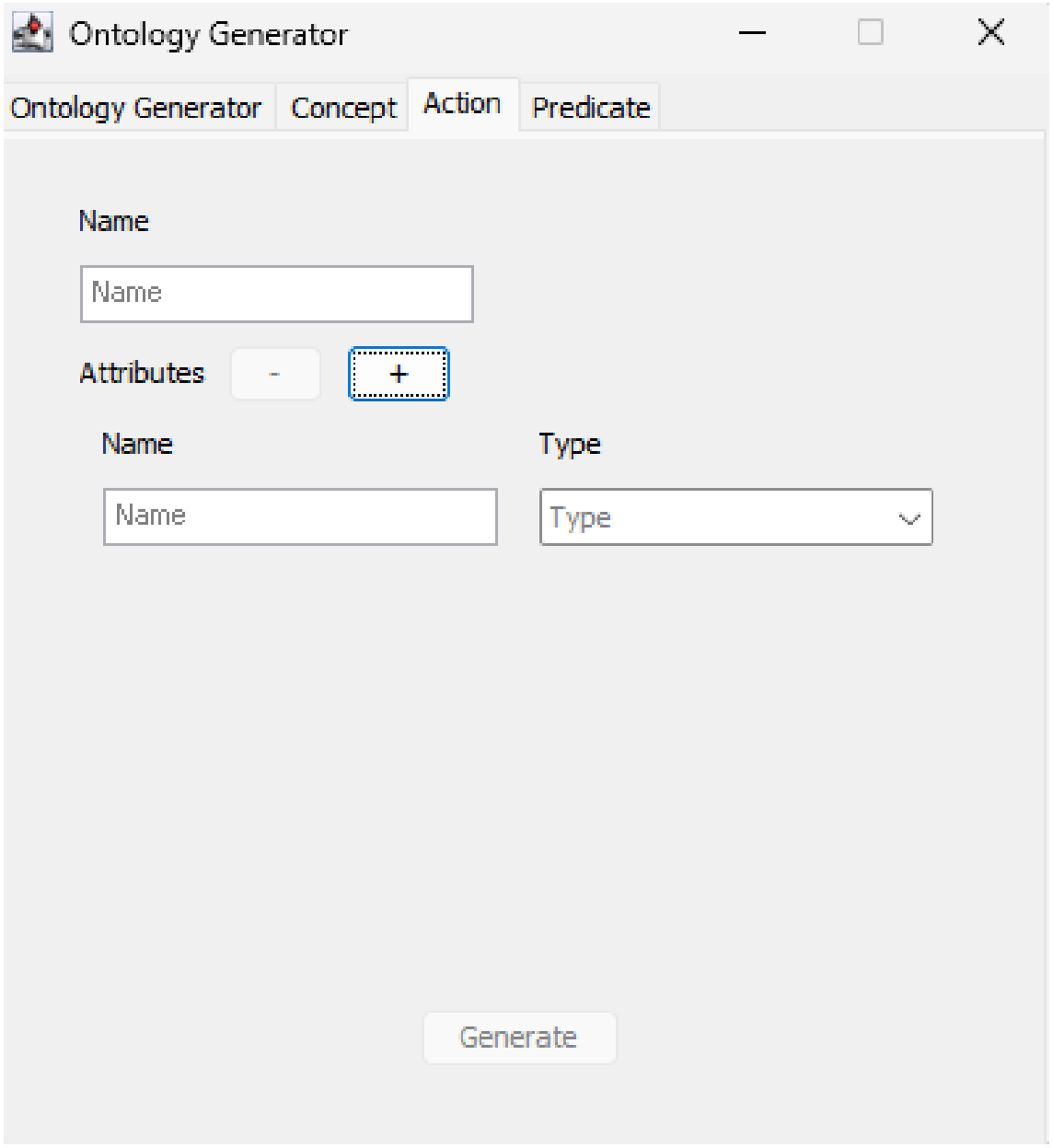


Figure 5.40: Ontology Generator interface - New Action screen

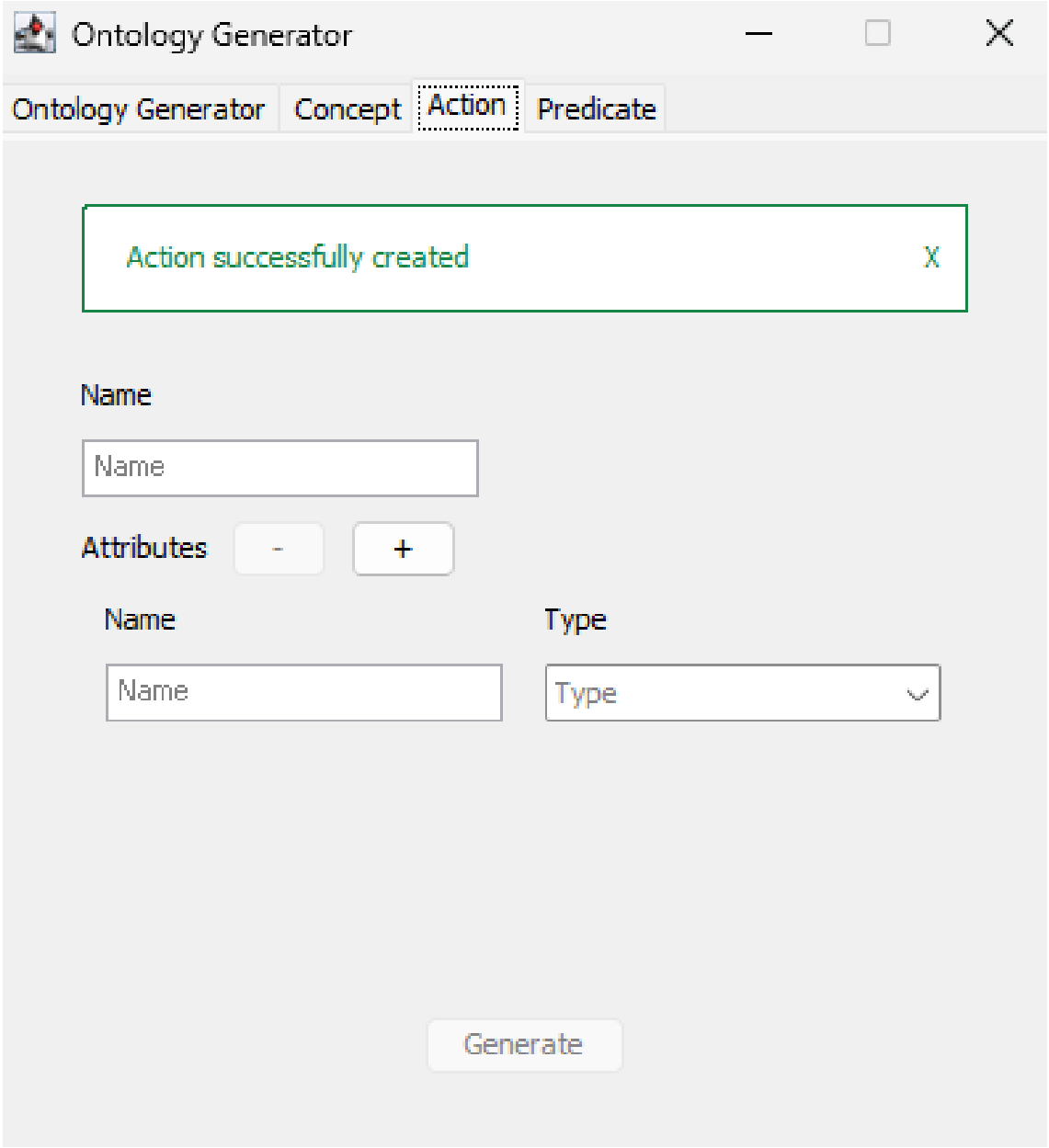


Figure 5.41: Ontology Generator interface - New Action screen success banner

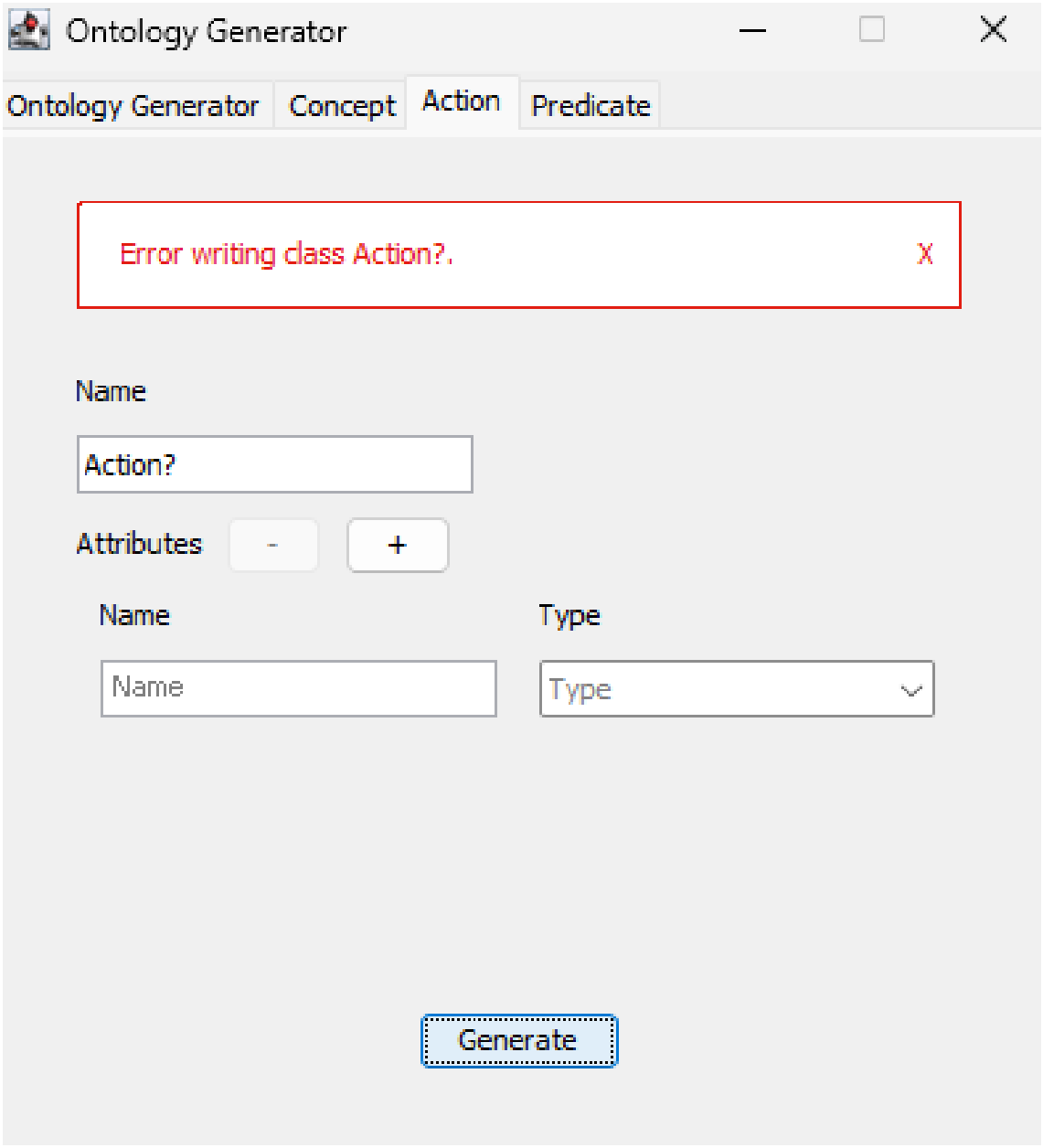


Figure 5.42: Ontology Generator interface - New Action screen error banner

Finally, clicking on the Predicate tab, the interface shows the form to add a new predicate.

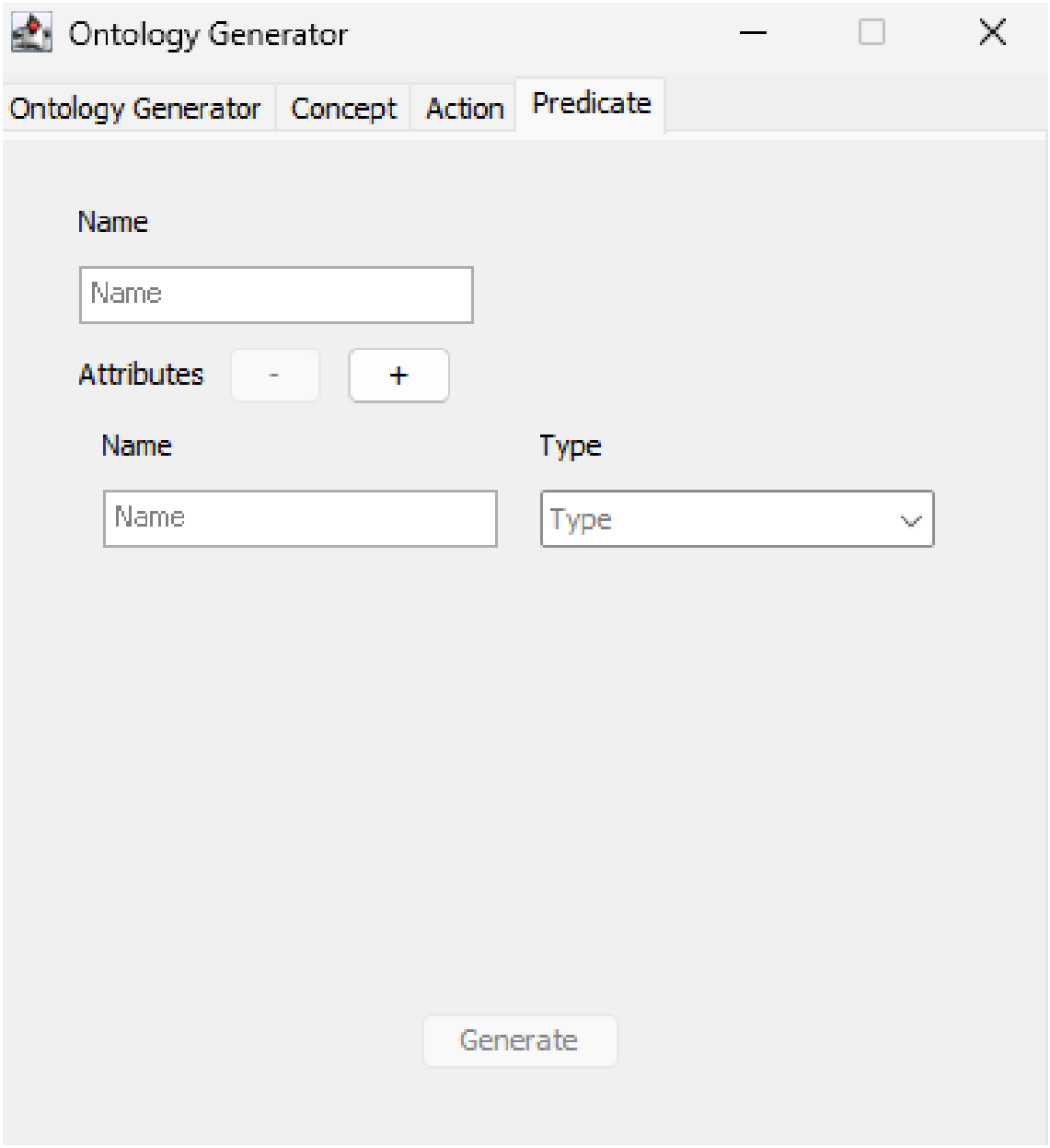


Figure 5.43: Ontology Generator interface - New Predicate screen

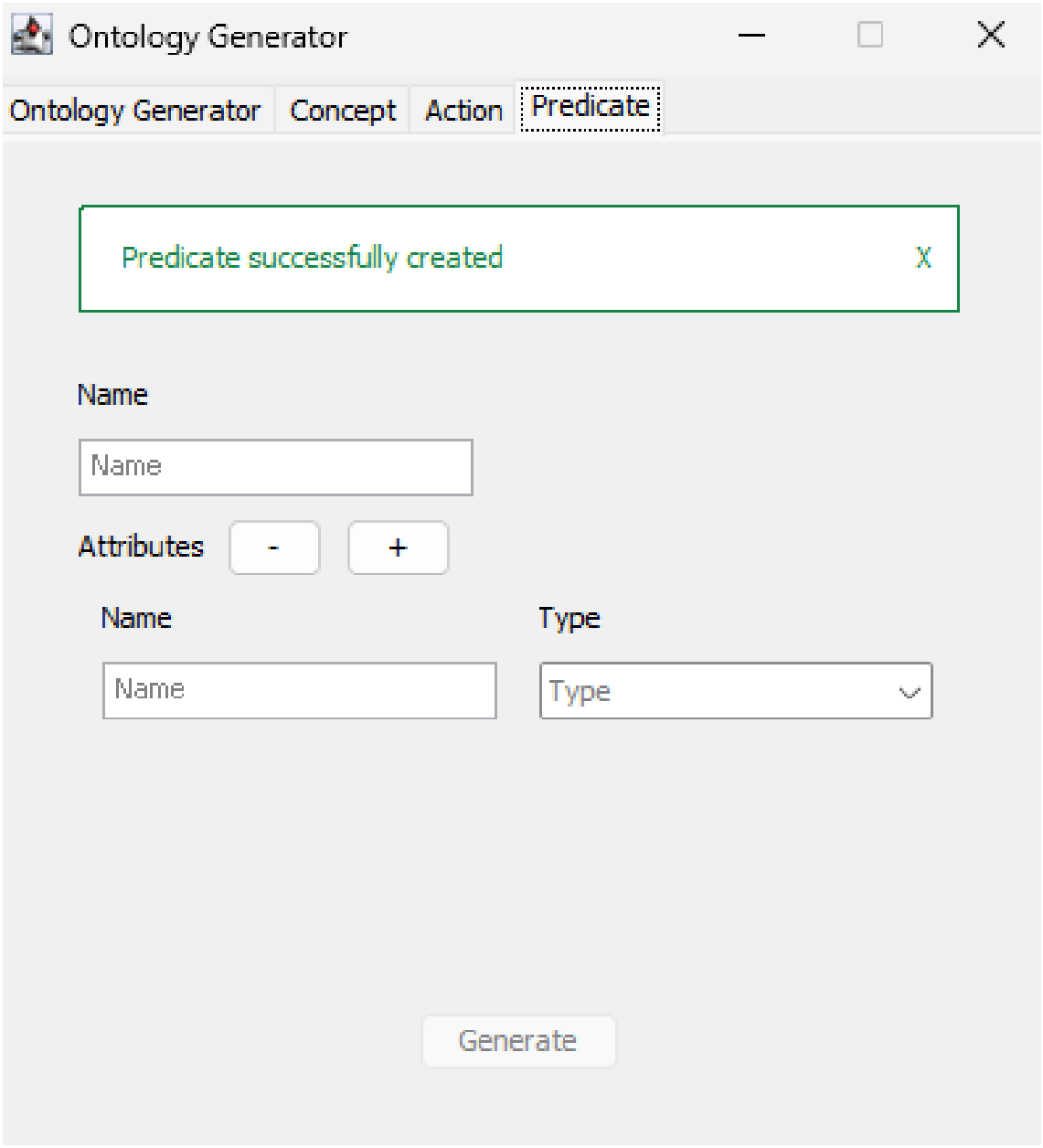


Figure 5.44: Ontology Generator interface - New Predicate screen success banner

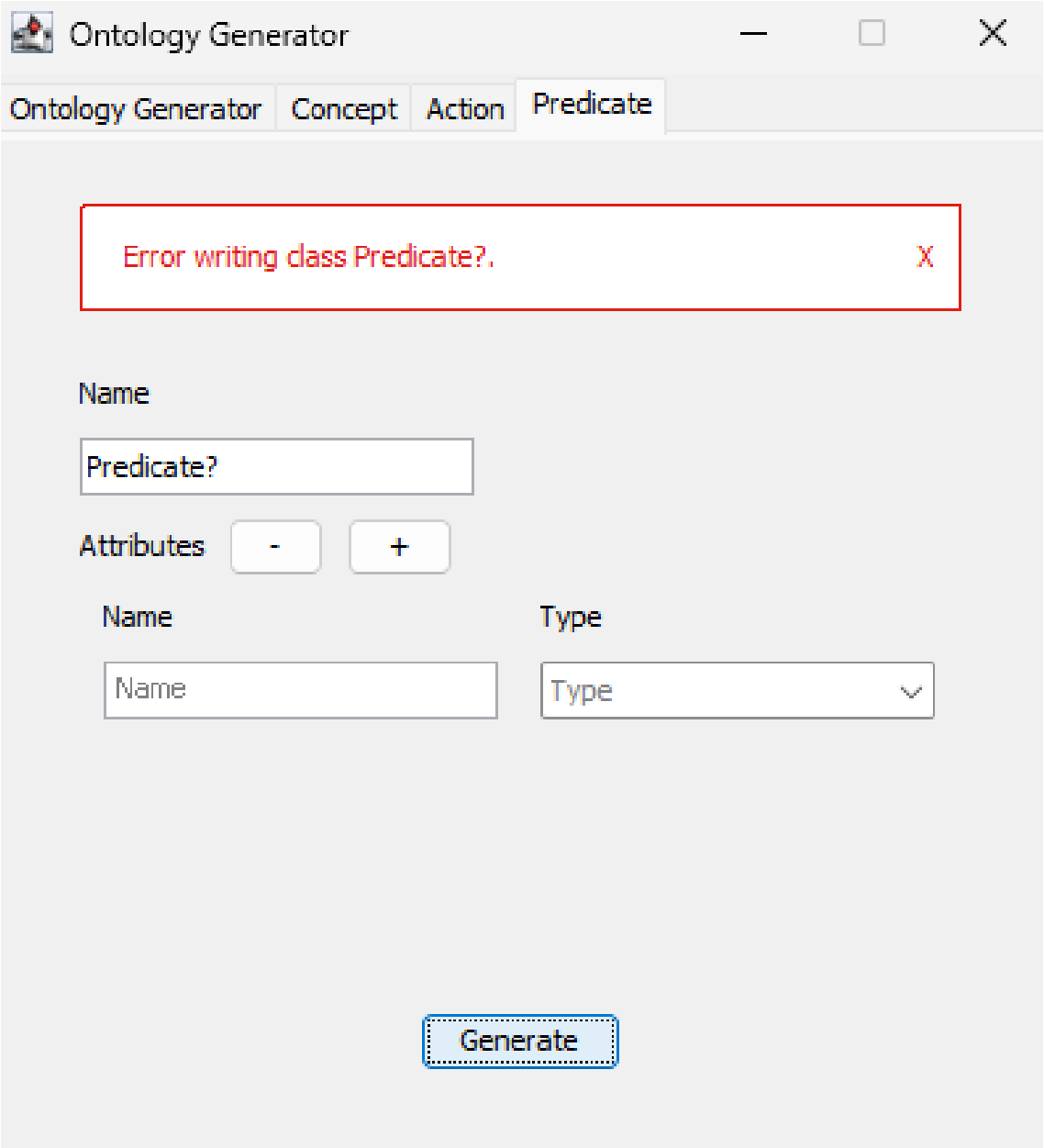


Figure 5.45: Ontology Generator interface - New Predicate screen error banner

The New Action and New Predicate screens has the same behavior with the data as the New Concept screen (Generate button, types selector, and scroll bar in attributes).

5.5. Response codes in the generation of the ontology from a specification file

The process of generating the ontology can have different response codes depending on the user input (specification file) and the file management system of the computer that executes the program.
The response codes are complemented with a descriptive message and they are divided in two groups:

- **Success response code:** The Ontology Generator returns this code to indicate that the ontology has been generated correctly.
  - **OK** (000): *"Operation completed successfully"*
- **Error response codes:** The Ontology Generator returns these codes when an error occurs during generation, they can be divided in two subgroups depending on the source of the error.
  - **Bad request:** Related with errors in the user input (ontologySpecificationFile). To avoid these errors, it is advisable to read sections 5.3.1 and 5.3.2 of this document.
    - **KO_SPECIFICATION** (003): *"Error in specification file [description of the error]"*.
    - **KO_INCORRECT_GROUP** (004): *"Incorrect group in specification file: [invalid group]"*.
    - **KO_FILE_NOT_FOUND** (005): *"File not found [path of the specification file]"*.
    - **KO_INCORRECT_CLASS_SPECIFICATION** (006): *"Incorrect specification of class: [name of the class]"*.
  - **System error:** Related to an internal service error. These errors are unlikely to occur if the Ontology Generator source code is not modified.
    - **KO_READING** (001): *"Error reading specification file [path of the specification file]"*.
    - **KO_WRITING** (002): *"Error writing class [name of the class]"*.

These responses are notified to the user in the console output using a model called ResponseBase (with two attributes: code and message). This model is used in service responses.

6 Ontology Generator - Black Box Testing

This section describes the tests performed to ensure the good functioning of the Ontology Generator. It uses a black box technique to define the test plan.

6.1. Equivalence classes definition

Input condition	Valid equivalence class	Equivalence class invalid
IV option	1. -h 2. --help 3. -d 4. --debug 5. -f 6. --file 7. -g 8. --gui 9. Empty	10. Any string that is not in the valid options
V specificationFile	11. Empty (help, debug and gui cases) 12. Name of a file (file case)	13. Any string (help, debug and gui cases) 14. Empty (file case) 15. A file that does not exist (file case)
VI content specificationFile	16. Specification of the ontology with the correct format (specified in the Ontology Generator: User Manual)	17. Empty 18. No ontology name 19. Incorrect class specification 20. Incorrect group in specification
VII packageDeclaration	21. Empty (help, debug and gui cases) 22. Package structure (file case)	23. Any string (help, debug and gui cases) 24. Empty (file case)
VIII destinationPath	25. Empty (help, debug and gui cases) 26. Valid path to save the ontology files (file case)	27. Any string (help, debug and gui cases) 28. Empty (file case) 29. Invalid path (file case)

Table 6.1: Equivalence classes definition

6.2. Equivalence classes table

Eq. class type	**Valid equivalence classes**																**Invalid equivalence classes**												
Input condition	**I**									**II**		**III**	**IV**		**V**		**I**	**II**			**III**				**IV**		**V**		
Eq. class	1	2	3	4	5	6	7	8	9	11	12	16	21	22	25	26	10	13	14	15	17	18	19	20	23	24	27	28	29
TC1	**X**	-	-	-	-	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TC2	-	-	-	**X**	-	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TC3	-	-	-	-	-	**X**	-	-	-	-	**X**	**X**	-	**X**	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-
TC4	-	**X**	-	-	-	-	-	-	-	-	-	-	**X**	-	**X**	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-
TC5	-	-	**X**	-	-	-	-	-	-	**X**	-	-	-	-	-	-	-	**X**	-	-	-	-	-	-	**X**	-	-	-	-
TC6	-	-	-	-	-	-	-	-	-	**X**	-	-	**X**	-	**X**	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-
TC7	-	-	-	-	**X**	-	-	-	-	-	-	-	**X**	-	**X**	-	-	-	**X**	-	-	-	-	-	-	-	-	-	-
TC8	-	-	-	-	-	**X**	-	-	-	-	-	-	**X**	-	**X**	-	-	-	-	**X**	-	-	-	-	-	-	-	-	-
TC9	**X**	-	-	-	-	-	-	-	-	-	-	-	**X**	-	-	-	-	**X**	-	-	-	-	-	-	**X**	-	**X**	-	-
TC10	-	-	-	-	**X**	-	-	-	-	-	**X**	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	**X**	-
TC11	-	-	-	-	-	**X**	-	-	-	-	**X**	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**X**
TC12	-	-	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**X**	-	**X**	-
TC13	-	-	-	-	-	**X**	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	**X**	-	-	-	-	-	-	-	-
TC14	-	-	-	-	-	**X**	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	-	**X**	-	-	-	-	-	-	-
TC15	-	-	-	-	**X**	-	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	-	-	-	**X**	-	-	-	-	-
TC16	-	-	-	-	-	**X**	-	-	-	-	**X**	-	-	**X**	-	**X**	-	-	-	-	-	-	-	-	**X**	-	-	-	-
TC17	-	-	-	-	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TC18	-	-	-	-	-	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TC19	-	-	-	-	-	-	-	-	**X**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.2: Equivalence classes table

6.3. Test cases definition table

6.3.1. TC1 - Test happy path help option

Context: This test checks that the program correctly displays the help text.
Inputs:

- option: -h
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: *Help text*

6.3.2. TC2 - Test happy path debug option

Context: This test checks that the program correctly generates an ontology with the debug option.
Inputs:

- option: --debug
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: The program creates an ontology structure inside the source code using as definition the ./specifications/ontologySpecification.txt file.

6.3.3. TC3 - Test happy path file option

Context: This test checks that the program correctly generates an ontology from a specification file.
Inputs:

- option: --file
- specificationFile: specificationFile.txt

```
    ExampleOntology
    concepts
        -ConceptName1
            attribute1: String
            attribute2: boolean

    actions
        -ActionName1
        -ActionName2
            concept1: ConceptName1

    predicates
        -PredicateName1
            attribute1: int
            attribute2: String
```

- packageDeclaration: org.ontology
- destinationPath: ./ontology

Expected output: OK code (000). An ontology structure is created on the destination path, with the package declaration specified and with the content indicated in the specification file.

6.3.4. TC4 - Test help option with a specification file

Context: This test checks the behavior of the program with the help option and a specification file.
Inputs:

- option: --help
- specificationFile: specificationFile.txt
- packageDeclaration: empty
- destinationPath: empty

Expected output: *Helper text*

6.3.5. TC5 - Test debug option with a specification file and package declaration

Context: This test checks the behavior of the program with the debig option, a specification file and a package declaration.
Inputs:

- option: -debug
- specificationFile: specificationFile.txt
- packageDeclaration: org.ontology
- destinationPath: empty

Expected output: *Helper text*

6.3.6. TC6 - Test invalid option

Context: This test checks the behavior of the program with an invalid option.
Inputs:

- option: -abc
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: *Helper text*

6.3.7. TC7 - Test file option without a specification file

Context: This test checks the behavior of the program with the file option and no specification file.
Inputs:

- option: -f
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: *Helper text*

6.3.8. TC8 - Test file option with a specification file that does not exist

Context: This test checks the behavior of the program with the file option and a specification file that does not exist.
Inputs:

- option: --file
- specificationFile: file_no_exist.txt
- packageDeclaration: empty
- destinationPath: empty

Expected output: KO_FILE_NOT_FOUND error (005)

6.3.9. TC9 - Test help option with a specification file, a package declaration, and a destination path

Context: This test checks the behavior of the program with the help option, a specification file, a package declaration, and a destination path.
Inputs:

- option: -help
- specificationFile: specificationFile.txt
- packageDeclaration: org.ontology
- destinationPath: ./ontology

Expected output: *Helper text*

6.3.10. TC10 - Test file option without destination path

Context: This test checks the behavior of the program with the file option, a specification file, a package declaration, and no destination path.
Inputs:

- option: -f
- specificationFile: specificationFile.txt
- packageDeclaration: org.ontology
- destinationPath: empty

Expected output: *Helper text*

6.3.11. TC11 - Test file option with invalid destination path

Context: This test checks the behavior of the program with the file option, a specification file, a package declaration, and an invalid destination path.
Inputs:

- option: -f
- specificationFile: specificationFile.txt
- packageDeclaration: org.ontology
- destinationPath: ./ontology??

Expected output: KO_WRITING error (002)

6.3.12. TC12 - Test file option without package declaration

Context: This test checks the behavior of the program with the file option, a specification file, and no package declaration.
Inputs:

- option: -f
- specificationFile: specificationFile.txt
- packageDeclaration: empty
- destinationPath: empty

Expected output: *Helper text*

6.3.13. TC13 - Test file option with empty specification file

Context: This test checks the behavior of the program with the file option, a package declaration and a destination path and an empty specification file.
Inputs:

- option: --file
- specificationFile: specificationFile.txt (empty)
- packageDeclaration: org.ontology
- destinationPath: ./ontology

Expected output: KO_SPECIFICATION error (003).

6.3.14. TC14 - Test file option without ontology name in the specification file

Context: This test checks the behavior of the program with the file option, a package declaration and a destination path, and a specification file without ontology name.
Inputs:

- option: --file
- specificationFile: specificationFile.txt

```
concepts
    -ConceptName1
        attribute1: String
        attribute2: boolean

actions
    -ActionName1
    -ActionName2
        concept1: ConceptName1

predicates
    -PredicateName1
        attribute1: int
        attribute2: String
```

- packageDeclaration: org.ontology
- destinationPath: ./ontology

Expected output: KO_SPECIFICATION error (003).

6.3.15. TC15 - Test file option with invalid class specification in the specification file

Context: This test checks the behavior of the program with the file option, a package declaration and a destination path, and a specification file with an invalid class specification.
Inputs:

- option: -f
- specificationFile: specificationFile.txt

```
ExampleOntology
concepts
    -ConceptName1
        attribute1: String
        attribute2: boolean

actions
    -ActionName1
    -ActionName2
        concept1: ConceptName1

predicates
    -PredicateName1
        attribute1
        attribute2: String
```

- packageDeclaration: org.ontology

- destinationPath: ./ontology

Expected output: KO_INCORRECT_CLASS_SPECIFICATION error (006).

6.3.16. TC16 - Test file option with incorrect group in the specification file

Context: This test checks the behavior of the program with the file option, a package declaration and a destination path, and a specification file with an incorrect group.
Inputs:

- option: --file
- specificationFile: specificationFile.txt

```
ExampleOntology
concepts
    -ConceptName1
        attribute1: String
        attribute2: boolean

group
    -ActionName1
    -ActionName2
        concept1: ConceptName1

predicates
    -PredicateName1
        attribute1: int
        attribute2: String
```

- packageDeclaration: org.ontology
- destinationPath: ./ontology

Expected output: KO_INCORRECT_GROUP error (004).

6.3.17. TC17 - Test happy path -g option

Context: This test checks that the program correctly displays the interface with the -g option.
Inputs:

- option: -g
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: The program displays the interface to generate an ontology through user inputs.

6.3.18. TC18 - Test happy path --gui option

Context: This test checks that the program correctly displays the interface with the --gui option.

Inputs:

- option: --gui
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: The program displays the interface to generate an ontology through user inputs.

6.3.19. TC19 - Test happy path no option

Context: This test checks that the program correctly displays the interface with no option provided.
Inputs:

- option: empty
- specificationFile: empty
- packageDeclaration: empty
- destinationPath: empty

Expected output: The program displays the interface to generate an ontology through user inputs.

7 Team - Platform Environment

This master thesis is focused on reducing the development effort of students in the Agent based Software Development course. The students in that course have to develop a game using agents in Java. The goal is to get students to focus on defining strategies and using agents without spending too much time on specific development issues.
To achieve this goal, this master thesis defines the Team - Platform Environment to provide the structure of the backend part of the game and the necessary functionality for sending messages between agents, performing the connection with the frontend part using an API, and printing debug messages with a standard format.

This section describes in depth the Team - Platform environment product that also counts with a developer manual appended to this document (12.3).

7.1. Structure of the source code

In this section we are going to explain the structure of the Team - Platform environment.

Similarly to the Ontology Generator, a private repository was created for the project code in GitHub [8] and the source code has also been implemented using Java [9].

The structure of the project provided to the students is the following.

- lib: Folder to add external libraries that cannot be downloaded via Maven.
  - jade-4.6.0.jar: JADE library [3].
- src.main: Main folder of the project.
  - java.org.game: Source code of the project.
    - agents: Folder with the implementation of the project agents.
      - platform: Folder to store platform agents.
        - AgPlatform: Example of a platform agent.
      - teamN: Folder with the agents of the team.
        - AgTeam: Example of a team agent.
      - BaseAgent: Agent with the common logic for all agents. It is described in detail in chapter 7.3 of this document.
      - MessagesManager: Interface that defines the methods to send and reply messages between agents.
    - clients: Folder with the different clients for external communications.

- GameHttpClient: Client to establish the connection with the frontend part of the game. It is described in detail in chapter 7.4 of this document.
- InterfaceCommunication: Interface that defines the methods to communicate with the frontend part.

- config: Folder to store configuration files.
  - AgentLogLayout: Custom log configuration. It is described in detail in chapter 7.5 of this document.
- dto: Folder with request classes to communicate with the frontend part.
  - AttackRequest: Definition of the request body required by the API to throw an attack to another unit.
  - ConfigRequest: Definition of the request body required by the API to configure the map.
  - EndGameRequest: Definition of the request body required by the API to end the game and to show final results.
  - KillUnitRequest: Definition of the request body required by the API to kill a unit.
  - MoveRequest: Definition of the request body required by the API move a unit.
  - TeamResult: Definition of the result of a team required in EndGameRequest.
  - UnitRequest: Definition of the request body required by the API to create a new unit.
  - UpdateHealthRequest: Definition of the request body required by the API to update the health of a unit.
- models: Optional folder to store the model used in the application.
- ontology: Folder with the ontology for communication between agents. Initially, it contains a mock ontology that students must replace for the real one.
- utils: Optional folder to store auxiliary classes.
  - LogColors: Class with the definitions of the log colors (platform logs, client logs and agents logs).
  - Utils: Class with some constants and methods that can be useful for students in the development of the project.
- AgentsApplication: File that has the main method of the application. This is the file that must be executed to create the agents and start the game. Chapter 7.2 contains a complete description of this class.

- resources: Folder with project resources.
  - log4j.properties: Properties of log4j logging framework.

- teams: Folder to store the JAR files of other teams.
- pom.xml: File with the project dependencies downloaded via Maven.

The **BaseAgent** and **GameHttpClient** classes are already implemented and students do not need to modify them. The **AgentsApplication** class is almost implemented and only requires students to perform some changes to specify the ontology and to add agents from other teams.

7.2. AgentsApplication

This section describes the methods (public and private methods) of the AgentsApplication class. These methods are provided to students to build the agents environment.

- ***main(programArguments)***: Main method of the program. It receives the arguments and builds the agents environment.
- ***loadBoot()***: Method that loads the JADE GUI and creates a default profile for the agents.
- ***loadMyPlatformAgents()***: Method that creates and starts the AgPlatform agent inside a container.
- ***loadMyTeamAgents()***: Method that creates and starts the AgTeam agent inside a container.
- ***loadTeamXAgents()***: Method that creates and starts the AgTeam agent of team X inside a container using the corresponding jar in the teams folder.
- ***printHelp()***: Method to print the help text of the program (options -h and --help).

All these methods are static and void.

7.3. BaseAgent

7.3.1. What is the BaseAgent?

The BaseAgent is an abstract class that implements the common functionality of all agents in the game. It establishes the **ontology** for the agent, initializes the **agent logger** with the Log standardization functionality (7.5), and allows the developers to add the **behaviors** of the agents through an abstract method.

Furthermore, it implements the **MessagesManager** interface that contains methods to send or reply messages and defines methods for managing the content of messages. The content of a message can be in two different locations inside a message:

- In the **content** field of the message. This option requires the content to be an action of the ontology (the concepts are also actions).
- In the **content object** field of the message.

The BaseAgent uses the first option in the implementation of the specific methods of the interface to send or reply a message, but it provides the necessary methods to create a message, set and get the content object of a given message.

7.3.2. How is it implemented?

This section explains the methods and properties that the BaseAgent abstract class provides to his children and the abstract methods that the children must implement.

7.3.2.1. Properties

The BaseAgent abstract class provides to his children three protected properties.

- **Logger logger**: The agent logger that it is recommended to use it to facilitate the debugging of the code. The logger works as explained in section 7.5.
- **Codec codec**: The codec of the ontology.
- **Ontology ontology**: The ontology that the agent uses for the communications.

7.3.2.2. Agent specific methods

The BaseAgent abstract class overrides the setup method of the Agent class to set some properties.

- ***setup()***: This method establishes the language and the ontology for communications between agents. The **ontology** must be specified in the **first position of the array of arguments** passed in the creation of the agent, otherwise, this method will delete the agent. In addition, this method initializes the logger with the agent name.

7.3.2.3. Abstract methods

The BaseAgent abstract class has one abstract method that his children must implement.

- ***configure()***: This method allows the children to specify configurations of the agent, for example, adding behaviors. It is executed inside the agent setup method.

7.3.2.4. MessagesManager methods

The BaseAgent abstract class implements the MessagesManager interface and, as a result of this, includes methods to send or reply messages with each type of performative and to fill or extract the content of a message.

The interface defines two types of methods to create and send messages between agents.

- The methods with the pattern ***send[Performative]*** receive a protocol, a list of receivers, and a content (null if not needed). These methods create an ACLMessage with the given parameters and send it.
- The methods with the pattern ***replyWith[Performative]*** receive a message and a content (null if not needed). These methods create a reply to the given message, add the content if specified, and send it.

The descriptions of **all methods** from the MessagesManager interface are as follows:

- ***newMsg(String protocol, List<AID> receivers, int perf)*** → ***ACLMessage***: Create an ACLMessage with the protocol and performative indicated, and set the receivers to the given list. Returns the message ready to be sent.
- ***extractConcept(ACLMessage msg, Class<T>type)*** → ***T***: Extract and return the concept of class T within the given message.
- ***getContentObject(ACLMessage msg, Class<T>type)*** → ***T***: Get and return the content object of class T within the given message.
- ***setContentObject(ACLMessage msg, T object)***: Set as content object of the given message the indicated object.
- ***sendAcceptProposal(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to ACCEPT_PROPOSAL.
- ***replyWithAcceptProposal(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to ACCEPT_PROPOSAL.
- ***sendAgree(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to AGREE.
- ***replyWithAgree(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to AGREE.
- ***sendCancel(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to CANCEL.
- ***replyWithCancel(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to CANCEL.
- ***sendCFP(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to CFP.
- ***replyWithCFP(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to CFP.
- ***sendFailure(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to FAILURE.
- ***replyWithFailure(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to FAILURE.
- ***sendInform(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to INFORM.
- ***replyWithInform(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to INFORM.

- ***sendInformIf(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to INFORM_IF.
- ***replyWithInformIf(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to INFORM_IF.
- ***sendNotUnderstood(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to NOT_UNDERSTOOD.
- ***replyWithNotUnderstood(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to NOT_UNDERSTOOD.
- ***sendPropose(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to PROPOSE.
- ***replyWithPropose(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to PROPOSE.
- ***sendQueryIf(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to QUERY_IF.
- ***replyWithQueryIf(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to QUERY_IF.
- ***sendQueryRef(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to QUERY_REF.
- ***replyWithQueryRef(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to QUERY_REF.
- ***sendRefuse(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REFUSE.
- ***replyWithRefuse(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REFUSE.
- ***sendRejectProposal(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REJECT_PROPOSAL.
- ***replyWithRejectProposal(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REJECT_PROPOSAL.
- ***sendRequest(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REQUEST.
- ***replyWithRequest(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REQUEST.
- ***sendSubscribe(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to SUBSCRIBE.
- ***replyWithSubscribe(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to SUBSCRIBE.

7.3.3. How to use it?

If we want to create an agent with the functionalities of the BaseAgent, we only have to extend the BaseAgent class. But extending the BaseAgent class implies two more actions:

1. Implement the **configure method**: This method allows us to add behaviors to the agent.

2. In the **agent creation**, it is necessary to pass the **ontology** as the first argument.

The initial project provided to students has two examples of agents that use the BaseAgent functionalities.

- **AgPlatform**: Example of a platform agent.

AgPlatform.java

```java
package org.game.agents.platform;

import ...

4 usages  Adrián Sánchez Rodero
public class AgPlatform extends BaseAgent {

    //1) Instantiate GameHttpClient
    2 usages
    private final GameHttpClient gameClient = new GameHttpClient();

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {

        this.addBehaviour((OneShotBehaviour) () -> {
            AgPlatform agent = (AgPlatform) this.myAgent;

            MessageTemplate template = MessageTemplate.and(
                    MessageTemplate.MatchLanguage(codec.getName()),
                    MessageTemplate.MatchOntology(ontology.getName()));
            template = MessageTemplate.and(template, MessageTemplate.MatchProtocol(PROTOCOL));

            //Wait a message with the protocol InitialCommunication
            ACLMessage msg = agent.blockingReceive(template);
            Concept1 content = agent.extractConcept(msg, Concept1.class);

            logger.info("I have received a message from " + msg.getSender().getLocalName());
            logger.info("The content of the message is: \"" + content.getAt1() + "\"");

            Concept1 contentReply = new Concept1();
            contentReply.setAt1("OK");

            //Reply the message with an AGREE performative
            agent.replyWithAgree(msg, contentReply);
            logger.info("I have sent an AGREE reply to " + TEAM_1);
            logger.info("The content of the reply is \"" + contentReply.getAt1() + "\"");


            //Configure the board and start the game
            agent.play();
        });
    }

    1 usage  Adrián Sánchez Rodero
    private void play() {
        //2) Use the method of the client
        //Initialize the map
        logger.info("I am going to initialize the map with 1 player");
        this.gameClient.initializeMap(new ConfigRequest( numPlayers: 1));

        //Start the game
        logger.info("I am going to start the game");
        this.gameClient.startGame();
    }
}
```

Figure 7.1: AgPlatform class

- **AgTeam**: Example of a team agent.

AgTeam.java

```java
package org.game.agents.teamN;

import ...

2 usages  Adrián Sánchez Rodero
public class AgTeam extends BaseAgent {

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {

        SequentialBehaviour steps = new SequentialBehaviour();
        steps.addSubBehaviour((OneShotBehaviour) () -> {
                AID platformAgent = new AID(PLATFORM_MANAGER, AID.ISLOCALNAME);
                Concept1 content = new Concept1();
                content.setAt1("I am alive");

                //Send a message to AgPlatform with the INFORM performative and InitialCommunication protocol
                ((BaseAgent) this.myAgent).sendInform(PROTOCOL, List.of(platformAgent), content);
                logger.info("I have sent an INFORM message to " + PLATFORM_MANAGER);
                logger.info("The content of the message is: \"" + content.getAt1() + "\"");
        });

        steps.addSubBehaviour((OneShotBehaviour) () -> {
                MessageTemplate template = MessageTemplate.and(
                        MessageTemplate.MatchLanguage(codec.getName()),
                        MessageTemplate.MatchOntology(ontology.getName()));
                template = MessageTemplate.and(template,
                        MessageTemplate.MatchProtocol(PROTOCOL));

                //Wait the response of the InitialCommunication protocol message
                ACLMessage msg = this.myAgent.blockingReceive(template);
                Concept1 content = ((BaseAgent) this.myAgent).extractConcept(msg, Concept1.class);

                logger.info("I have received a message from " + msg.getSender().getLocalName());
                logger.info("The content of the message is: \"" + content.getAt1() + "\"");
        });

        this.addBehaviour(steps);
    }
}
```

Figure 7.2: AgTeam class

The AgentsApplication class creates these two example agents in the **loadMyPlatformAgents** and **loadMyTeamAgents** methods passing the ontology as the first argument (7.2).

```
private static void loadMyPlatformAgents() {
    ProfileImpl agentContainerProfile = new ProfileImpl( host: null,  port: 1200,  platformID: null);
    agentContainerProfile.setParameter(Profile.CONTAINER_NAME, PLATFORM_MANAGER);
    cc = rt.createAgentContainer(agentContainerProfile);

    try {
        //TODO Change ontology: MockOntology for real ontology
        cc.createNewAgent(PLATFORM_MANAGER, AgPlatform.class.getName(),
                new Object[]{MockOntology.getInstance()}).start();
    } catch (StaleProxyException e) {
        System.err.println("Error creating platform agents!!!");
        System.exit( status: 1);
    }
}
```

Figure 7.3: Creation of AgPlatform in AgentsApplication class

```
private static void loadMyTeamAgents() {
    ProfileImpl agentContainerProfile = new ProfileImpl( host: null,  port: 1200,  platformID: null);
    agentContainerProfile.setParameter(Profile.CONTAINER_NAME, TEAM_1);
    cc = rt.createAgentContainer(agentContainerProfile);

    try {
        //TODO Change ontology: MockOntology for real ontology
        cc.createNewAgent(TEAM_1, AgTeam.class.getName(), new Object[]{MockOntology.getInstance()}).start();
    } catch (StaleProxyException e) {
        System.err.println("Error creating my team agents!!!");
        System.exit( status: 1);
    }
}
```

Figure 7.4: Creation of AgTeam in AgentsApplication class

After creation, these two agents try to establish a conversation: First, AgTeam **sends an INFORM message** using the **InitialCommunication protocol** to AgPlatform. Then AgPlatform receives the messages and **replies with an AGREE**.

There are two ways to add a new agent to the environment.

a. Create the agent in the **AgPlatform** class (if it is a platform agent) or in the **AgTeam** class (if it is a team agent) in the **configure** method. The following box has the code to do that:

```
try {
    this.getContainerController().createNewAgent(agentName,
     ↪  NewAgent.class.getName(), new
     ↪  Object[]{MockOntology.getInstance()}).start();
} catch (StaleProxyException e) {
    System.err.println("Error creating the agent!!!");
    System.exit(1);
}
```

From the above code, it is only necessary to change the name of the agent, use the corresponding agent class and pass the real ontology in the objects array.

b. Create the agent in the **loadMyPlatformAgents** method (if it is a platform agent) or in the **loadMyTeamAgents** (if it is a team agent) method of the

AgentsApplication class. In this case, the necessary code is similar to the creation of AgPlatform and AgTeam agents.

7.4. GameHttpClient

7.4.1. What is the GameHttpClient?

The GameHttpClient acts as an **intermediary** between agents and the frontend interface. This client allows the platform agents to **communicate with the interface** to: initialize the map, start the game, end the game and show the final results, create a new unit, move a unit, attack another unit, update the health of a unit, or kill a unit. It performs the necessary HTTP requests and handles the responses.

The GameHttpClient is implemented according the API definition in the **Agents Game - API document** (12.2).

7.4.2. How is it implemented?

This section describes how GameHttpClient is implemented and some preculiarities of the implementation of its unit tests.

7.4.2.1. Client

The GameHttpClient implements the **InterfaceCommunication interface** that defines the methods to communicate with the frontend part. The InterfaceCommunication interface defines the following methods:

- ***initializeMap(ConfigRequest request)***: Initializes the map with the number of players.
- ***startGame()***: Starts a new game phase, initializing phase-specific actions.
- ***endGame(EndGameRequest request)***: Ends the current game phase and shows final results.
- ***createUnit(UnitRequest request)***: Creates a new unit at the specified coordinates and assigns it to a player.
- ***move(MoveRequest request)***: Moves a unit from one position to another on the game board.
- ***attack(AttackRequest request)***: Performs an attack action between two units and updates the attacked unit's health points.
- ***updateUnitHealth(UpdateHealthRequest request)***: Updates the health of a unit with the new value.
- ***killUnit(KillUnitRequest request)***: Removes a unit from the board.

To implement these methods, GameHttpClient uses the **Gson library** [12] for the request bodies. Gson is a Java library that can be used to convert Java Objects into their JSON representation.

The **request bodies** used in the communications are defined in the dto folder of the project and have the following attributes:

- **AttackRequest**:
  - **attackerId**: Unique identifier of the unit that performs the attack (String).
  - **targetId**: Unique identifier of the unit that receives the attack (String).
  - **damage**: Integer that indicates the damage done by the attack (int).
  - **targetHealthLeft**: Remaining target unit health (int).
- **ConfigRequest**:
  - **numPlayers**: Number of players of the game (int).
- **EndGameRequest**:
  - **teams**: List with the result of each team in the game (List<TeamResult>).
- **MoveRequest**:
  - **id**: Unique identifier of the unit to move (String).
  - **toX**: New X position of the unit (int).
  - **toY**: New Y position of the unit (int).
- **TeamResult**:
  - **teamName**: Team identifier (String).
  - **points**: Total points scored (int).
- **UnitRequest**:
  - **x**: X position of the unit (int).
  - **y**: Y position of the unit (int).
  - **id**: Unique identifier of the unit (String).
  - **type**: Integer that indicates the type of the unit (int).
  - **team**: Team affiliation (int).
- **UpdateHealthRequest**:
  - **unitId**: Unique identifier of the unit (String).
  - **healthLeft**: New health value (int).
- **KillUnitRequest**:
  - **unitId**: Unique identifier of the unit to kill (String).

7.4.2.2. Unit tests

The unit tests of GameHttpClient (**GameHttpClientTest**) are different from other classes. GameHttpClientTest uses the **wiremock library** [13] to test that the client performs the different requests correctly. This library allows us to create a mock server that responds to the requests emitted by GameHttpClient.

GameHttpClientTest **initializes the mock server** before starting to run the test cases. First, it creates a new WireMockServer in a specified port, and then it starts the server.

```
@BeforeAll
static void init() {
    wireMockServer = new WireMockServer(PORT);
    wireMockServer.start();
}
```

After all tests have been run, it **stops the server**.

```
@AfterAll
static void endTest() {
    wireMockServer.stop();
}
```

In each test case, we have to **mock the response of the server** to the corresponding endpoint. For example, in the next code we are mocking the response of */initializeMap* endpoint to test the initializeMap method of GameHttpClient.

```
wireMockServer.stubFor(post(urlEqualTo(INIT_MAP))
    .withHeader("Content-Type", equalTo("application/json"))
    .willReturn(aResponse().withStatus(200)));
```

And, to **validate** that the server has received a **request at that endpoint**, we have the WireMockServer verify method. In the following code, we are validating that the server has received a POST request at the */initializeMap* endpoint with the corresponding header.

```
wireMockServer.verify(postRequestedFor(urlEqualTo(INIT_MAP))
   .withHeader("Content-Type", equalTo("application/json")));
```

7.4.3. How to use it?

If we want to communicate with the interface to perform an action, we have to follow these steps:

1. **Instantiate GameHttpClient** in the corresponding class.

   ```
   private final GameHttpClient gameClient = new
    ↪  GameHttpClient();
   ```

2. **Use the method of the client** associated with the action we want to perform passing the required arguments.

   ```
   //Initialize the map
   this.gameClient.initializeMap(new ConfigRequest(1));
   ```

The initial project provided to students has an example of the usage of the GameHttpClient in AgPlatform. This agent, after handling the AgTeam message, uses GameHttpClient to initialize the map and start the game (in the AgPlatform **play method**, we can see the implementation of step 2 above).

AgPlatform.java

```java
package org.game.agents.platform;

import ...

4 usages  Adrián Sánchez Rodero
public class AgPlatform extends BaseAgent {

    //1) Instantiate GameHttpClient
    2 usages
    private final GameHttpClient gameClient = new GameHttpClient();

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {...}

    1 usage  Adrián Sánchez Rodero
    private void play() {
        //2) Use the method of the client
        //Initialize the map
        logger.info("I am going to initialize the map with 1 player");
        this.gameClient.initializeMap(new ConfigRequest( numPlayers: 1));

        //Start the game
        logger.info("I am going to start the game");
        this.gameClient.startGame();
    }
}
```

Figure 7.5: AgPlatform implementation of step 1 and 2

7.5. Log standardization

7.5.1. What is the log standardization?

The log standardization is a functionality to **print messages** on the program console with a standard format. These messages help developers to **debug the code** in case of an error or to check that everything is working correctly.

The logs have a **standard format** that contains the date, the level, the name of the source, and the message.

```
[yyyy-MM-dd HH:mm:ss] LEVEL SourceName: Message
```

There are three possible **levels** for the logs:

- **INFO**: Used to print decision/operation messages. INFO level messages are printed in different colors depending on the source of the message.

- **Platform agents** print their messages in **white** color.

Figure 7.6: Example of platform log

- **GameHttpClient** prints its messages in **magenta** color.

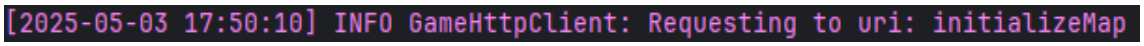


Figure 7.7: Example of GameHttpClient log

- **Team agents** print their messages in the **color associated with the team**.

- **WARN**: Used to print warning messages. It is associated with the **yellow** color.

Figure 7.8: Example of warning log

- **ERROR**: Used to print error messages. It is associated with the **red** color.

Figure 7.9: Example of error log

7.5.2. How is it implemented?

The log standardization functionality is implemented using the **log4j logging framework** [14]. This framework allows us to have the three levels of logs already defined in three methods.

- **info(Object message)**: To print information messages.
- **warn(Object message)**: To print warning messages.
- **error(Object message)**: To print error messages.

To customize the logs with the format and color desired, we have created a custom Layout class and override the format method. This implementation is in the **AgentLogLayout class** in the config folder. In this class, the private method **getColorForLevelOrAgent(String name, String level)** is in charge of assigning the color to the logs depending on the **level** and the **logger name**.

- If the level of the message is **ERROR** then the color is red.
- If the level of the mesage is **WARN** then the color is yellow.
- If the logger name is **equal to GameHttpClient** then the color is magenta.
- If the logger name **does not contain the word 'Team'** (Platform agent) then the color is white.

- Otherwise (Team agents), the method extracts the team identifier from the name (Unit5Team2 → Team2) and checks if the team already has a color. If it has a color, then the log uses it. If not, assign a color.

The **LogColors class** in the utils folder contains the declaration of the colors. There are **12 different colors reserved for teams**. They are assigned in sequence, and if there are more than 12 teams, then the first color is assigned again.

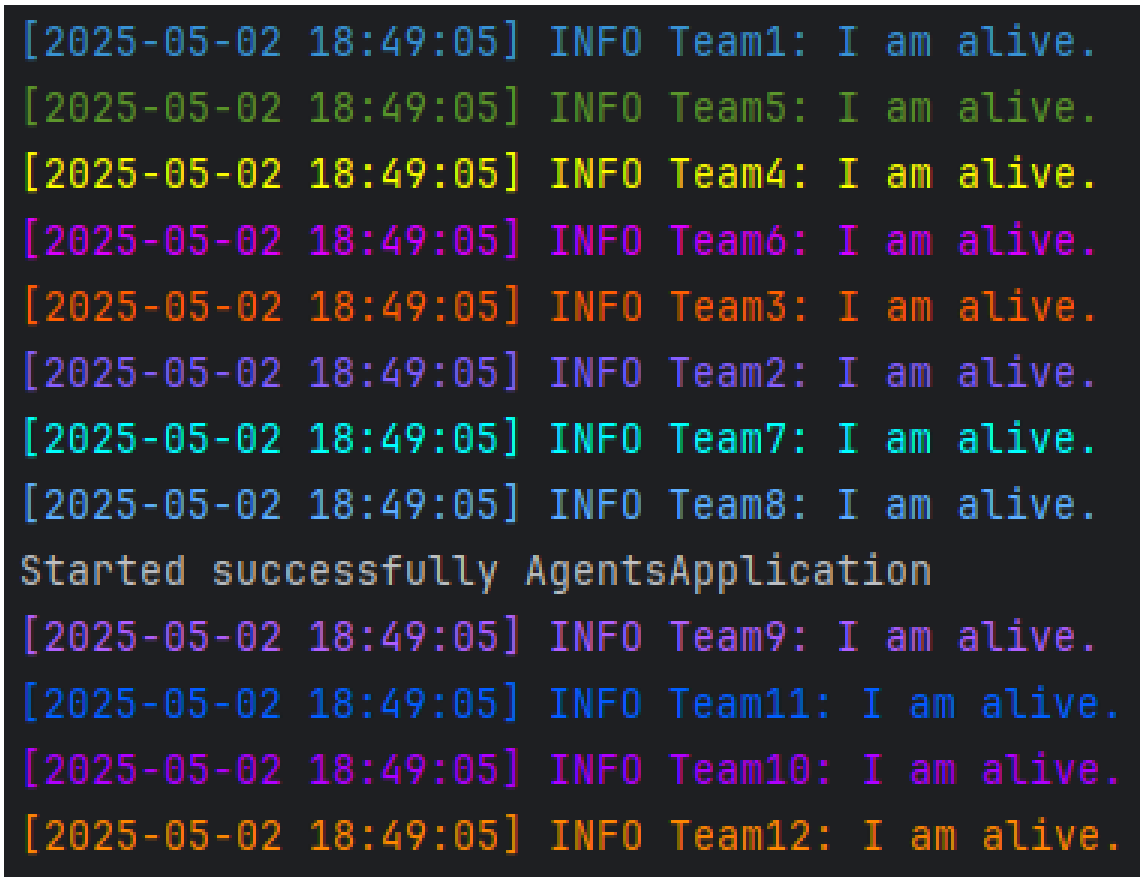


Figure 7.10: Example of 12 team agents logs

Finally, the log4j logging framework needs some properties to indicate the new Layout class. These properties are defined in the **log4j.properties** file of the resources folder.

7.5.3. How to use it?

To use the log standardization functionality we have to distinguish two cases:

- In an **agent**: The BaseAgent implementation has already integrated the functionality in its **protected logger property**. So, its child classes only need to call the log4j methods (info, warn and error) to print a message. The logger layout will assign the color depending on the level of the message and the agent name. In this aspect, it is **very important** that **team agents names** contain the word **'Team'** following by the identifier of the team (Unit5Team1, Team4Unit1, ManTeam2Unit1...), and the **platform agents names** do **not** contain the **'Team' word** to avoid possible problems.

- In **another class**: In this case, it is necessary to **initialize the logger** with a logger name in the class, **add the color** we want for the new class in the LogColors class and **add a new if statement** in the getColorForLevelOrAgent method of the AgentLogLayout to return the new color. The **GameHttpClient** logger is an example of this.

```
import org.apache.log4j.Logger;
```

Figure 7.11: Example of logger import

```
private final Logger logger = Logger.getLogger( name: "GameHttpClient");
```

Figure 7.12: Example of GameHttpClient logger initialization

```
public static final String CLIENT_COLOR = "\u001B[38;5;213m";   // Bright magenta
```

Figure 7.13: Example of GameHttpClient color

```
private String getColorForLevelOrAgent(String name, String level) {
    switch (level) {
        case "ERROR":
            return LogColors.RED;
        case "WARN":
            return LogColors.YELLOW;
        case "INFO":
        default:
            if (name.equals("GameHttpClient")) {
                return LogColors.CLIENT_COLOR;
            }
            if (!name.contains("Team")) {
                return LogColors.PLATFORM_COLOR;
            }
            String teamId = name.replaceAll( regex: ".*(Team\\d+).*", replacement: "$1");
            return teamColors.computeIfAbsent(teamId, k -> LogColors.TEAM_COLORS[teamColors.size() %
                    LogColors.TEAM_COLORS.length]);
    }
}
```

Figure 7.14: Example of GameHttpClient if statement

8 Results

This chapter presents the results for each sprint, the global result, and the conclusions of the work. For each sprint, its corresponding section contains the sprint burndown chart, the state of each user stories at the end of the sprint (completed, in progress or pending), the products generated, and the retrospective table with the evaluation of the sprint.

8.1. Sprint 1

8.1.1. Sprint burndown chart

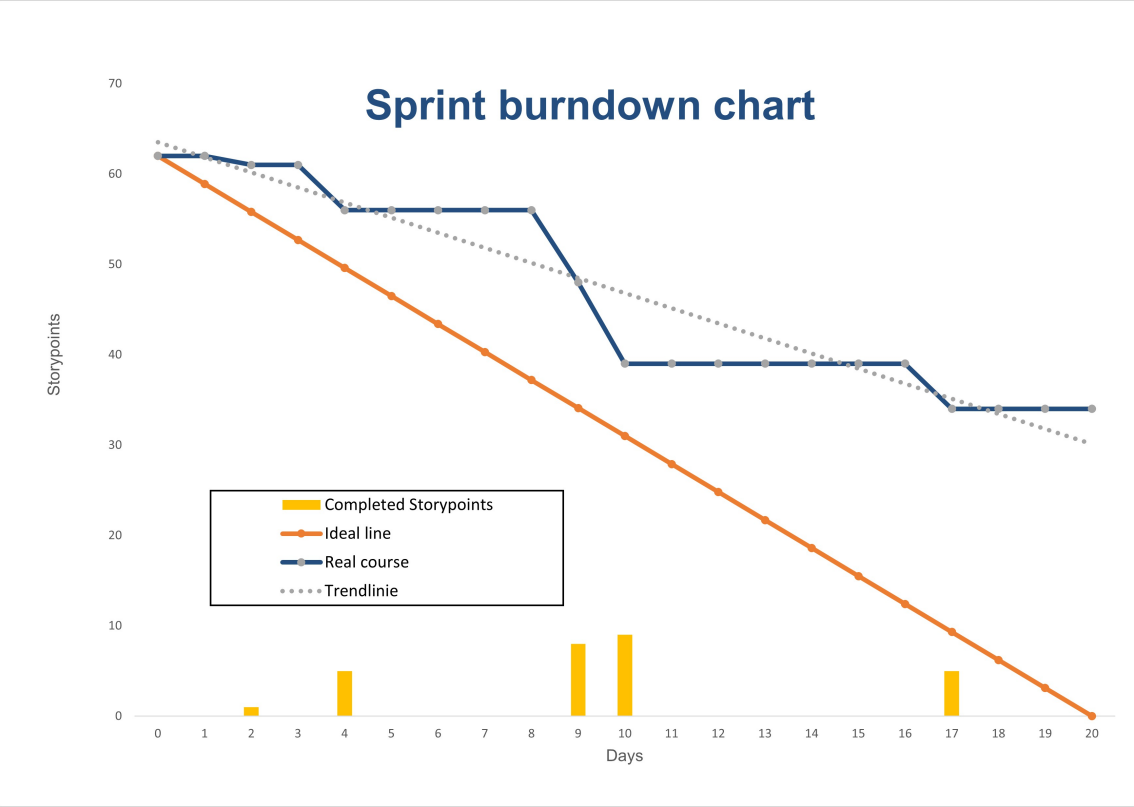


Figure 8.1: Burndown chart Sprint 1

	Completed	In progress	Remaining
Stories	8	1	8
Story points	28	8	26

Table 8.1: Table with the state of the work at the end of Sprint 1

As a summary of the work done in Sprint 1, we can consider the following three metrics:

- **Number of epics completed:** 0

- **Duration of the sprint:** 18 days
- **Sprint velocity:** 1,55 story points per day

8.1.2. State of the user stories

8.1.2.1. Completed

- **US 1.1 Set up of the ontology project (4.2.1.1):** AS a developer, I WANT to create the ontology project with the basic structure IN ORDER TO store the generator.
- **US 1.3 Parse the ontology specification file (4.2.1.3):** AS a developer, I WANT to parse the ontology specification txt file IN ORDER TO build the ontology.
- **US 1.4 Generate a concepts (4.2.1.4):** AS a developer, I WANT to generate a concept from the ontology specification file IN ORDER TO build the ontology.
- **US 1.5 Generate an action (4.2.1.5):** AS a developer, I WANT to generate an action from the ontology specification file IN ORDER TO build the ontology.
- **US 1.6 Generate a predicate (4.2.1.6):** AS a developer, I WANT to generate a predicate from the ontology specification file IN ORDER TO build the ontology.
- **US 1.7 Generate the ontology vocabulary file (4.2.1.7):** AS a developer, I WANT to generate the ontology vocabulary file from the ontology specification file IN ORDER TO build the ontology.
- **US 1.2 Define the specification of the ontology (4.2.1.2):** AS a developer, I WANT to define the specification of the ontology in a txt file IN ORDER TO specify the contents of the ontology.
- **US 1.17 Documentation of the sprint (4.2.1.17):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.1.2.2. In progress

- **US 1.8 Generate the ontology file (4.2.1.8):** AS a developer, I WANT to generate the ontology file from the ontology specification file IN ORDER TO build the ontology.

8.1.2.3. Pending

- **US 1.9 Design a basic user interface for the Ontology Generator (4.2.1.9):** AS a developer, I WANT to design a basic user interface for the ontology IN ORDER TO build the ontology from user inputs.

- **US 1.10 Generate the ontology from a specification file using the interface (4.2.1.10):** AS a developer, I WANT to generate the ontology from a specification file IN ORDER TO build the ontology from the user interface.
- **US 1.11 Generate a concept with the user interface (4.2.1.11):** AS a developer, I WANT to generate a concept from user inputs IN ORDER TO build the ontology from the user interface.
- **US 1.12 Generate an action with the user interface (4.2.1.12):** AS a developer, I WANT to generate an action from user inputs IN ORDER TO build the ontology from the user interface.
- **US 1.13 Generate a predicate with the user interface (4.2.1.13):** AS a developer, I WANT to generate a predicate from user inputs IN ORDER TO build the ontology from the user interface.
- **US 1.14 Update ontology vocabulary file after each modification of the ontology with the user interface (4.2.1.14):** AS a developer, I WANT to update the ontology vocabulary file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.
- **US 1.15 Update ontology file after each modification of the ontology with the user interface (4.2.1.15):** AS a developer, I WANT to update the ontology file after each modification of the ontology performed through the user interface IN ORDER TO build the ontology from the user interface.
- **US 1.16 Documentation of the Ontology Generator (4.2.1.16):** AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.

8.1.3. Products

- **Ontology Generator: User Manual v1.2**: The result of US 1.2 Define the specification of the ontology (4.2.1.2) was a description of the structure of the ontology specification file and some examples (5.3). This information was added to the Ontology Generator: User Manual (12.1) to obtain the version 1.2.
- **First class design of the Ontology Generator:** In order to start developing the Ontology Generator, it was necessary to have a global vision of its structure. To cover that need, the following class diagram was designed. Also, in this class diagram we can see the state of the implementation at the end of Sprint 7. The detailed explanation of each part of the class diagram can be found in section 5.1.

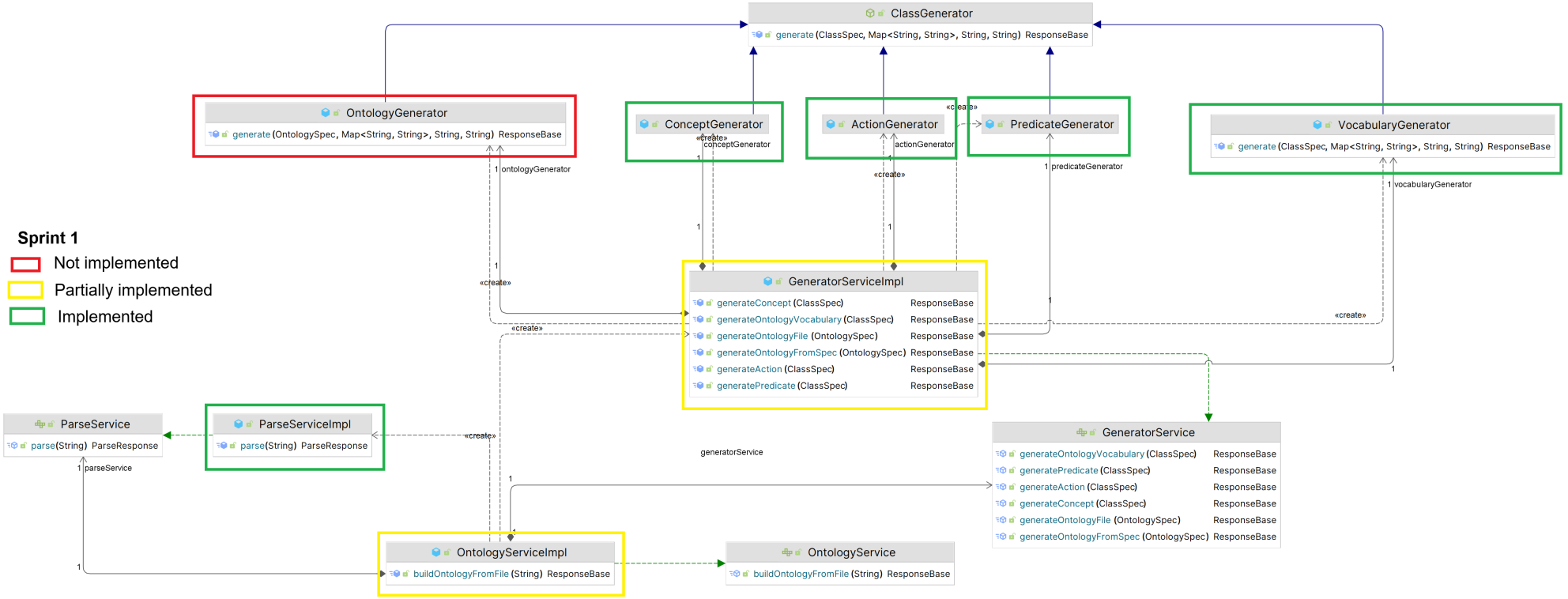


Figure 8.2: First class design of Ontology Generator

- **Ontology Generator response codes:** During the implementation of the Ontology Generator in this sprint, different paths have arisen in the execution of the program depending on the program arguments and system conditions. For this reason, it was decided to standardize the possible response codes. They are listed in the corresponding section of the Ontology Generator product (5.5).

- **First version of the Ontology Generator application**: A program that generates concepts, actions, predicates, and vocabulary of the ontology given an ontology specification file.

8.1.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Organization in services and generators. ▪ Management of errors. ▪ First implementation of the Ontology Generator.	▪ Too many story points in the sprint. ▪ Test the generation of files.	▪ Limit the scope of the sprint.

8.2. Sprint 2

8.2.1. Sprint burndown chart

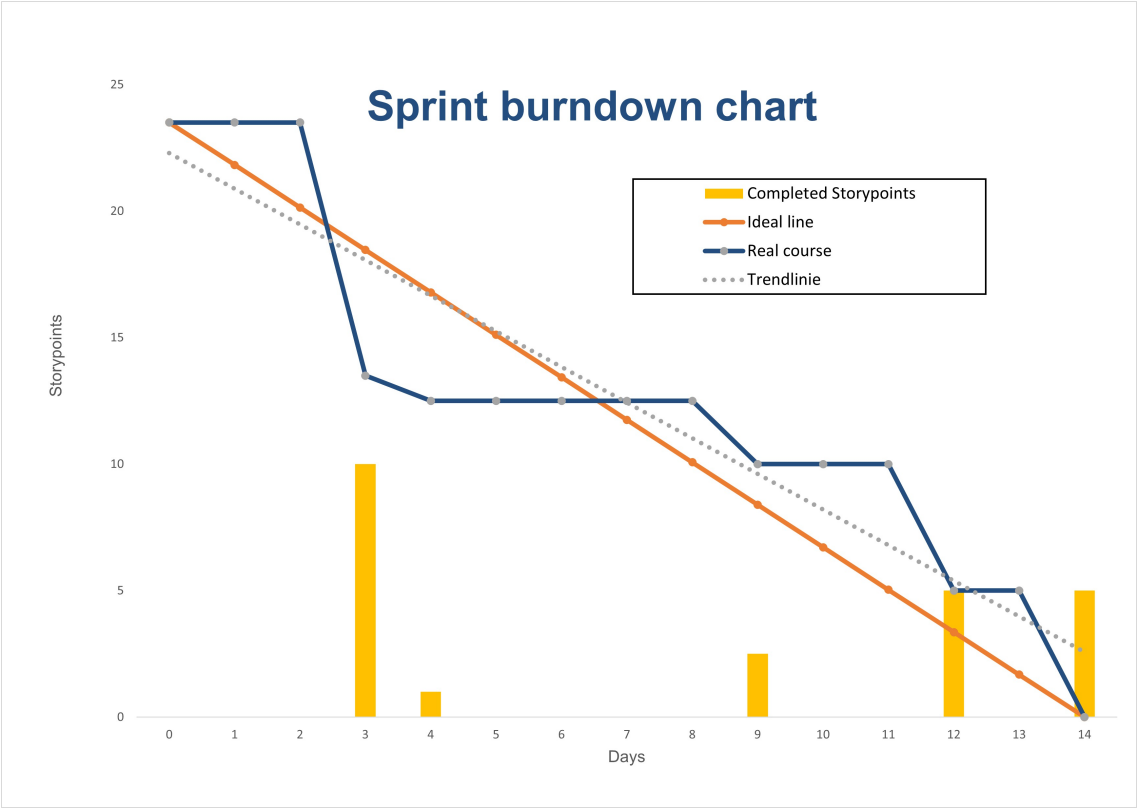


Figure 8.3: Burndown chart Sprint 2

	Completed	In progress	Remaining
Stories	7	0	0
Story points	23,5	0	0

Table 8.2: Table with the state of the work at the end of Sprint 2

As a summary of the work done in Sprint 2, we can consider the following three metrics:

- **Number of epics completed:** 0
- **Duration of the sprint:** 14 days
- **Sprint velocity:** 1,67 story points per day

8.2.2. State of the user stories

8.2.2.1. Completed

- **US 1.8 Generate the ontology file (4.2.1.8):** AS a developer, I WANT to generate the ontology file from the ontology specification file IN ORDER TO build the ontology.

- **Error 2.4 Fix constant names in ontology vocabulary file (4.3.1.4):** AS a developer, I WANT to fix the construction of constant names IN ORDER TO have a more readable ontology vocabulary file.
- **US 2.1 Define the arguments of the Ontology Generator application (4.3.1.1):** AS a developer, I WANT to define the arguments of the ontology generator application IN ORDER TO pass the necessary information to the application.
- **US 2.2 Generate a JAR file with the Ontology Generator application (4.3.1.2):** AS a developer, I WANT to generate a JAR file with the ontology generator application IN ORDER TO run the application easily.
- **US 1.16 Documentation of the Ontology Generator (4.2.1.16):** AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.
  - **Task 1.16.1 Documentation of how to generate an ontology from a specification file (4.2.1.16.1):** Indicate the steps to generate an ontology from a specification file using the Ontology Generator.
- **US 1.9 Design a basic user interface for the ontology (4.2.1.9):** AS a developer, I WANT to design a prototype of a user interface for the ontology IN ORDER TO build the ontology from user inputs.
- **US 2.3 Documentation of the sprint (4.3.1.3):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.2.2.2. In progress

No remaining user stories in progress.

8.2.2.3. Pending

No remaining user stories pending.

8.2.3. Products

- **Second version of the class design of the Ontology Generator:** During this sprint the implementation of the Ontology Generator to generate an ontology from a specification file has been completed. The functionality to generate the ontology file has been developed (OntologyGenerator and GeneratorServiceImpl classes). In addition, as result of the US 2.1 Define the arguments of the Ontology Generator application (4.3.1.1), the GeneratorService and OntologyService have incorporated two methods (setPackageDeclaration and setDestinationPath) to manage the input arguments. The details of the methods are described in the Ontology Generator class diagram section (5.1).

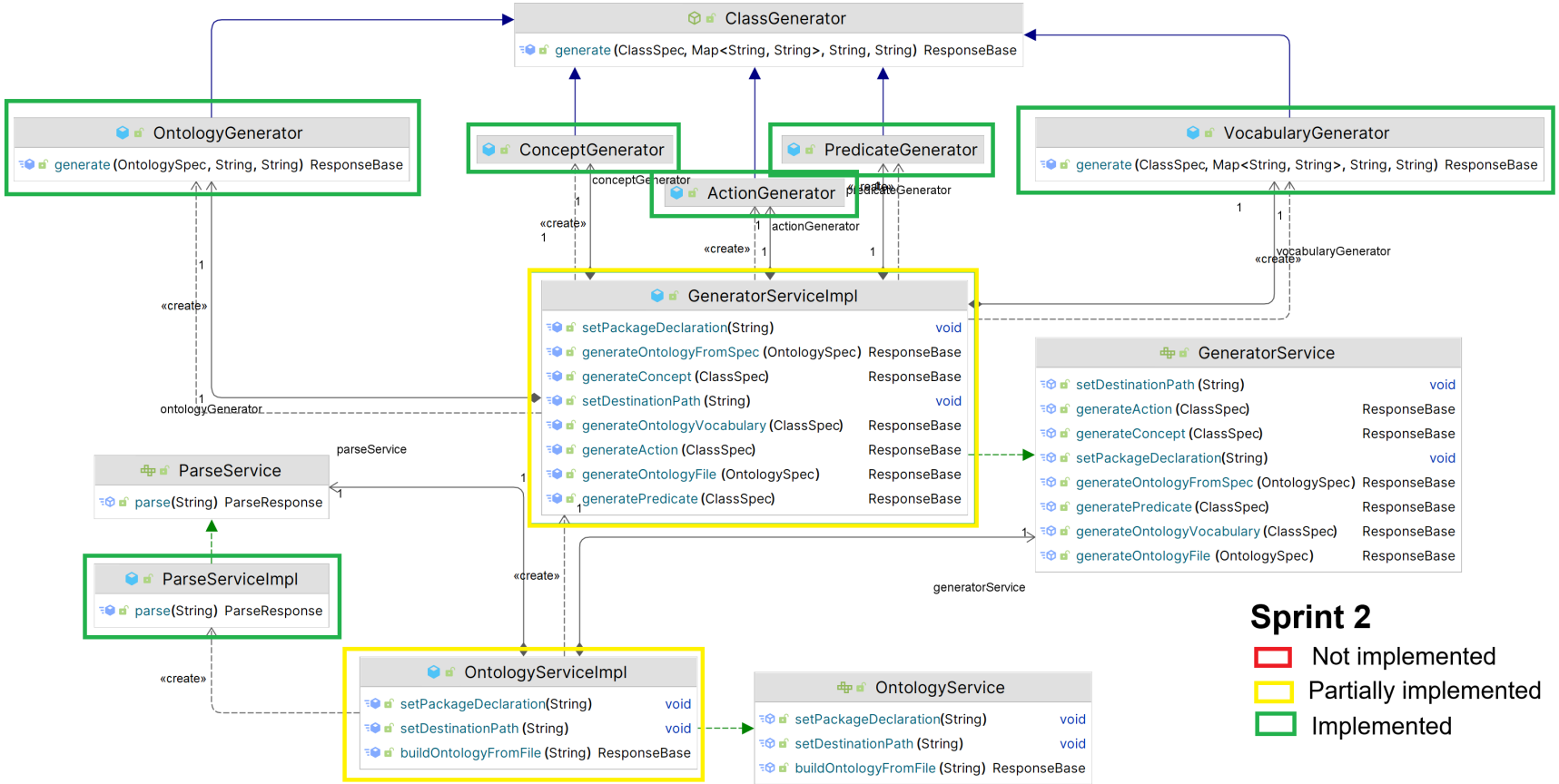


Figure 8.4: Second class design of Ontology Generator

The OntologyServiceImpl and GeneratorServiceImpl are still incomplete due to missing GUI functionality that will be developed in next sprints.

- **Test plan:** Since in this sprint we have obtained a minimum viable product of the Ontology Generator application (generate an ontology from a specification file), to ensure that all the functionality implemented is working as expected, a test plan has been developed using a black-box technique. This test plan is described in the Test Plans section of this document (6).

- **Ontology Generator application:** Minimum viable product of the Ontology Generator. At the moment, the application allows to generate an ontology from a specification file.

- **Ontology Generator: User Manual v1.4:** The manual (12.1) has incorpored a new chapter related to the configurations and steps needed to run the Ontology Generator application in order to generate an ontology from a specification file.

- **First version of the Ontology Generator interface design:** The result of US 1.9 Design a basic user interface for the Ontology Generator (4.2.1.9) was the design of the screens for the user interface (5.4.1).

- **First version of the Ontology Generator prototype:** Also another result of US 1.9 Design a basic user interface for the Ontology Generator (4.2.1.9) was a prototype of the user interface (5.4.2).

8.2.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Ontology Generator application first functional product. ▪ First design of the Ontology Generator interface. ▪ First prototype of the Ontology Generator interface.		

8.3. Sprint 3

8.3.1. Sprint burndown chart

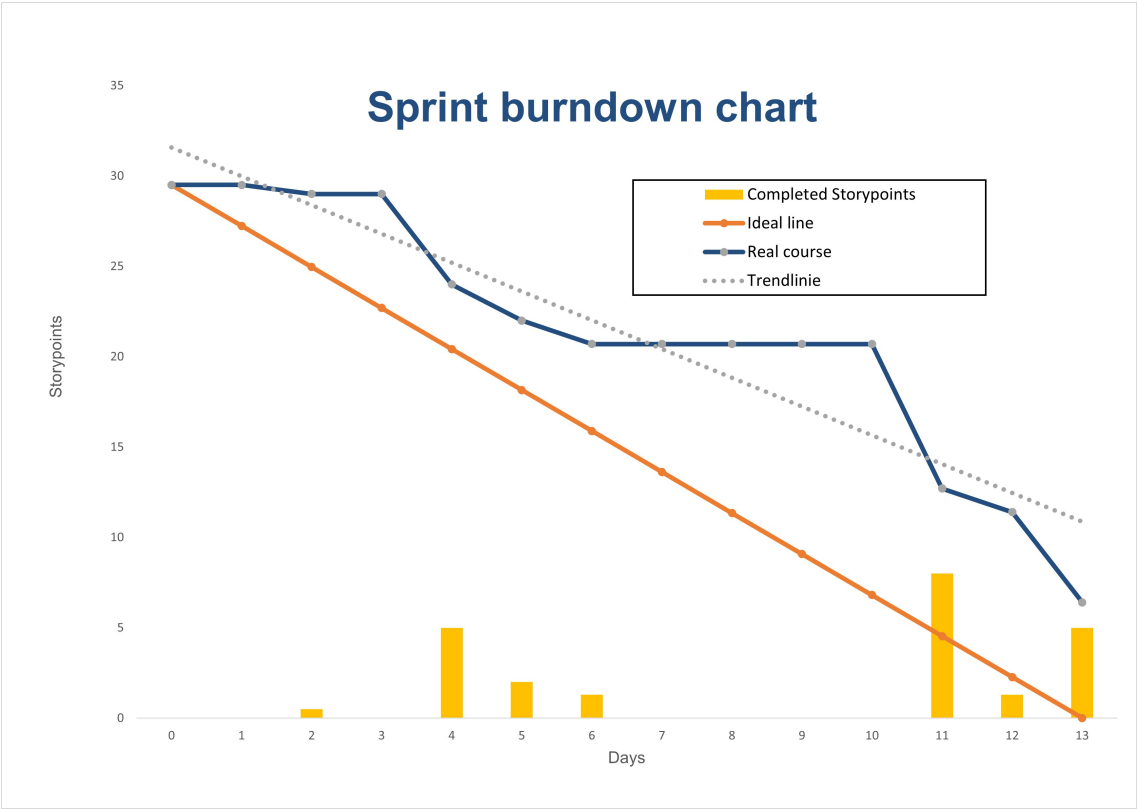


Figure 8.5: Burndown chart Sprint 3

	Completed	**In progress**	**Remaining**
Stories	6	1	1
Story points	23,1	5	$1,\overline{3}$

Table 8.3: Table with the state of the work at the end of Sprint 3

As a summary of the work done in Sprint 2, we can consider the following three metrics:

- **Number of epics completed:** 0
- **Duration of the sprint:** 13 days
- **Sprint velocity:** 1,77 story points per day

8.3.2. State of the user stories

8.3.2.1. Completed

- **US 3.2 Documentation response codes of how to generate an ontology from a specification (4.4.1.2):** AS a developer, I WANT to add to the On-

tology Generator: User Manual the description of the response codes IN ORDER TO have a complete documentation.

- **US 3.3 Documentation Ontology Generator from a specification file in final memory (4.4.1.3):** AS a developer, I WANT to document the progress in the Ontology Generator in the final memory IN ORDER TO start with the final document.
- **US 3.4 Set up of the agents project (4.4.1.4):** AS a developer, I WANT to create the project IN ORDER TO store agents of the game.
- **US 3.5 Structure of the agents project (4.4.1.5):** AS a developer, I WANT to decide and implement the organization of the agents project IN ORDER TO define the Team-Platform environment.
- **US 3.7 Implementation of the BaseAgent (4.4.1.7):** AS a developer, I WANT to implement a base agent with message builders IN ORDER TO allocate the common code for agents.
- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.
  - **Task 3.6.1 Documentation of the structure of the agents project (4.4.1.6.1):** Describe the organization of the agents project.
    - Platform agents.
    - Team agents.
    - API client.
    - BaseAgent.
  - **Task 3.6.2 Documentation of the implementation of BaseAgent (4.4.1.6.2):** Describe the organization and methods of the BaseAgent.
- **US 3.1 Documentation of the sprint (4.4.1.1):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.3.2.2. In progress

- **US 3.8 Definition of API (4.4.1.8):** AS a developer, I WANT to define the API endpoints, request bodies, responses, and errors IN ORDER TO establish the basis for its implementation.

8.3.2.3. Pending

- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.
  - **Task 3.6.3 Documentation of how to create an agent using the BaseAgent (4.4.1.6.3):** Indicate the steps to create a new agent using the functionalities of the BaseAgent.

8.3.3. Products

- **Ontology Generator: User Manual v1.5:** The manual (12.1) has incorpored the description of the different response codes of the Ontology Generator.

- **Begining of the final memory:** Creation of this document, definition of the structure, and writing of the Ontology Generator part related to Sprint 1 and 2.

- **Team - Platform: Developer Manual v1.0:** First version of the developer manual (12.3) of the agents project. It has a description of the structure of the project and an explanation of the implementation of the BaseAgent.

- **Team - Platform environment:** First approach of the agents environment with the organization in folders (7.1) and the implementation of the abstract class BaseAgent (7.3) to group the common code for all agents.

8.3.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Ontology Generator application completed completed from the point of view of generate an ontology from a specification file. ▪ Structure of the Team - Platform environment defined. ▪ Final memory in progress	▪ Test the BaseAgent implementation. Too many methods and need to search solutions of how to test it.	

8.4. Sprint 4

8.4.1. Sprint burndown chart

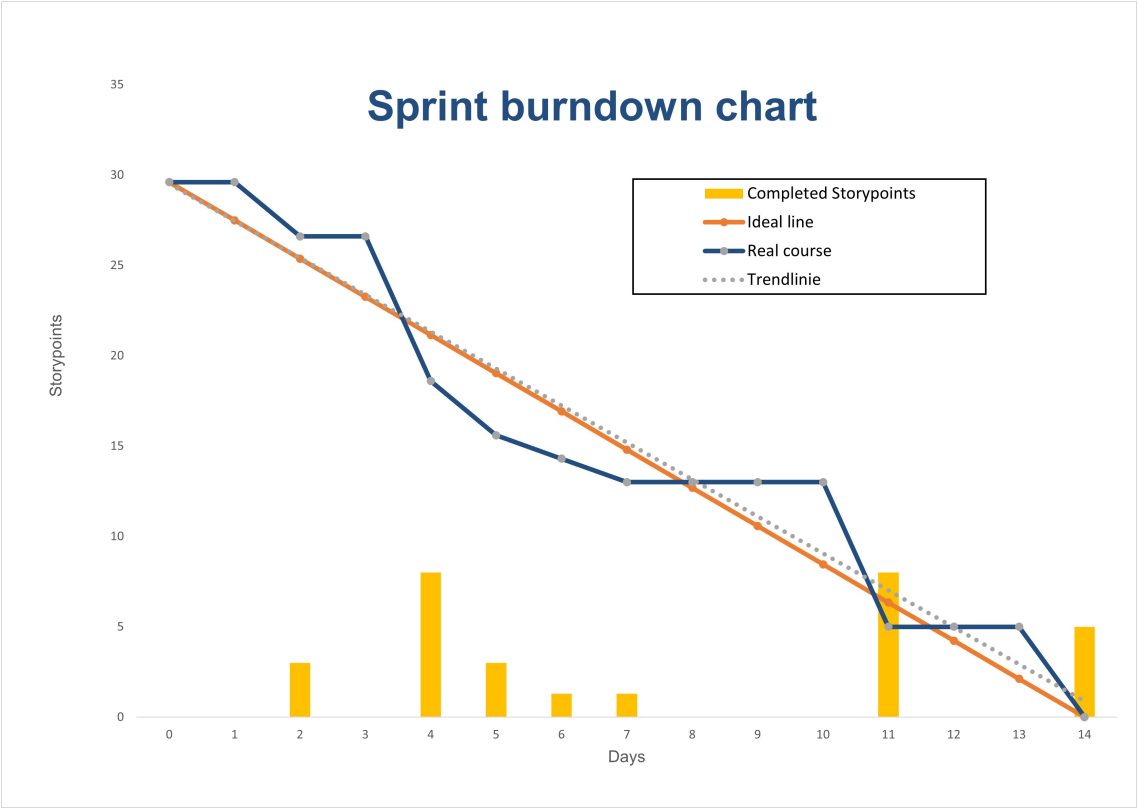


Figure 8.6: Burndown chart Sprint 4

	Completed	In progress	Remaining
Stories	7	0	0
Story points	29,6	0	0

Table 8.4: Table with the state of the work at the end of Sprint 4

As a summary of the work done in Sprint 4, we can consider the following three metrics:

- **Number of epics completed:** 1 (Environment definition)
- **Duration of the sprint:** 14 days
- **Sprint velocity:** 2,11 story points per day

8.4.2. State of the user stories

8.4.2.1. Completed

- **US 4.2 Documentation of Sprint 3 in final memory (4.5.1.2):** AS a developer, I WANT to document the progress in Sprint 3 in the final memory IN ORDER TO have all information in the final memory.

- **US 3.8 Definition of API (4.4.1.8):** AS a developer, I WANT to define the API endpoints, request bodies, responses, and errors IN ORDER TO establish the basis for its implementation.
- **US 4.3 Creation of Team - Platform example agents (4.5.1.3):** AS a developer, I WANT to create Team - Platform example agents IN ORDER TO illustrate Team - Platform agents.
- **US 4.4 Agent loading (4.5.1.4):** AS a developer, I WANT to load team and platform agents IN ORDER TO initialize the game.
- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.
  - **Task 3.6.3 Documentation of how to create an agent using the BaseAgent (4.4.1.6.3):** Indicate the steps to create a new agent using the functionalities of the BaseAgent.
  - **Task 3.6.6 Documentation of how to set up the project (4.4.1.6.6):** Indicate the necessary steps to set up the Team - Platform environment.
- **US 4.5 Implementation of API client (4.5.1.5):** AS a developer, I WANT to implement the API client following the definition IN ORDER TO establish the connection with the frontend.
- **US 4.1 Documentation of the sprint (4.5.1.1):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.4.2.2. In progress

No remaining user stories in progress.

8.4.2.3. Pending

No remaining user stories pending.

8.4.3. Products

- **Final memory:** With the US 4.2 Documentation of Sprint 3 in final memory (4.5.1.2), this document is aligned with the development process.
- **Agents Game - API v1.0:** As a result of US 3.8 Definition of the API (4.4.1.8), we have obtained a document with the definition of the API for the communications with the frontend part (12.2).
- **Team - Platform: Developer Manual v1.1:** The manual (12.3) has incorpored the AgentsApplication chapter, the BaseAgent chapter has been completed with How to use it? section, and the Set up chapter has been written.
- **Team - Platform environment:** The environment is ready for students to develop their agents. It has the BaseAgent with the MessageManager,

and examples of platform and team agents (7.3). Furthermore, the project includes the creation of agents' containers (7.2).

- **API client:** As a result of the US 4.5 Implementation of API client (4.5.1.5), the environment is prepared to communicate with the frontend part through the client implemented.

8.4.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Definition and implementation of the API for the communications with the frontend part. ▪ Team - Platform environment with the basic functionality ready for students to develop their agents.		

8.5. Sprint 5

8.5.1. Sprint burndown chart

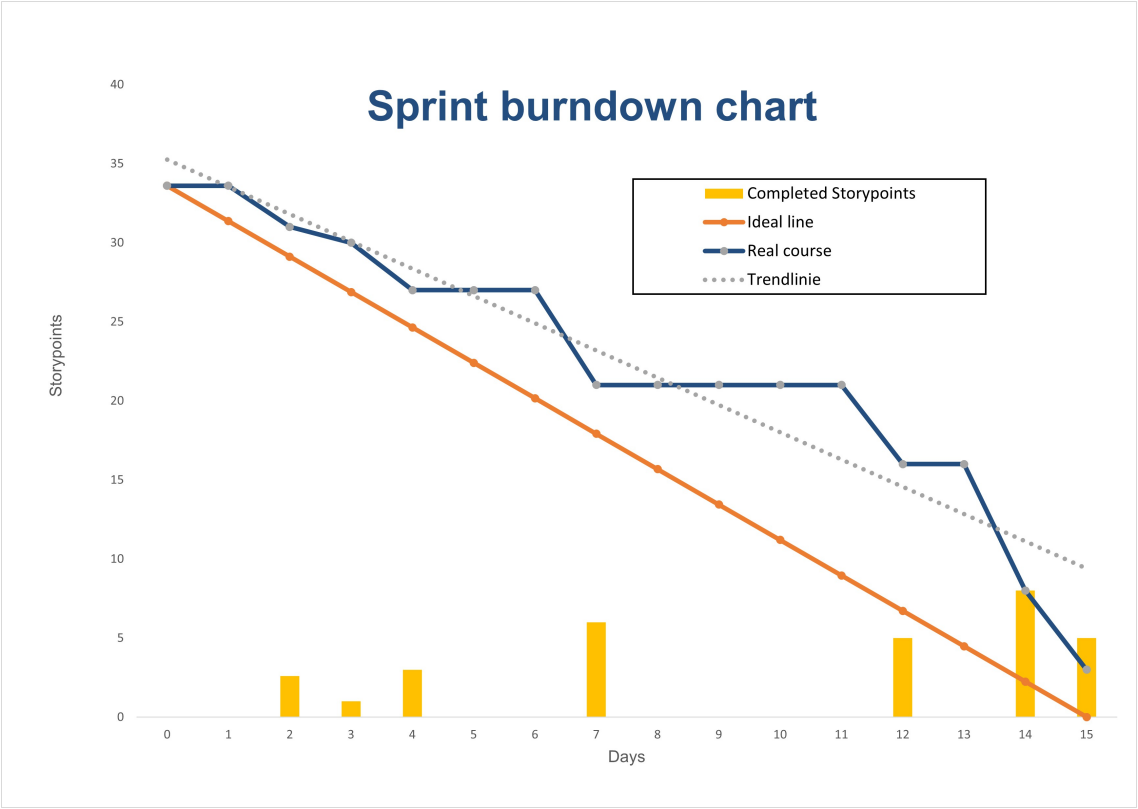


Figure 8.7: Burndown chart Sprint 5

	Completed	In progress	Remaining
Stories	8	0	1
Story points	30,6	0	3

Table 8.5: Table with the state of the work at the end of Sprint 5

As a summary of the work done in Sprint 5, we can consider the following three metrics:

- **Number of epics completed:** 0
- **Duration of the sprint:** 15 days
- **Sprint velocity:** 2,04 story points per day

8.5.2. State of the user stories

8.5.2.1. Completed

- **US 3.6 Documentation of the agents environment (4.4.1.6):** AS a developer, I WANT to document the agents project IN ORDER TO have a Team - Platform Environment: Developer Manual.

  - **Task 3.6.4 Documentation of implementation of API client (4.4.1.6.4):** Describe the organization and methods of the API client.
  - **Task 3.6.5 Documentation of how to use the API client (4.4.1.6.5):** Indicate the necessary actions to use the API client.

- **US 5.3 Modifications of the Ontology Generator interface (4.6.1.3):** AS a developer, I WANT to modify the design of the Ontology Generator interface IN ORDER TO add the changes proposed in the Sprint Review of Sprint 4.
- **US 5.4 Set up of the Ontology Generator interface (4.6.1.4):** AS a developer, I WANT to set up the Ontology Generator interface IN ORDER TO prepare the interface to add the screens.
- **US 5.5 Ontology Generator interface visual Main screen (4.6.1.5):** AS a developer, I WANT to implement the visual part of the Main screen IN ORDER TO develop the Ontology Generator interface.
- **US 1.10 Generate the ontology from a specification file using the interface (4.2.1.10):** AS a developer, I WANT to generate the ontology from a specification file IN ORDER TO build the ontology from the user interface.
- **US 5.6 Ontology Generator interface visual New concept (4.6.1.6):** AS a developer, I WANT to implement the visual part of the New concept screen IN ORDER TO add a new concept through the Ontology Generator interface.
- **US 5.7 Generate the ontology specification file (4.6.1.7):** AS a developer, I WANT to implement a file specification service IN ORDER TO generate the ontology specification file after any change in the ontology through the Ontology Generator interface.
- **US 5.1 Documentation of the sprint (4.6.1.1):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.5.2.2. In progress

No remaining user stories in progress.

8.5.2.3. Pending

- **US 5.2 Modifications of the API (4.6.1.2):** AS a developer, I WANT to implement the necessary changes IN ORDER TO align with the frontend endpoints.

8.5.3. Products

- **Team - Platform: Developer Manual v1.2:** As a result of Task 3.6.4 Documentation of implementation of API client (4.4.1.6.4) and Task 3.6.5 Documentation of how to use the API client (4.4.1.6.5), the GameHttpClient section of the manual (12.3) has been completed with details of the implementation and descriptions of the steps needed to use the client.

- **Second version of the Ontology Generator interface design:** The result of US 5.3 Modifications of the Ontology Generator interface (4.6.1.3) was to replace the buttons on the Main screen with tabs at the top of the screen (5.4.1).
- **Second version of the Ontology Generator prototype:** Another result of US 5.3 Modifications of the Ontology Generator interface (4.6.1.3) was some modifications on the Ontology Generator prototype as a consequence of the changes made (5.4.2).
- **Third version of the class design of the Ontology Generator:** The OntologyGui class and Listener (5.1.4) interface have been added to the class design to connect the interface to the ontology generation logic and to control communications between GUI components. Also, the FileService (5.1.2) to write the ontology specification file has been defined and implemented as result of US 5.7 Generate the ontology specification file (4.6.1.4).

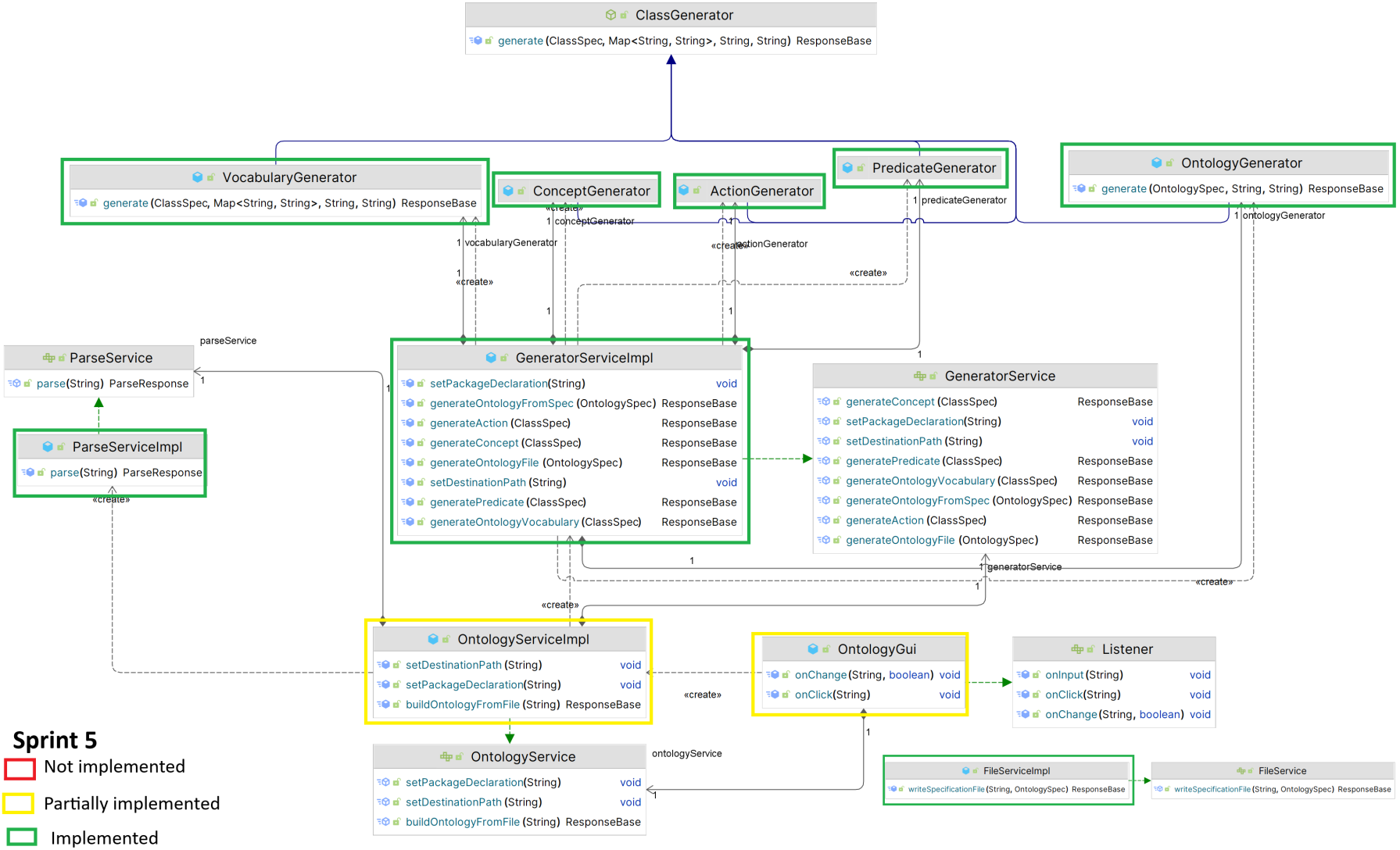


Figure 8.8: Third class design of Ontology Generator

- **Ontology Generator interface with some functionality:** In this sprint, the implementation of the interface for the Ontology Generator has started with the development of the Main screen and the New Concept screen (5.4.4). In addition, the functionality of generating an ontology from a specification file has been implemented as a result of US 1.10 Generate the ontology from a specification file using the interface (4.2.1.10).

8.5.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ First full Team - Platform environment documentation. ▪ Refinement of the Ontology Generator interface design and prototype. ▪ Ontology Generator interface first functional product.	▪ Implementation of the Ontology Generator interface. I had some problems with the positioning of components using the swing library [11]. ▪ Communication to finish the API definition.	▪ Improve the communication to finish the API definition.

8.6. Sprint 6

8.6.1. Sprint burndown chart

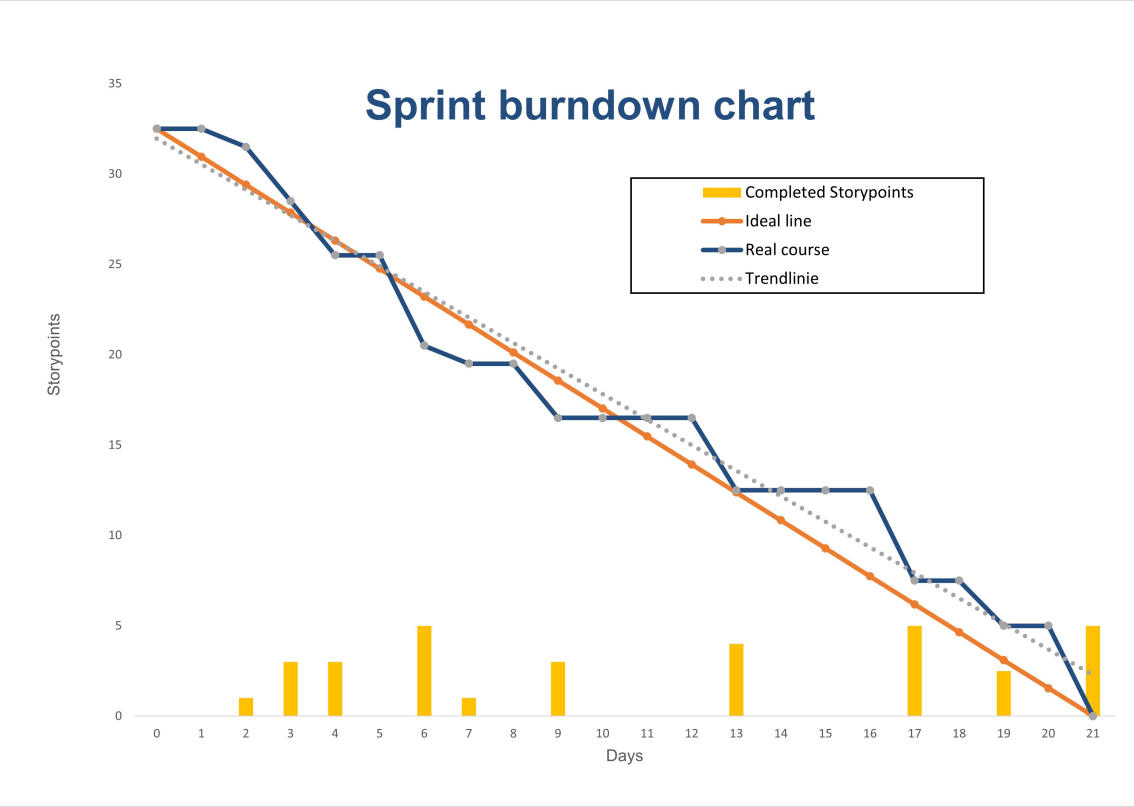


Figure 8.9: Burndown chart Sprint 6

	Completed	In progress	Remaining
Stories	11	0	0
Story points	32,5	0	0

Table 8.6: Table with the state of the work at the end of Sprint 6

As a summary of the work done in Sprint 6, we can consider the following three metrics:

- **Number of epics completed:** 2 (API and Ontology generator)
- **Duration of the sprint:** 21 days
- **Sprint velocity:** 1,54 story points per day

8.6.2. State of the user stories

8.6.2.1. Completed

- **Bug 6.5 Open Ontology Generator interface with double click on JAR file (4.7.1.5):** AS a developer, I WANT to open the Ontology Generator user

interface with a double click on the JAR file IN ORDER TO make the Ontology Generator more accessible to the users.

- **US 5.2 Modifications of the API (4.6.1.2):** AS a developer, I WANT to implement the necessary changes IN ORDER TO align with the frontend endpoints.
- **Bug 6.4 The inputs modify its width depending on its content (4.7.1.4):** AS a developer, I WANT to change the distribution of the Main screen to one column IN ORDER TO avoid the changes in width depending on user input.
- **US 1.11 Generate a concept with the user interface (4.2.1.11):** AS a developer, I WANT to generate a concept from user inputs IN ORDER TO build the ontology from the user interface.
- **US 6.2 Ontology Generator interface visual New Action screen (4.7.1.2):** AS a developer, I WANT to implement the visual part of the New action screen IN ORDER TO add a new action through the Ontology Generator interface.
- **US 1.12 Generate an action with the user interface (4.2.1.12):** AS a developer, I WANT to generate an action from user inputs IN ORDER TO build the ontology from the user interface.
- **US 6.3 Ontology Generator interface visual New Predicate screen (4.7.1.3):** AS a developer, I WANT to implement the visual part of the New predicate screen IN ORDER TO add a new predicate through the Ontology Generator interface.
- **US 1.13 Generate a predicate with the user interface (4.2.1.13):** AS a developer, I WANT to generate a predicate from user inputs IN ORDER TO build the ontology from the user interface.
- **US 6.6 Ontology Generator interface Selector of types in New Concept screen (4.7.1.6):** AS a developer, I WANT to add a selector of types to the New Concept screen IN ORDER TO make it easier to insert attributes.
- **US 1.16 Documentation of the Ontology Generator (4.2.1.16):** AS a developer, I WANT to document the ontology generator IN ORDER TO have a User Manual: Ontology Generator.
  - **Task 1.16.2 Documentation of how to generate an ontology from the user interface (4.2.1.16.2):** Indicate the steps to generate an ontology from the user interface using the Ontology Generator.
- **US 6.1 Documentation of the sprint (4.7.1.1):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.6.2.2. In progress

No remaining user stories in progress.

8.6.2.3. Pending

No remaining user stories pending.

8.6.3. Products

- **Team - Platform: Developer Manual v1.3:** As a result of US 5.2 Modifications of the API (4.6.1.2), the GameHttpClient and Structure of the project sections of the manual (12.3) have been updated with the last changes of the API.
- **Third version of the Ontology Generator interface design:** US 6.4 The inputs modify its width depending on its content (4.7.1.4) and US 6.6 Ontology Generator interface Selector of types in New Concept screen (4.7.1.6) have implied some modifications in the design of the interface screens (5.4.1).
- **Third version of the Ontology Generator prototype:** As a consequence of modifications to the design of the Ontology Generator interface, a new version of the prototype has emerged (5.4.2).
- **Fourth version of the class design of the Ontology Generator:** The OntologyGui class has been connected with the OntologyService service to generate the different items of the ontology. In addition, the OntologyService (5.1.2) has integrated the necessary methods to generate concepts, actions, predicate, the ontology vocabulary file, the ontology file, and the final specification file.

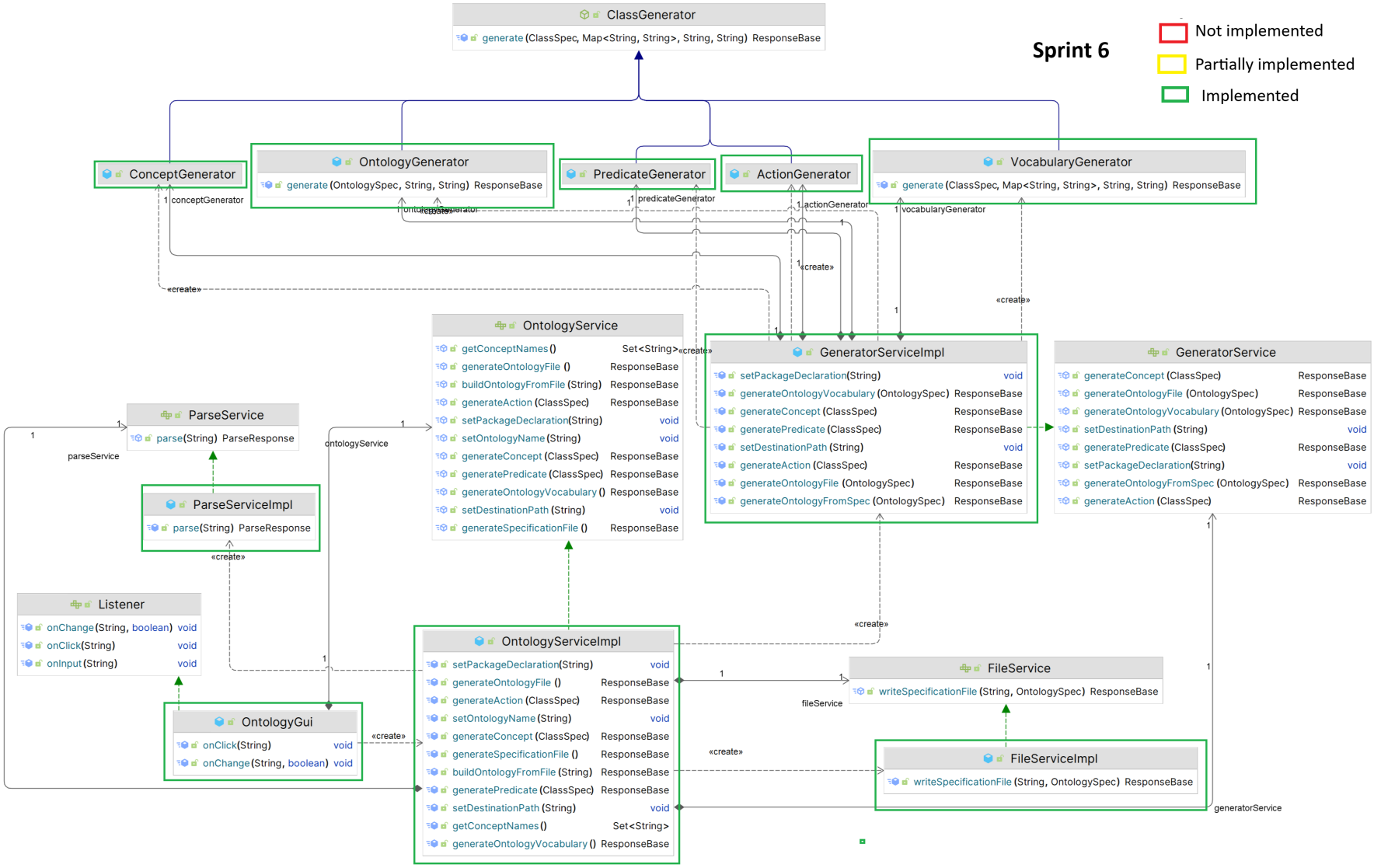


Figure 8.10: Fourth class design of Ontology Generator

- **Ontology Generator interface:** In this sprint, the implementation of the interface for the Ontology Generator has completed with the development of the New Action and New Predicate screens (5.4.4), and the integration with all the generation logic developed in previous sprints.

- **Ontology Generator: User Manual v2.0:** As a result of Task 1.16.2 Documentation of how to generate an ontology from the user interface (4.2.1.16.2), the manual (12.1) has a new subsection in Section Steps to generate an ontology that describes the steps to generate an ontology using the user interface.

8.6.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Improved communication to finish the API epic. ▪ Ontology Generator interface with complete functionality. ▪ Ontology Generator interface design with changes that improve usability (types selector).		

8.7. Sprint 7

8.7.1. Sprint burndown chart

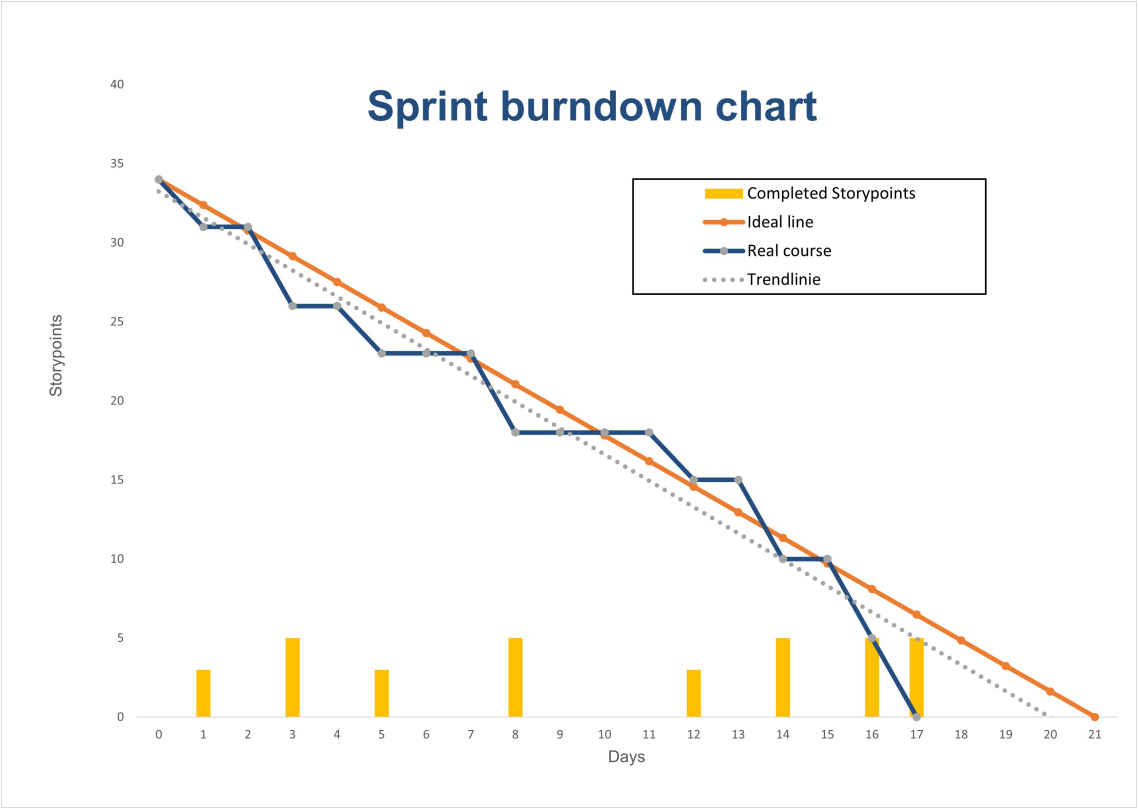


Figure 8.11: Burndown chart Sprint 7

	Completed	In progress	Remaining
Stories	8	0	0
Story points	34	0	0

Table 8.7: Table with the state of the work at the end of Sprint 7

As a summary of the work done in Sprint 7, we can consider the following three metrics:

- **Number of epics completed:** 2 (Log standardization and Documentation)
- **Duration of the sprint:** 21 days
- **Sprint velocity:** 2 story points per day

8.7.2. State of the user stories

8.7.2.1. Completed

- **US 7.2 Analysis of log standardization (4.8.1.2):** AS a developer, I WANT to analyze the possibilities to standardize the logs in the agents project IN ORDER TO implement the best solution.

- **US 7.3 Implementation of log standardization (4.8.1.3):** AS a developer, I WANT to implement the best log standardization solution in the agents project IN ORDER TO recognize each agent and team.
- **US 7.4 Documentation of log standardization (4.8.1.4):** AS a developer, I WANT to document the log standardization implementation IN ORDER TO use it in the agents of the agents project.
- **US 7.5 Introduction of the memory (4.8.1.5):** AS a researcher, I WANT to write the introduction chapter of the final memory IN ORDER TO offer the readers of the document the purpose, scope, and context of the project.
- **US 7.8 Game description section (4.8.1.8):** AS a researcher, I WANT to write a game description chapter in the final memory IN ORDER TO give the readers an overview of the basis of the new game.
- **US 7.6 Global results of the memory (4.8.1.6):** AS a researcher, I WANT to write the global results section of the final memory IN ORDER TO offer the readers the overall outcomes, key findings, or trends derived from the analysis presented.
- **US 7.7 Abstract of the memory (4.8.1.7):** AS a researcher, I WANT to write the Abstract section (Spanish and English version) of the final memory IN ORDER TO offer the readers the main objectives, key findings, and conclusions without reading the entire text.
- **US 7.1 Documentation of the sprint (4.8.1.1):** AS a developer, I WANT to document progress in the current sprint IN ORDER TO have a history of the project.

8.7.2.2. In progress

No remaining user stories in progress.

8.7.2.3. Pending

No remaining user stories pending.

8.7.3. Products

- **Log standardization (7.5):** As a result of the US 7.3 Implementation of log standardization (4.8.1.3), the agents and classes of the Team - Platform environment have a custom logger to print messages in the output terminal of the program.
- **Team - Platform: Developer Manual v1.4:** As a result of US Documentation of log standardization (4.8.1.4), the manual (12.3) has a new chapter that describes the log standardization functionality.
- **Ontology Generator: User Manual v2.1:** The manual (12.1) has incorpored some clarifications about how to modify items of the ontology with the Ontology Generator interface.

- **Agents Game - API v1.2:** The Agents Game - API document (12.2) has been updated with the modifications implemented in the previous sprint (US 5.2 Modifications of the API (4.6.1.2)).
- **Final memory:** With the US 7.5 Introduction of the memory (4.8.1.5), US 7.6 Global results of the memory (4.8.1.6), US 7.7 Abstract of the memory (4.8.1.7) and US 7.8 Game description section (4.8.1.8), this document has been completed with all the necessary information about this master thesis.

8.7.4. Retrospective

What went well?	What did not go as well?	What will we do differently?
▪ Log standardization functionality to help students to debug the errors. ▪ All manuals and specifications updated. ▪ Final memory completed.		

9 Discussion

This section performs a global analysis of the results of all sprints from the SCRUM point of view.

Firstly, similar to what we have done for all sprints, the figure below shows the global burndown chart for the entire project. The estimated story points for this master's thesis was 201,3. In the chart, the orange line represents the ideal burndown trajectory, while the blue line shows the actual work in each sprint. During the initial sprints, the real course was below the ideal line, mainly due to the typical uncertainties and adjustments that occur in the early phases of a project. From Sprint 3 onward, the performance improved notably, and the actual progress began to closely follow the ideal trend. By the final sprint, the chart shows a near-perfect convergence to zero, confirming that the entire planned scope has been successfully delivered. Only the last of the epics enumerated in the Kickoff section (4.1), Testing of the agents, has not been addressed because it requires a depth research in the agents world, and we considered that it is not essential for the objective of this master thesis. Here is a starting point for future improvement of this work.

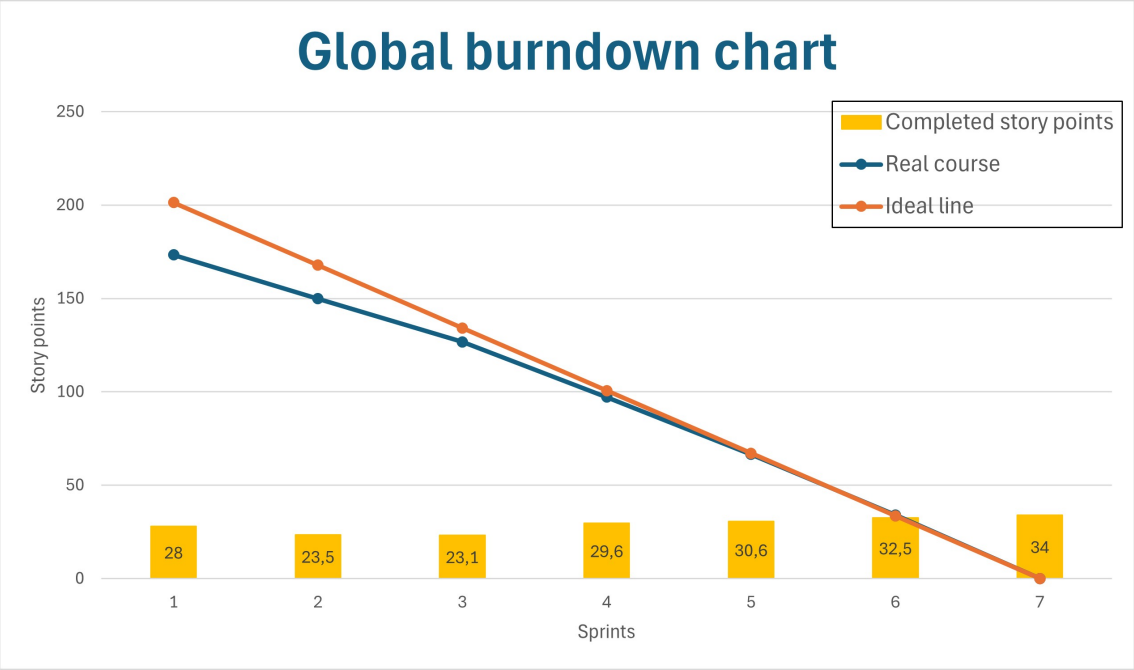


Figure 9.1: Global burndown chart

Secondly, the following figure shows an overall view of the performance in each sprint. In the graph, we can see that during the first sprint, a significant portion of the planned work remained incomplete or in progress due to the initial estimation challenges already discussed in the Sprint 1 retrospective (8.1.4). In Sprint 3, the number of remaining story points had decreased dramatically, indicating better sprint planning. From Sprint 4 onward, nearly all assigned work was completed, with only a minor exception in Sprint 5, where 3 story points remained pending due to the communication issue mentioned in the section 8.5.4.

In the last two sprints, all assigned story points were completed, demonstrating growing efficiency and consistency as the project progressed.

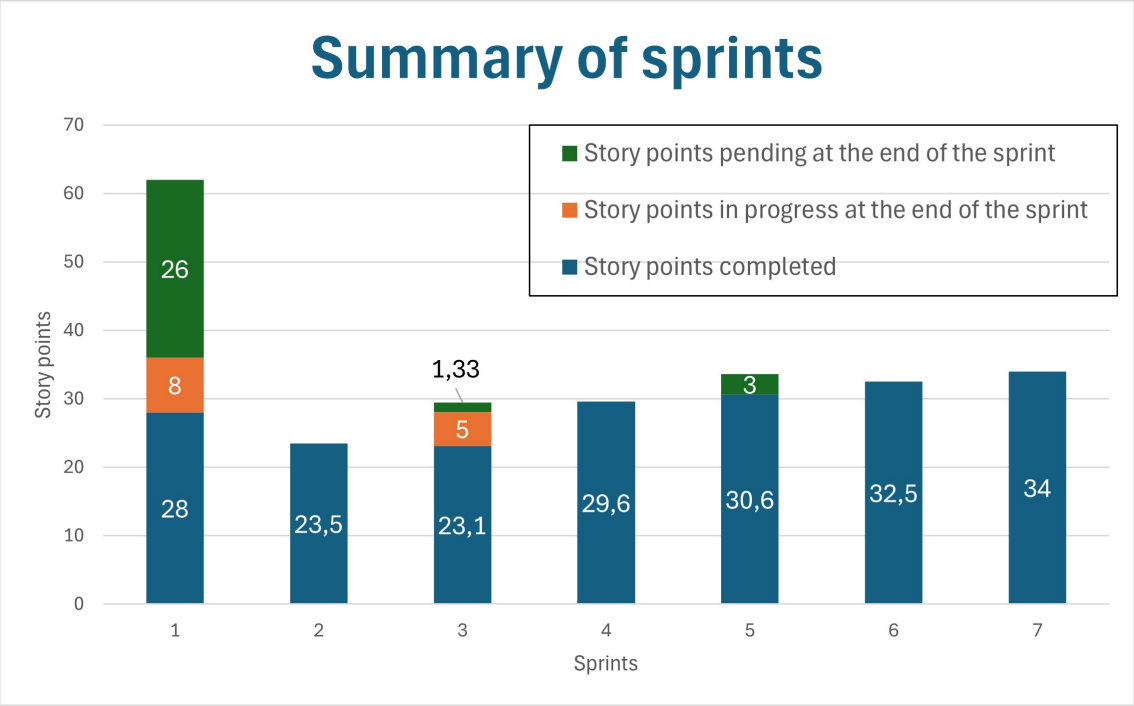


Figure 9.2: Summary of sprints

Finally, the velocity chart in the figure below illustrates the evolution of productivity in terms of story points completed per day. As is typical in most projects, Sprint 1 recorded the lowest velocity (1,55 story points / day) due to the initial adaptation phase. In subsequent sprints, performance improved, reaching the maximum value in Sprint 4 (2,11 story points / day). Overall, I maintained a consistent and predictable pace throughout the development process, with an average velocity of 1,81 story points / day, providing a solid basis for effective planning and workload estimation.

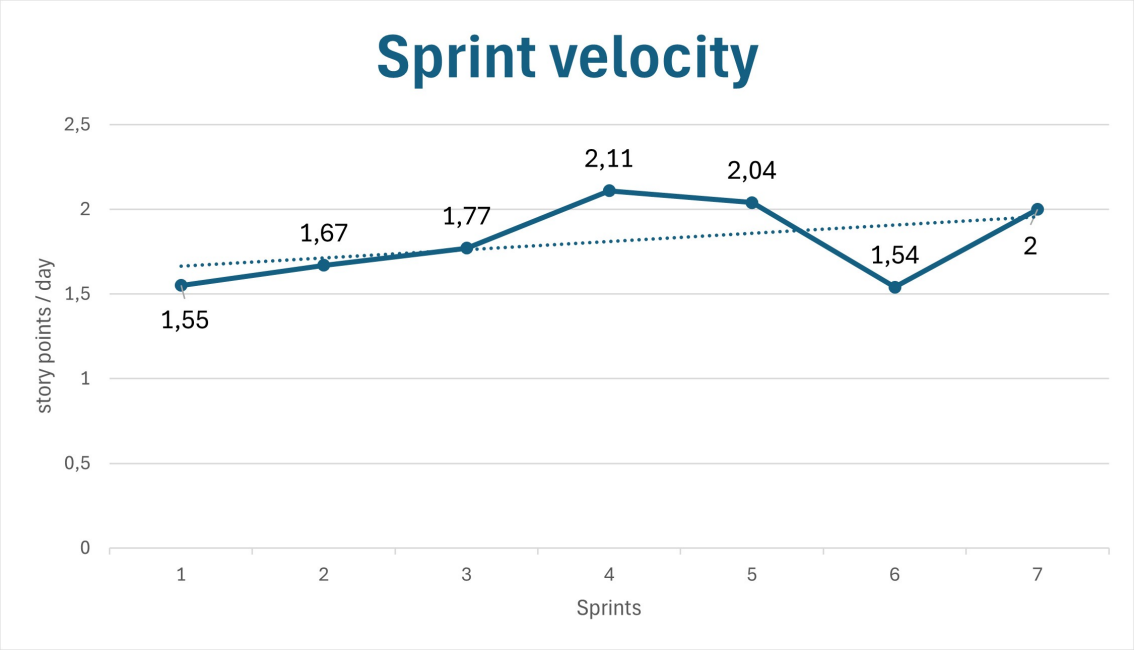


Figure 9.3: Sprint velocity

After the analysis of the three graphs, we can conclude that from the point of view of the SCRUM methodology, this master thesis has obtained great results. The real development course in the last sprints of the project is very close to the ideal one. Moreover, in the end, excellent planning and estimation capabilities have been achieved thanks to the predictable and consistent pace maintained in the development process, solving the problem detected in the first sprint.

10 Conclusions

In the previous section, we have discussed the results of this master thesis from the point of view of the SCRUM methodology. In addition, this section discusses the results from the point of view of objectives and deliverable products.

As we mentioned in the introduction chapter (1), the main objective of this work was to provide the necessary tools and manuals for the students of the Agent-based Software Development course to carry out the practical project. In my opinion, this objective has been achieved, and I am going to argue the reasons in the following paragraphs.

Firstly, with the Ontology Generator implemented (5), the ontology implementation no longer needs to be assigned to a team each sprint. This reduces the time dedicated at the start of the sprint for the implementation of the ontology, and therefore, the team will have more time for agents development tasks. The Ontology Generator is a flexible tool that offers the students two ways to generate the ontology using the design discussed in the Agent-based classes. On the one hand, students can generate the ontology using the Ontology Generator interface that is a user-friendly and efficient user interface where the users can load an ontology specification file or create ontology items one by one interacting with the different screens. On the other hand, the more minimalist students can generate the ontology executing the Ontology Generator in the terminal with an ontology specification file. So, we can consider that with the functionalities provided in the Ontology Generator, we have complied with the first subgoal of this work.

Secondly, the Team - Platform Environment (7) prepared for the backend part contains several functionalities that can simplify the students' work.

Students can use the GameHttpClient (7.4) to communicate with the frontend part of the game. This client performs the necessary calls according to the defined API (12.2) that allows the agents in the game to notify the events and display them in the interface. This GameHttpClient was another of the subgoals of the main objective.

Furthermore, the environment contains a BaseAgent implementation (7.3) with the shared functionality that all agents of the project will use. It has the capacity to send messages to other agents with a custom configuration (performative, protocol, and message content). This functionality allows communication between units of a team and the platform to perform required actions, such as moving or attacking, and the communication of units between them to carry out the prepared strategy. BaseAgent and GameHttpClient have also incorpored a logger functionality (7.5) to allow agents and managers to print different types of messages in the output prompt of the program execution. The logger functionality is a simple but useful capacity for the agents and managers of the project because the integration process between teams is sometimes a chaotic process where errors can arise and the teams need to identify the source and cause of the problems to solve them. For the resolution of those problems, they can use the execution logs, which are differentiated by colors. The BaseAgent and the log standardization are the completion of the last two subgoals of the master thesis.

In summary, the objective of this master thesis has been met by offering students four tools to implement the backend part of the Agent-based Software Development project and the associated manuals with the necessary instructions.

Finally, from a personal point of view, I have improved my knowledge of the agents world and I have learned about new tech tools such as the swing library [11] or the log4j logging framework [14]. In addition, the development of this SCRUM project [1] is a very important experience for my future in the current job environment. In this work, I have managed different types of situations from which I have learned great lessons.

11 Bibliography

[1] What is scrum and how to get started. [Online]. Available: https://www.atlassian.com/agile/scrum

[2] J. R. de Santiago, *New Woa Interface, Adapting and Improving an Old Agents Practice using Unity 3D*, 2025.

[3] Jade site | java agent development framework. [Online]. Available: https://jade.tilab.com/

[4] Jade a white paper. [Online]. Available: https://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf

[5] Empowering collective intelligence: The use of multiagent systems in jade. [Online]. Available: https://medium.com/%40nachomnl/empowering-collective-intelligence-the-use-of-multiagent-systems-in-jade-ecc825bbf8bc

[6] Comparison of multi-agent platform usability for industrial-grade applications. [Online]. Available: https://www.mdpi.com/2076-3417/14/22/10124

[7] Git. [Online]. Available: https://git-scm.com/

[8] Github. [Online]. Available: https://github.com/

[9] Java | orable. [Online]. Available: https://www.java.com/en/

[10] Figma: The collaborative interface design tool. [Online]. Available: https://www.figma.com/

[11] javax.swing (java platform se 7). [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

[12] google/gson: A java serialization/deserialization library to convert java objects into json and back. [Online]. Available: https://github.com/google/gson

[13] Wiremock - flexible, open source api mocking | wiremock. [Online]. Available: https://wiremock.org/

[14] Apache log4j :: Apache log4j. [Online]. Available: https://logging.apache.org/log4j/2.x/index.html

12 Annexes

12.1. Ontology Generator: User Manual

Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

European Master in Software Engineering

Master Thesis

Ontology Generator: User Manual

Author: Adrián Sánchez Rodero

Version control

Version	Date	Change
v1.0	03/01/2025	First document.
v1.1	25/01/2025	Add ontology name in the specification file.
v1.2	27/01/2025	Add symbol - to indicate class names.
v1.3	15/02/2025	Delete unnecessary restriction on the specification file name.
v1.4	21/02/2025	Add Run Ontology Generator application chapter.
v1.5	28/02/2025	Add description of each response code in Run Ontology Generator application chapter.
v2.0	28/04/2025	Add to the documentation the user interface functionality.
v2.1	08/05/2025	Clarify how to modify items in the Ontology Generator interface.

Table of Contents

1 Specification file

This chapter explains how to write the ontology specification file to allow the software to correctly generate the ontology.

The user must provide a text file following the structure and rules described in the next sections.

1.1. Structure of the file

This section specifies the requirements that the text file must comply for the software to work correctly.

The structure of the file must be the following:

```
OntologyName
concepts
    -ConceptName1
        attributeName1: TypeAttribute1
        attributeName2: TypeAttribute2

actions
    -ActionName1
    -ActionName2
        attributeName1: TypeAttribute1

predicates
    -PredicateName1
        attributeName1: TypeAttribute1
        attributeName2: TypeAttribute2
```

The file must start with the name of the ontology (**OntologyName**) and must have three reserved words that indicate the three different groups of classes (described in detail in chapter 2):

- **concepts**: In this group must be specified the concepts of the ontology.
- **actions**: In this group must be specified the actions of the ontology.
- **predicates**: In this group must be specified the predicates of the ontology.

Each group can be empty or can contain one or more class specifications. Each class specification must follow the structure described in the class specification subsection 1.1.1.

1.1.1. Class specification

A class specification is the scheme to define the properties of a class (name and attributes). It must have the following structure:

```
-Name
    attributeName1: TypeAttribute1
    attributeName2: TypeAttribute2
    ...
```

Where:

- **Name** is the name of the class.
- **attributeName**$_i$ is the name of the attribute i of the class.
- **TypeAttribute**$_i$ is the type of the attribute i of the class.

To define a class without attributes, it is only necessary to specify the name of the class.

Examples of class specifications:

- Car class with two attributes: model of type String and price of type double.

```
    -Car
        model: String
        price: double
```

- Utils class with no attributes.

```
    -Utils
```

1.2. Rules

This section enumerates the rules that must be followed for the correct processing of the file with the ontology specification.

1. The **name of the ontology** must be in the first line of the file and it can not be omitted.
2. The three **reserved words** (concepts, actions and predicates) must appear in the file whether or not they contain specifications.
3. If a group does not contain any class specification, leave it empty.
4. The different groups of class specifications defined by the reserved words (concepts, actions and predicates) must be separated by a blank line.
5. The **class specifications** inside each group must follow the structure described in subsection 1.1.1:
   5.1. The class specification must start with the symbol - following by the name of the class.
   5.2. All attributes of a class specification must have a name and a type.
   5.3. The names and types of the attributes of a class specification must be separated by the symbol :.
   5.4. If a class does not contain any attribute, indicate only the name of the class.
6. If a concept (X) has any attribute whose type is another concept (Y), the concept Y **must be declared before** the concept X.

2 Ontology classes

This chapter describes the different ontology classes generated by the Ontology Generator.

The Ontology Generator generates five different types of classes (concepts, actions, predicates, ontology vocabulary and ontology), but the user only need to specify three: concepts, actions, and predicates. The following sections explain how to specify them in the specification file.

2.1. Concepts

The concepts of the ontology are the classes that implement the Concept interface of the JADE library.

The specifications of the concept classes of the ontology must be included in the **concepts group** described in the section 1.1 and following the rules of section 1.2.

Good example of concepts group with three concepts:

```
    concepts
        -Building
            type: String
        -Coordinate
            xValue: int
            yValue: int
        -Cell
            coord: Coordinate
            content: String
```

Bad example of concepts group:

```
    concepts
        -Cell
            coord: Coordinate
            content: String
        -Building
            type: String
        -Coordinate
            xValue: int
            yValue: int
```

The above example is not applying rule 5 of section 1.2.

2.2. Actions

The actions of the ontology are the classes that implement the AgentAction interface of the JADE library.

The specifications of the action classes of the ontology must be included in the **actions group** described in section 1.1 and following the rules in section 1.2.

Example of actions group with two actions:

```
    actions
        -ConstructBuilding
            building: Building
        -DistributeMap
            mapSize: Coordinate
```

2.3. Predicates

The predicates of the ontology are the classes that implement the Predicate interface of the JADE library.

The specifications of the predicate classes of the ontology must be included in the **predicates group** described in section 1.1 and following the rules in section 1.2.

Example of predicates group with one predicate:

```
    predicates
        -Proposes
            estimation: Estimation
```

3 Example of the specification file

This chapter contains a complete example of an ontology described using the specification file explained in this document.

```
WoAOntology
concepts
    -Building
        type: String
    -Coordinate
        xValue: int
        yValue: int
    -Cell
        coord: Coordinate
        content: String
    -Resource
        typeRes: String
        amount: float
    -CurrentResources
        gold: Resource
        stone: Resource
        wood: Resource
    -Destination
        newDest: Coordinate
    -Direction
        dir: int
    -NewPhase
        phase: int
    -StorageCapacity
        size: float

actions
    -AllocateUnit
        initialPosition: Coordinate
    -AssignNewUnit
        unitID: AID
    -ChangePhase
    -CollectResource
    -ConstructBuilding
        building: Building
    -CreateUnit
    -DistributeMap
        mapSize: Coordinate
    -InformCurrentResources
        tribeResources: CurrentResources
        storage: StorageCapacity
    -InformNewBuilding
        cell: Cell
    -Move
        dest: Direction
    -Register
    -RevealCell
        cellContent: Cell

predicates
```

4 Run Ontology Generator application

This chapter describes the information needed to generate an ontology.

4.1. Dependencies

This section indicates the software that is required to run the Ontology Generator.

- **JAVA**: To be able to run the Ontology Generator application, you will need JAVA installed on your computer. Any version higher than 21.0.2.

4.2. Program arguments

The Ontology Generator application has four arguments.

```
java -jar ontology_generator.jar [option] [specificationFile]
 [packageDeclaration] [destinationPath]
```

where:

- **option**: Argument to indicate the mode to run the generator. It can have six values:
  - -h and --help: Displays the application help text.
  - -d and --debug: Run the program in debug mode (only for Ontology Generator developers).
  - -f and --file: Run the program to generate an ontology in the **destinationPath**, with the content specified in the **specificationFile** and following the **packageDeclaration** structure.
  - -g and --gui: Run the Ontology Generator user interface.
- **specificationFile**: Path of the ontology specification file to process. It is required for the -f and --file options.
- **packageDeclaration**: Common package declaration for all ontology classes. It is required for the -f and --file options.
- **destinationPath**: Path to save the ontology files. It is required for the -f and --file options.

4.3. Steps to generate an ontology

4.3.1. Generate an ontology with the console

In order to generate an ontology structure using the console, it is necessary to follow the next points:

1. Write the **ontology specification file** following the guidelines described in this document → ontologySpecificationFile.txt
2. Save the file in a known directory → ./ontologySpecificationFile.txt
3. Define the **package structure** that you want the ontology to have → org.ontology
4. Choose a **destination path** to save the ontology files → ./TestOntology
5. Open a terminal in the folder where you have the Ontology Generator jar (**ontology_generator.jar**) and run the following command:

```
java -jar ontology_generator.jar --file
 ./ontologySpecificationFile.txt org.ontology ./TestOntology
```

6. Check that ./TestOntology contains an ontology structure with at most three folders (concepts, actions and predicates) and two java files (OntologyNameVocabulary and OntologyName).

4.3.2. Generate an ontology with the user interface

In order to load the Ontology Generator interface and create an ontology iteractively, it is necessary to perform the following operations:

1. Run the Ontology Generator interface. This step can be carried out in two ways.
   a. Open a terminal in the folder you have the Ontology Generator jar (**ontology_generator.jar**) and run the following command:

```
java -jar ontology_generator.jar --gui
```

   b. Double-clicking on the Ontology Generator jar. In this case, you should have configured on your computer the application to open jar files. If you do not have configured, you have to open the jar with the **javaw application** inside your Java bin folder (C:\Program Files\Java\jdk-21\bin\javaw.jar, in my case). You can configure your computer to open jar files always with that application. If you want to see the **logs of the program**, you can open the jar using the **java application** in the same folder as the javaw application.
2. If you have to modify or add items to an **ontology previously created**, you should have to load the ontology specification file from the ontology before performing any operation (if not, you can skip this step). For this, on the

main screen you should specify the path of the specification file (./specificationFile.txt), the package declaration (org.ontology), and the destination path (./ontology) and click the "Generate all from specification file" button.

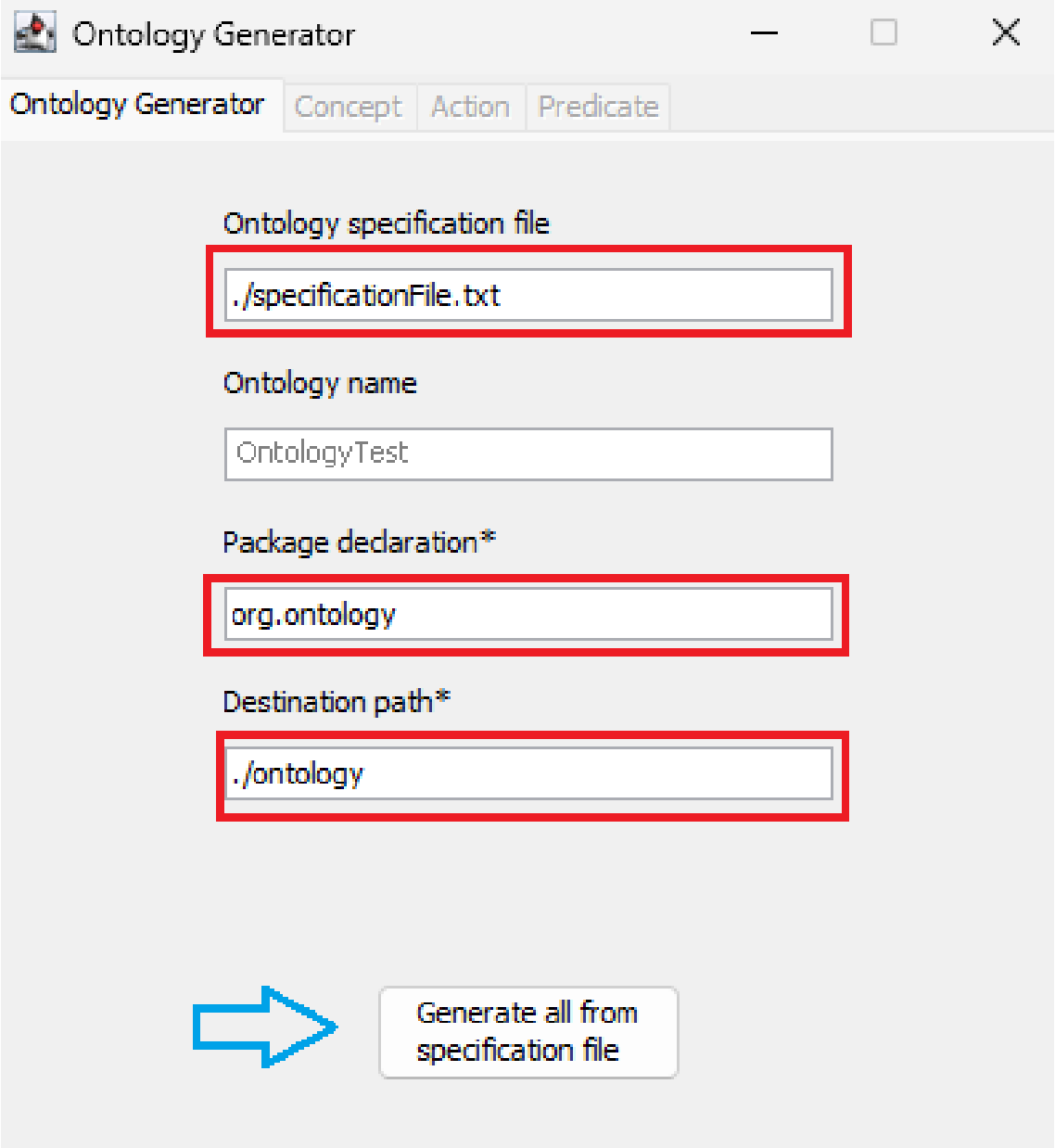


Figure 4.1: Ontology Generator interface - Load ontology specification file

If the ontology has been loaded correctly, a success banner appears.

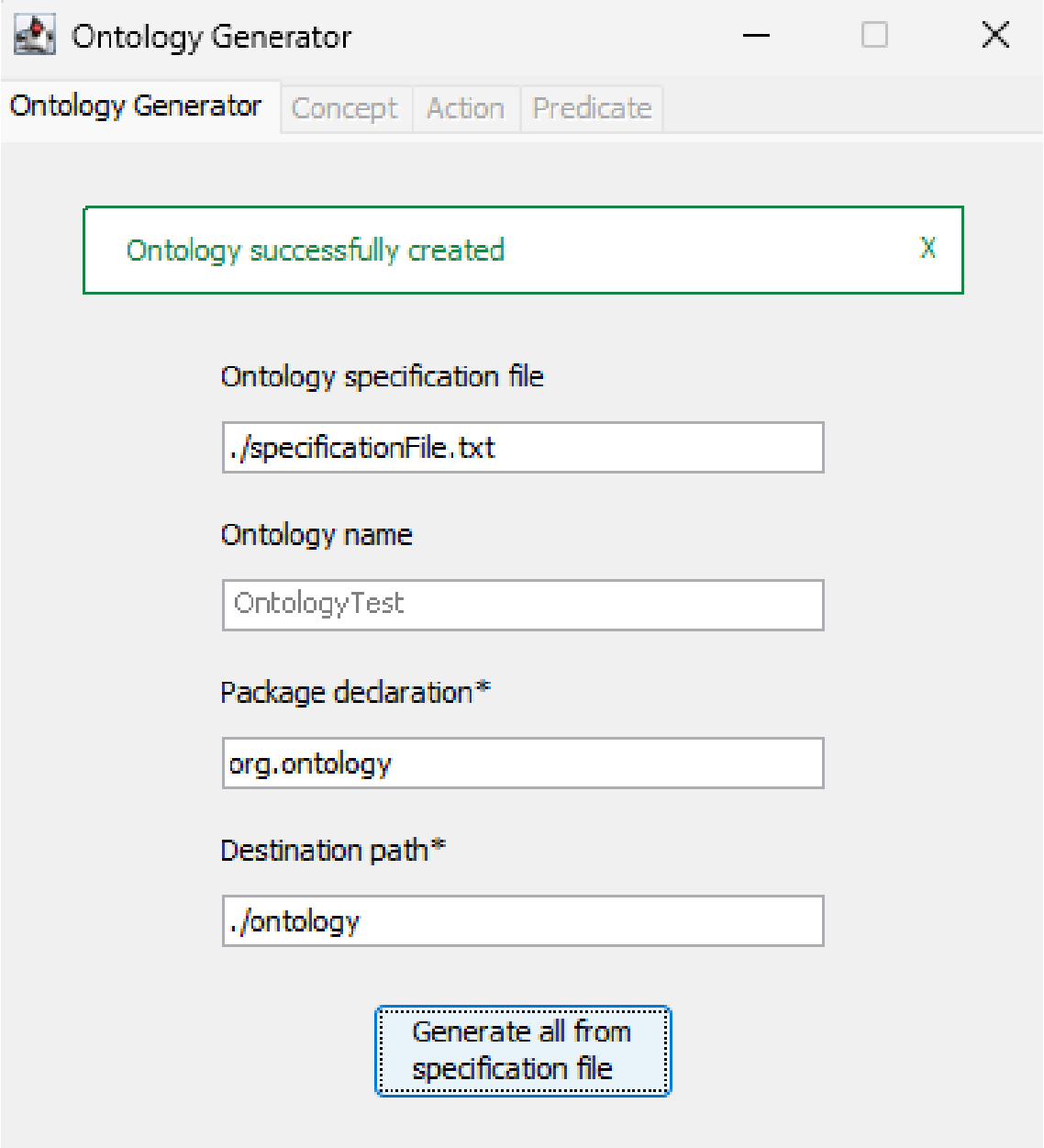


Figure 4.2: Ontology Generator interface - Success ontology specification file

If an error occurs during the operation, the screen indicates the error with a descriptive message.

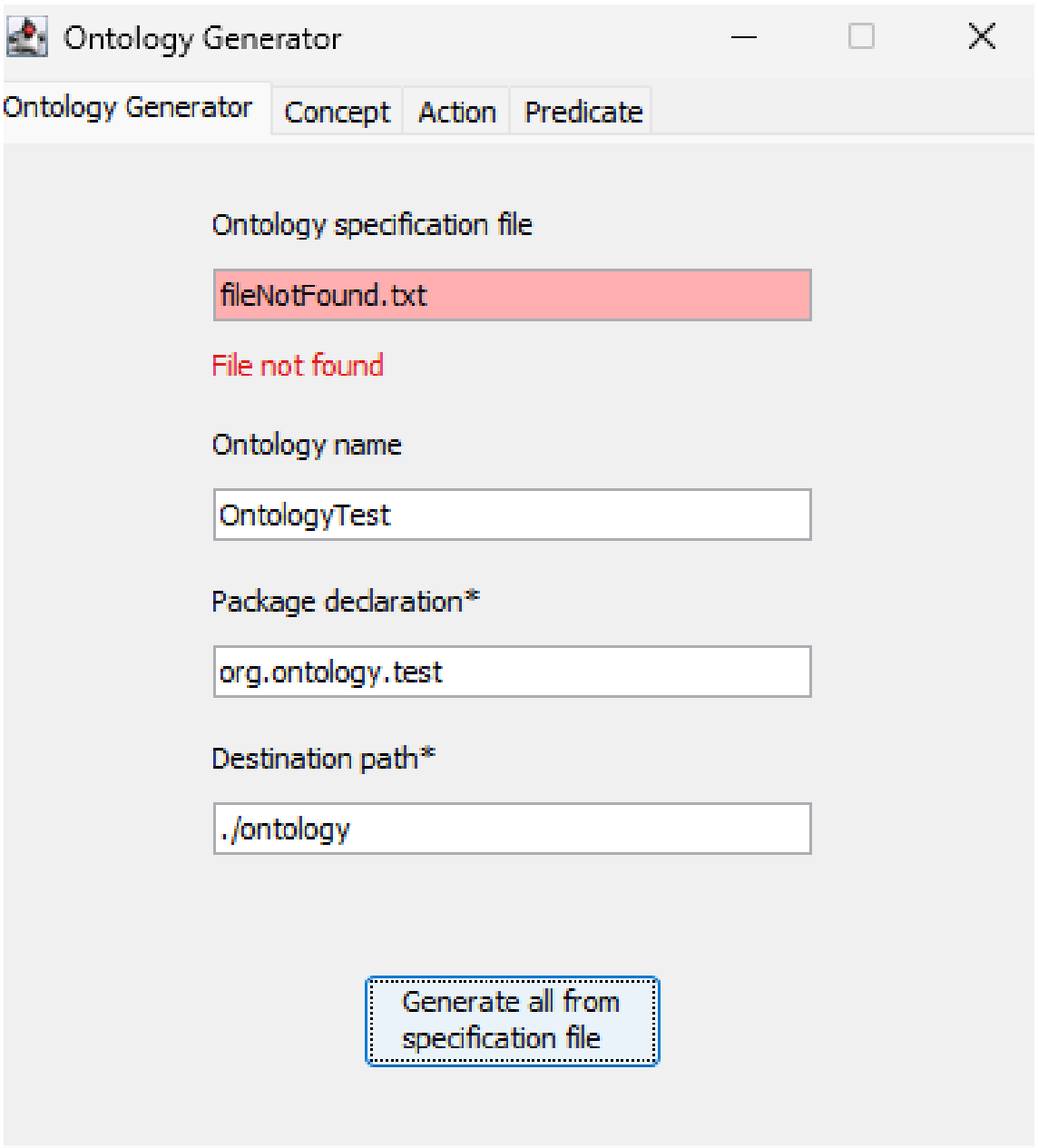


Figure 4.3: Ontology Generator interface - File not found error loading specification file

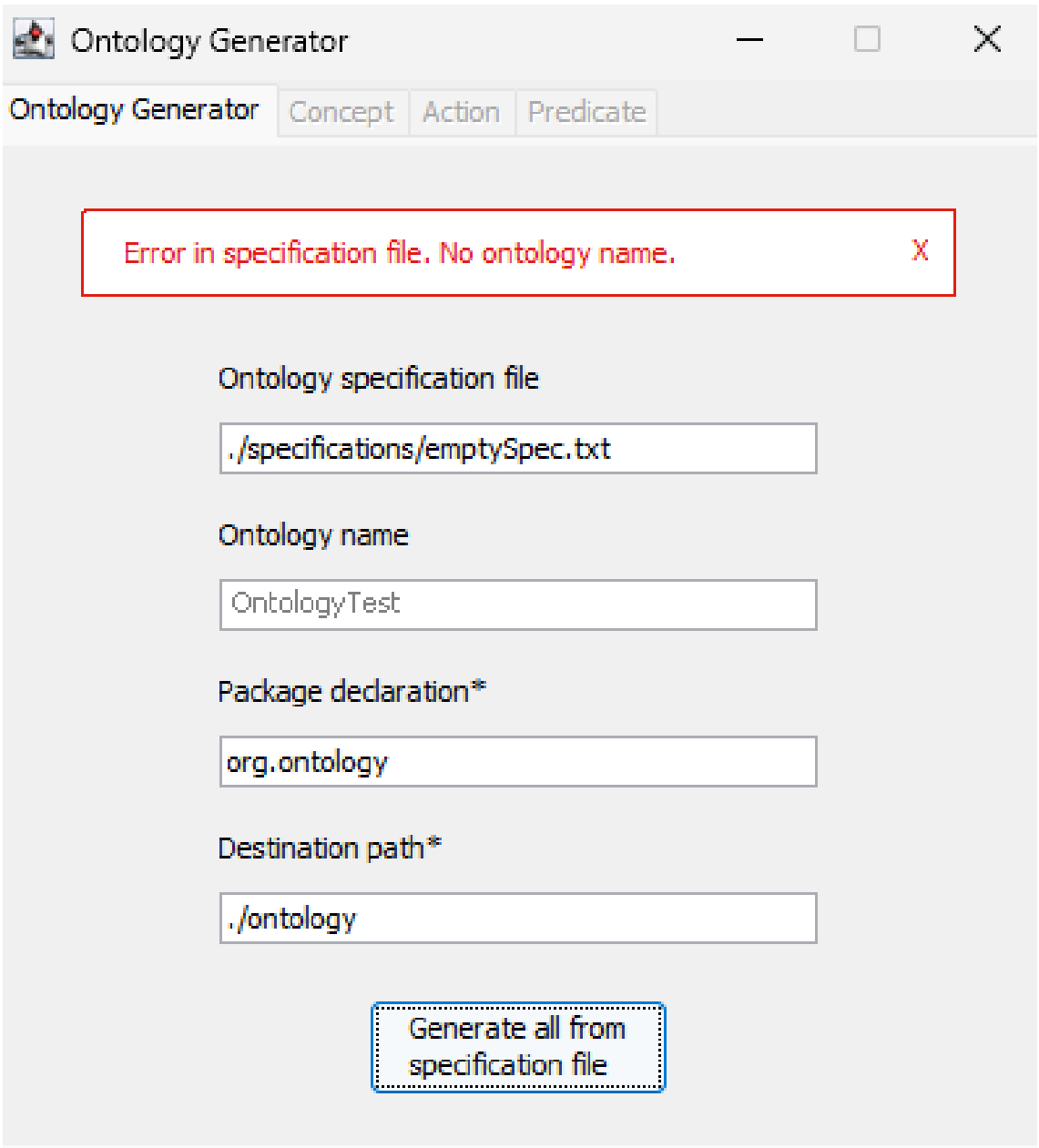


Figure 4.4: Ontology Generator interface - Error loading specification file

3. If you have skipped the previous step (you are creating a **new ontology**), you should specify the **package declaration** and the **destination path** on the main screen.

4. Specify the **ontology name** on the main screen. At this point, the Concept, Action and Predicate tabs become enabled.

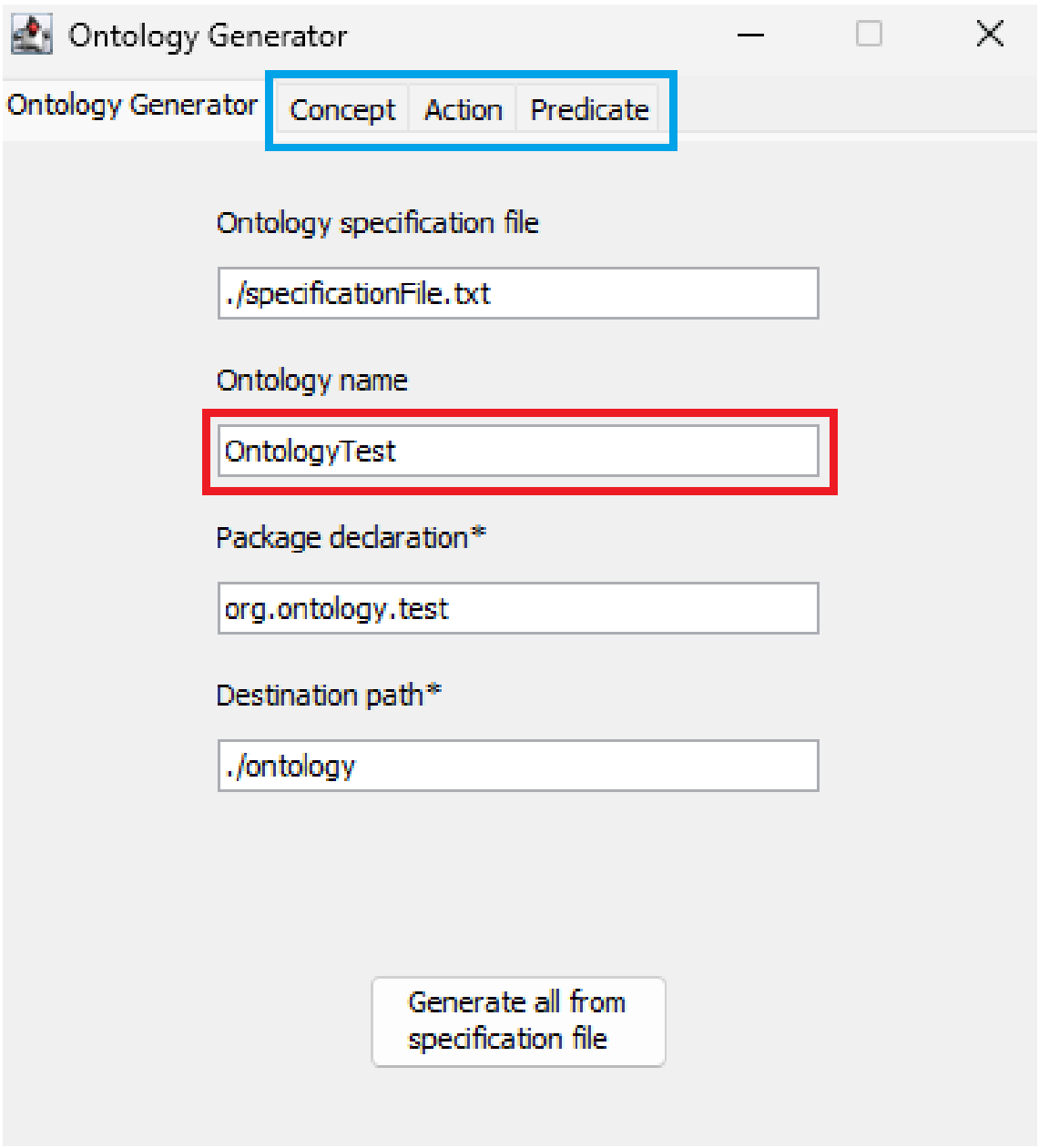


Figure 4.5: Ontology Generator interface - Concept, Action and Predicate tabs enabled

5. **Modify items** already created in the ontology.

   a. For **concepts**, you have to take into account rule 6 of the ontology specification file (Rule 6). In terms of the user interface, this means that you have to check the concepts group in the ontology specification file to not use a concept type declared after the concept you want to modify because concepts preserve the generation order in the ontology specification file. If you want to add that dependency, you must delete all the concepts created after and add before the used concepts in the ontology specification file before you load it.

   b. For **actions** and **predicates**, you only have to use the same name as the already created item and define the item again with the new attributes.

6. **Add items** (concepts, actions and predicates) to the ontology.

    a. To add a **new concept**, you should go to the Concept tab, enter at least the required data (concept name), and click on the "Generate" button. To add attributes to the concept, you should insert the attribute name and select a type on the selector list.

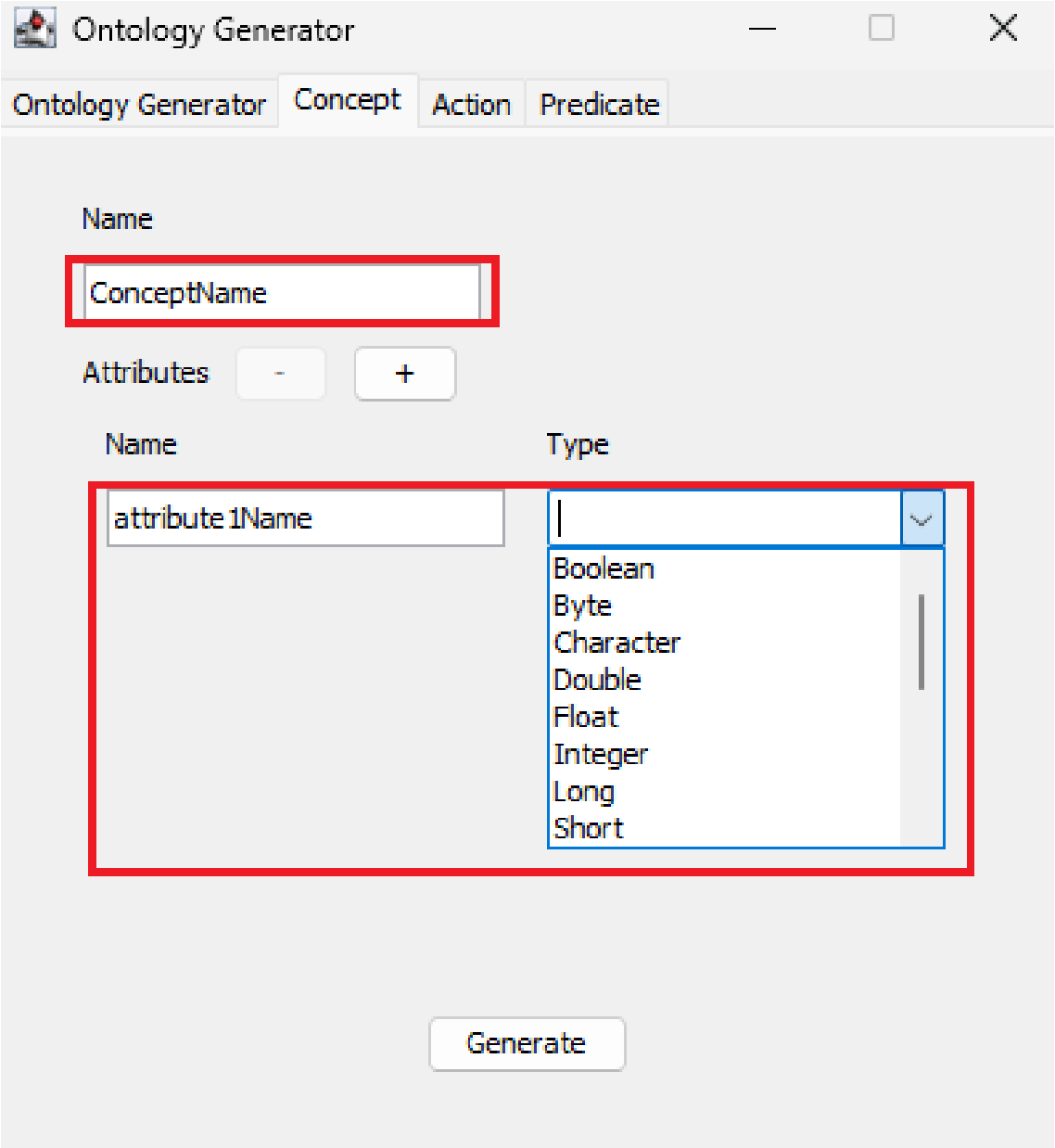


Figure 4.6: Ontology Generator interface - New concept

The attribute type selector has search functionality, so you can insert letters of the type, and the list of types is getting smaller.

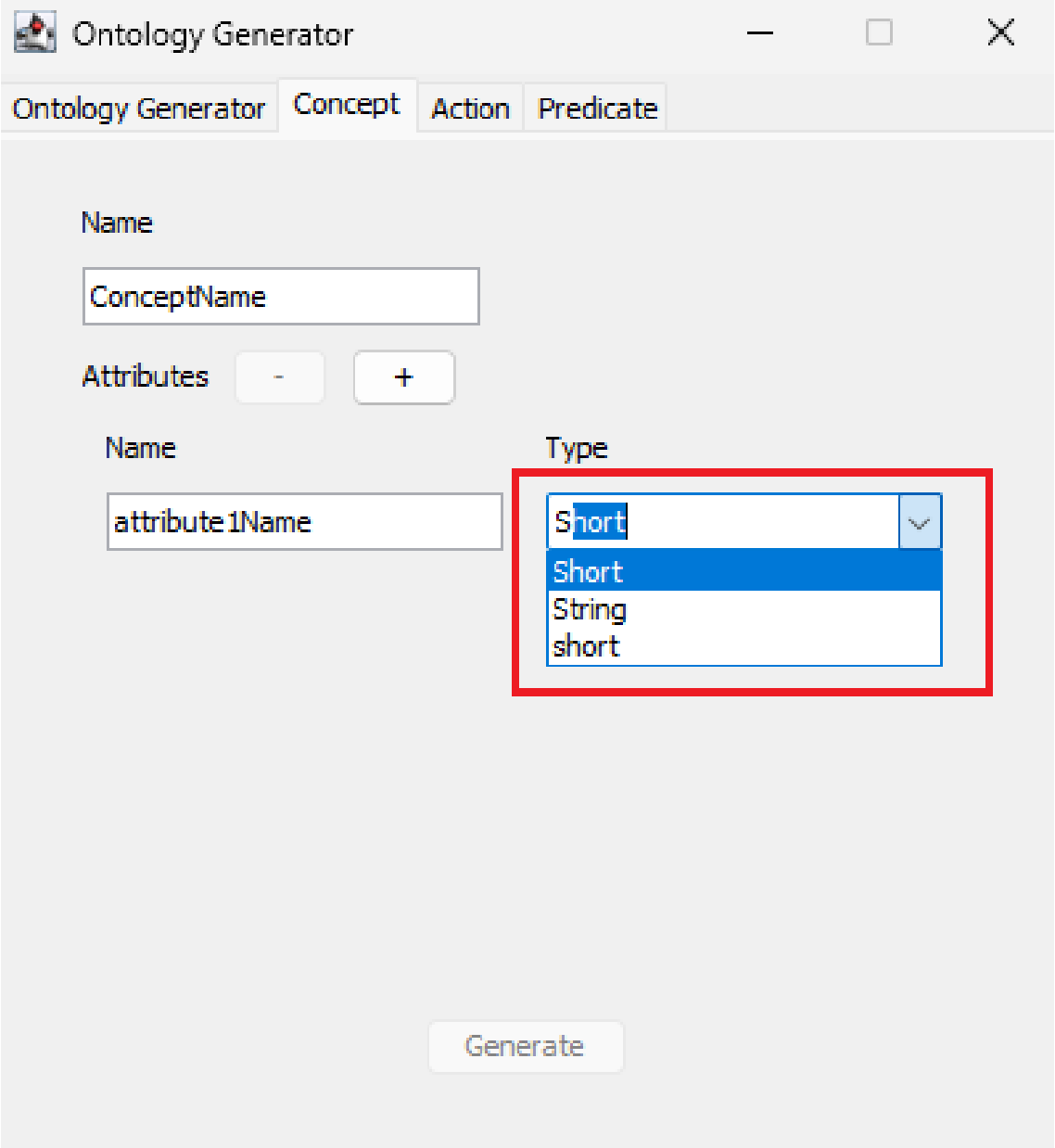


Figure 4.7: Ontology Generator interface - New concept search functionality

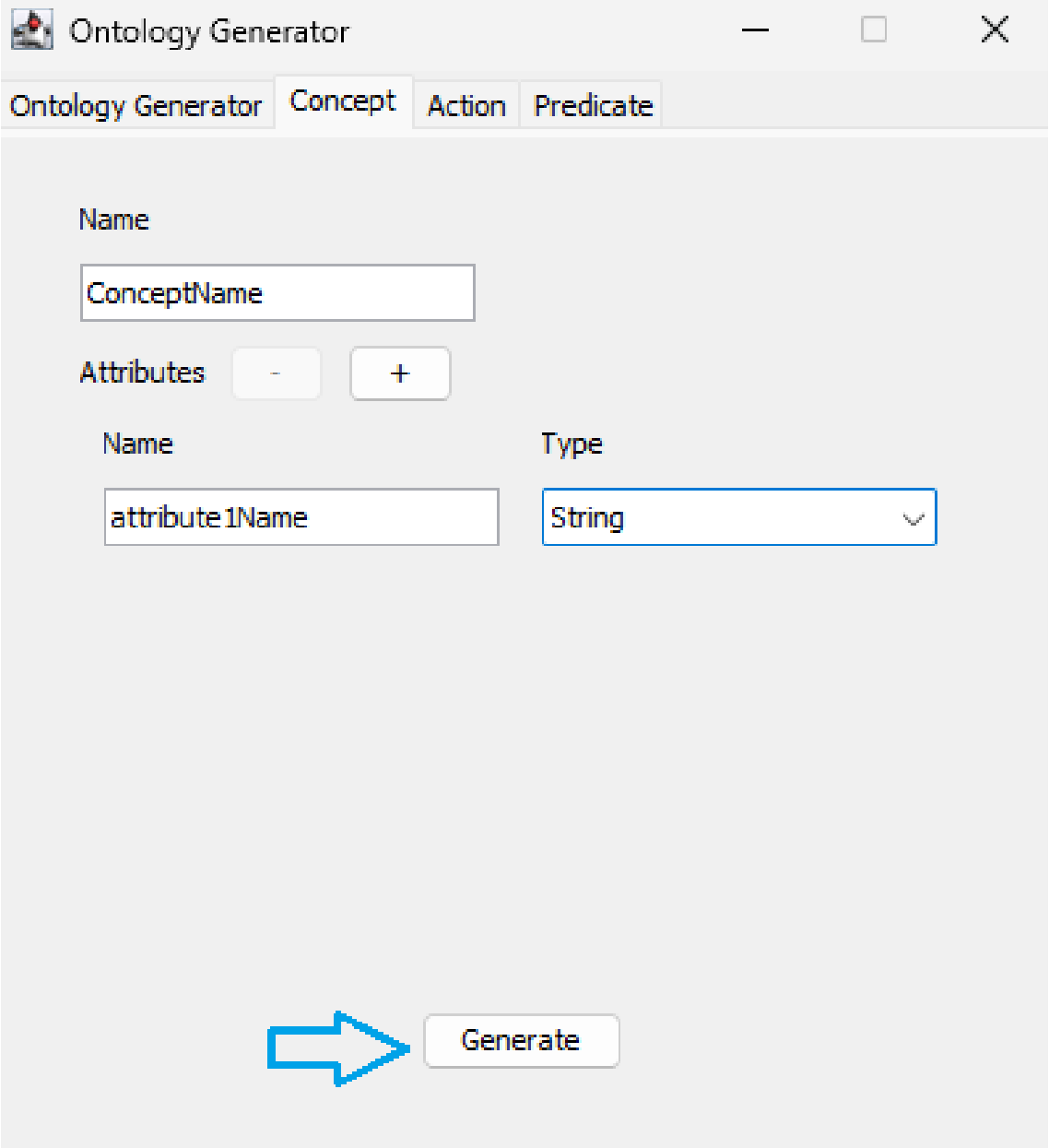


Figure 4.8: Ontology Generator interface - New Concept generate enabled

If the concept has been created correctly, a success banner appears.

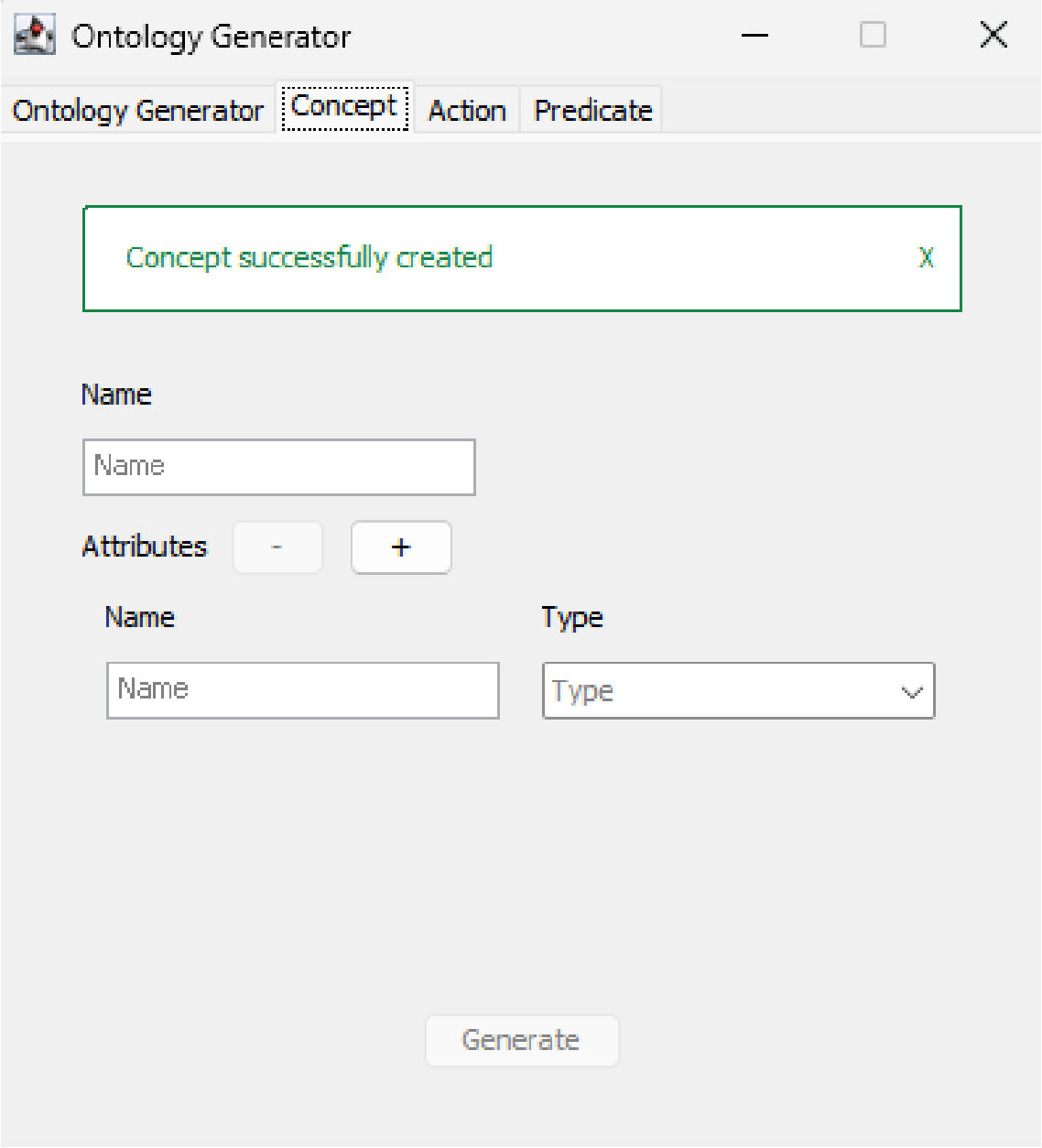


Figure 4.9: Ontology Generator interface - Success new concept

b. To add a **new action**, you should go to the Action tab and follow the same steps as in the concept case. If the action has been created correctly, a success banner appears.

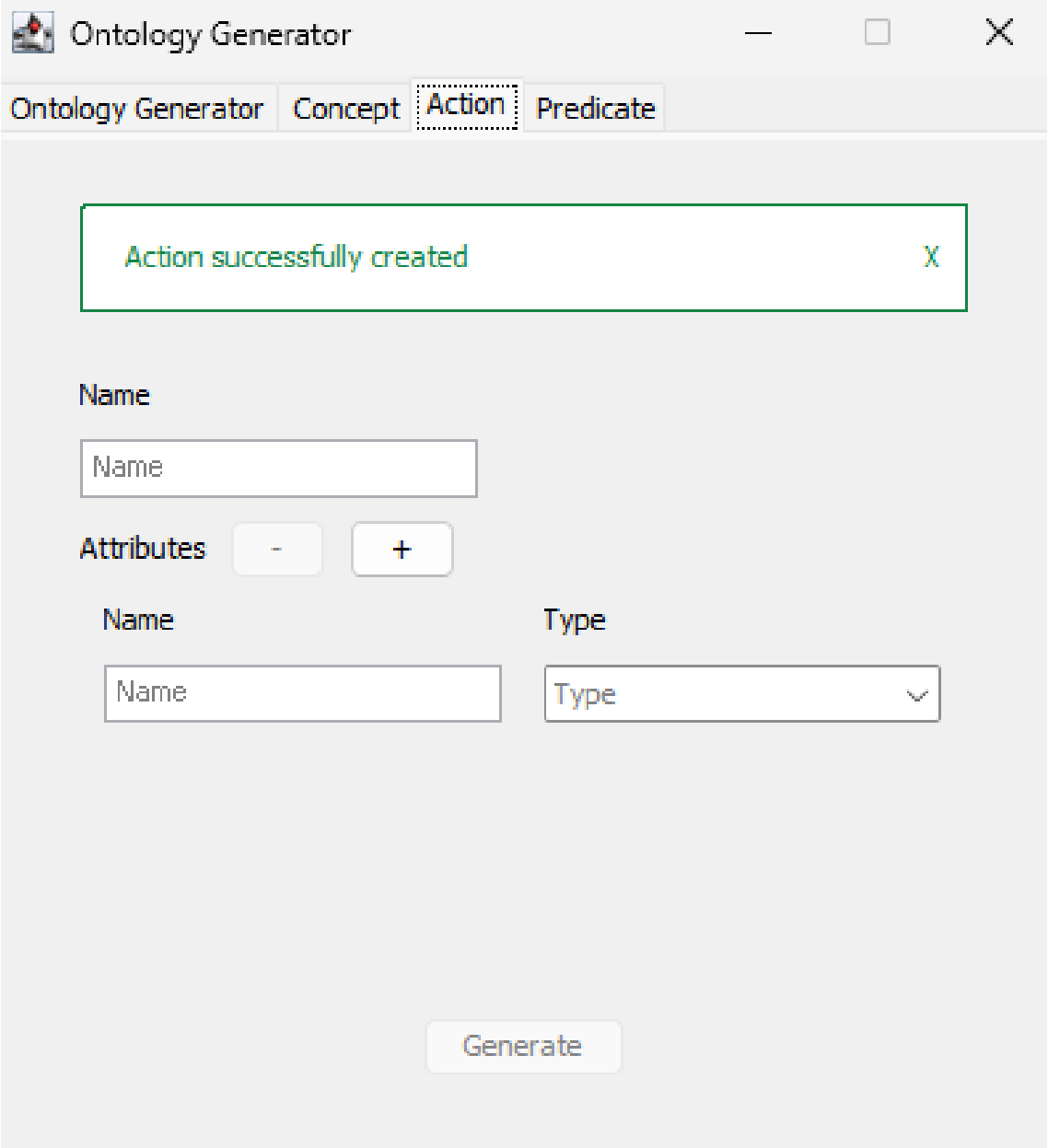


Figure 4.10: Ontology Generator interface - Success new action

c. To add a **new predicate**, you should go to the Predicate tab and follow the same steps as in the previous cases. If the predicate has been created correctly, a success banner appears.

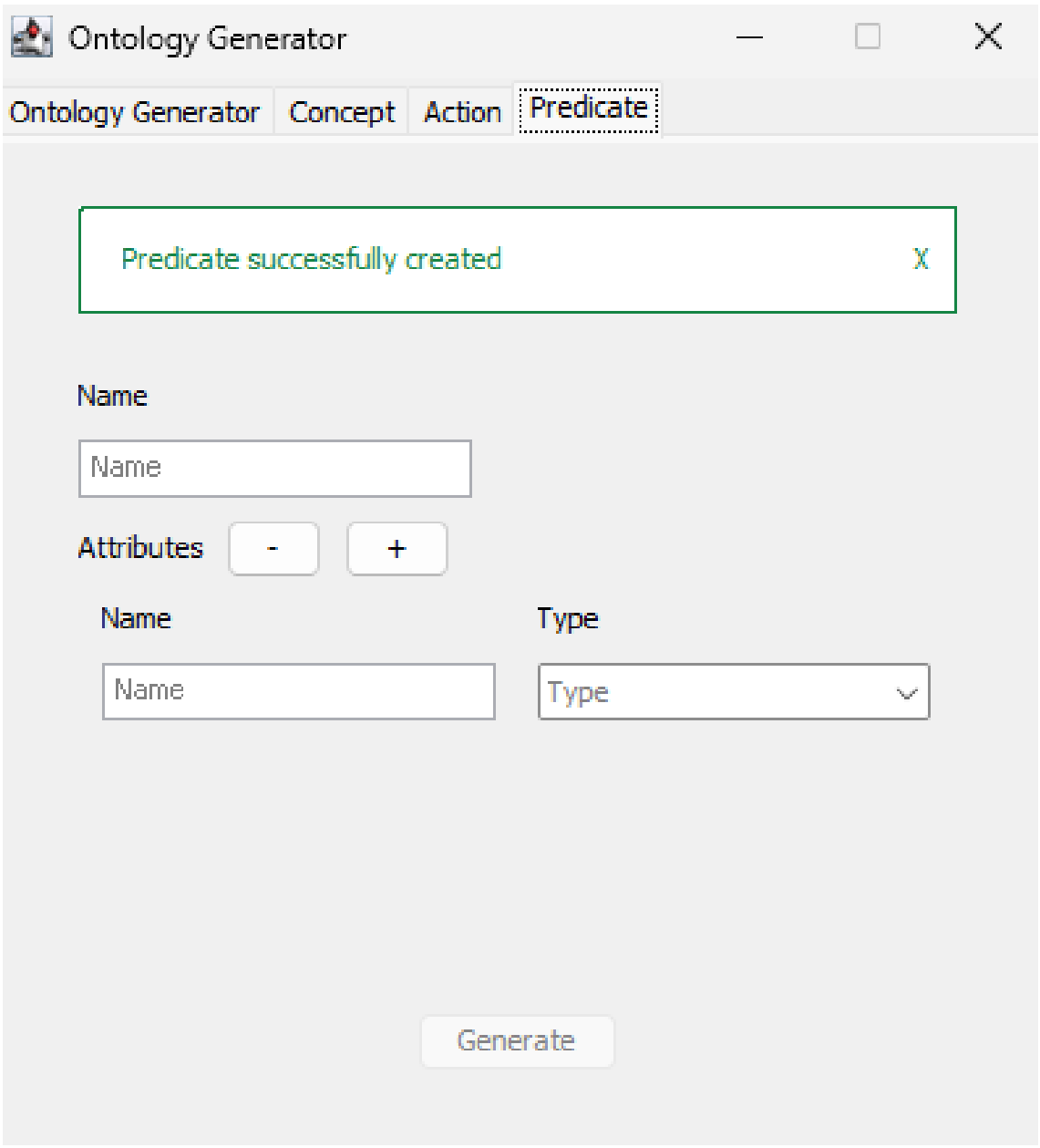


Figure 4.11: Ontology Generator interface - Success new predicate

7. In case of **any error** in the previous step, an error banner appears on the corresponding screen with a descriptive message.

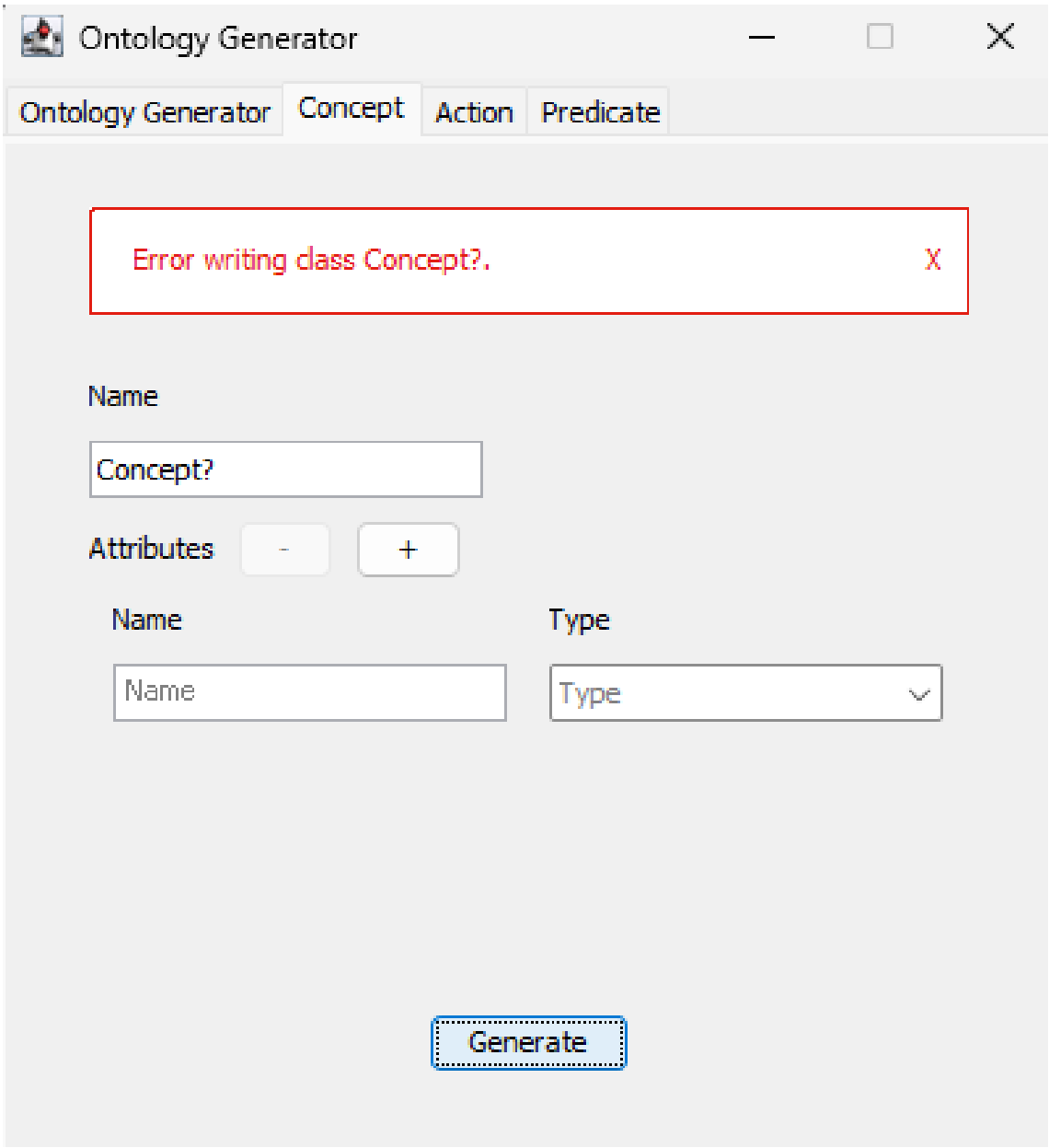


Figure 4.12: Ontology Generator interface - Error new concept

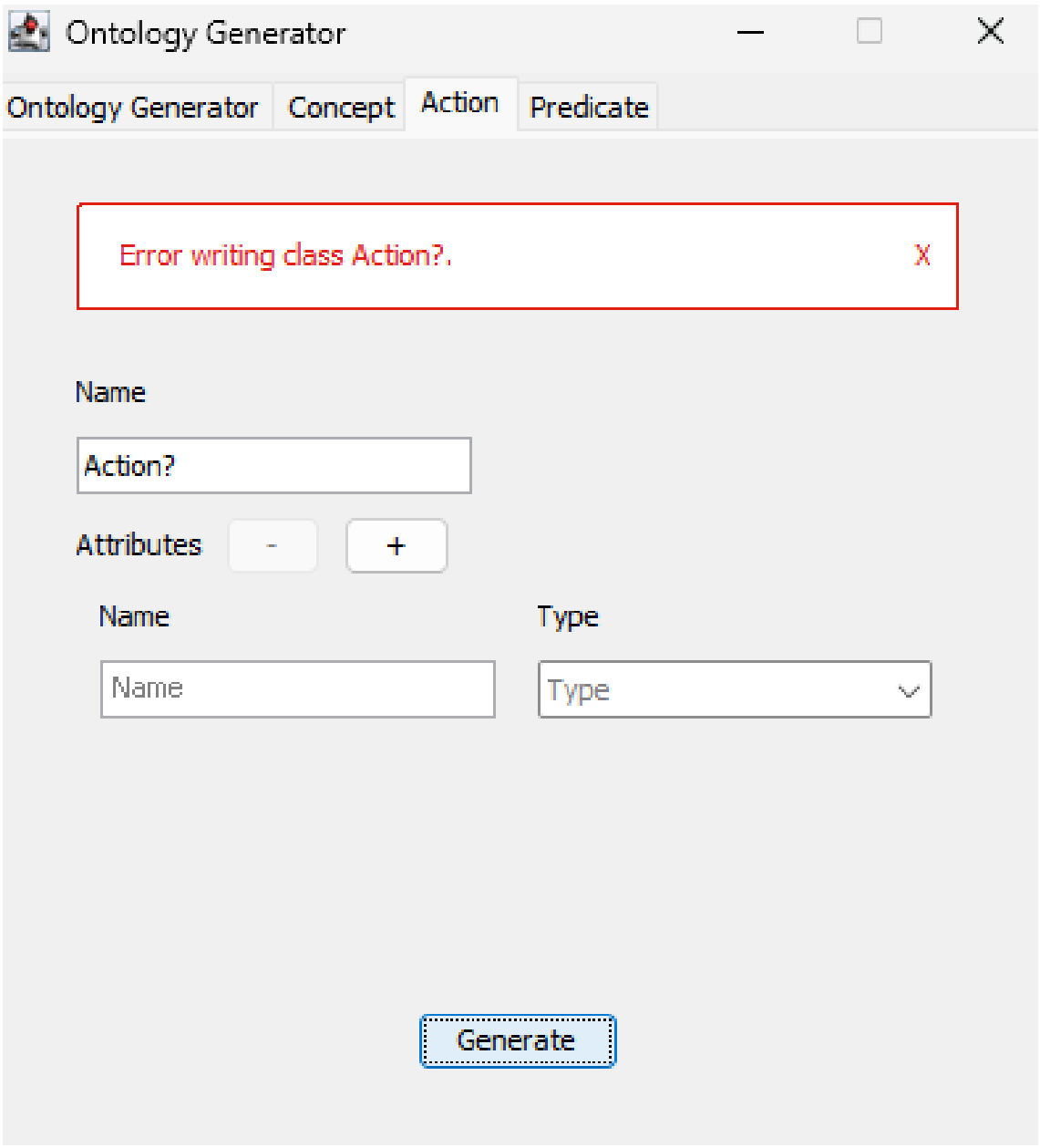


Figure 4.13: Ontology Generator interface - Error new action

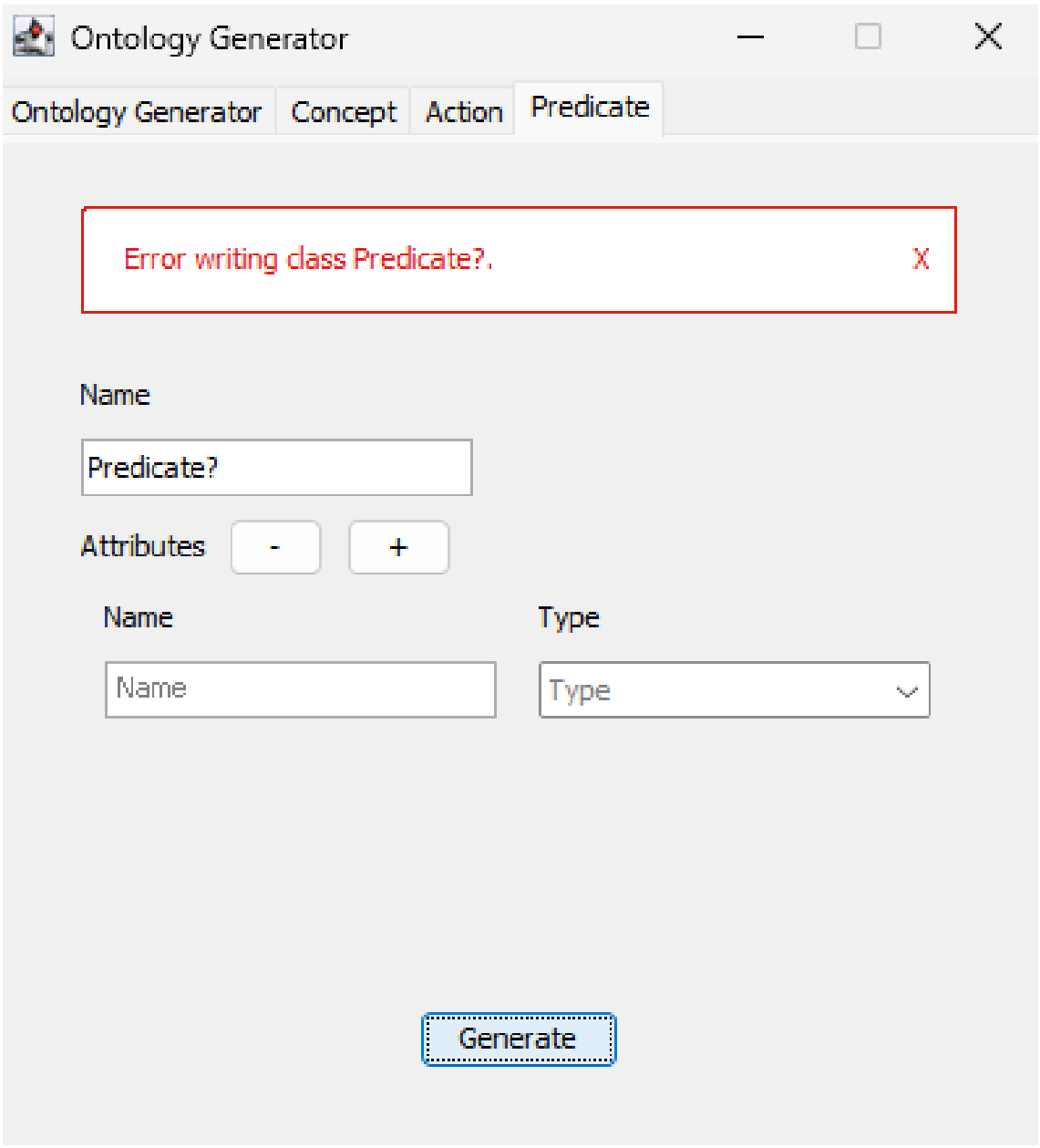


Figure 4.14: Ontology Generator interface - Error new predicate

8. When you have generated all ontology items, you should **close the user interface** by clicking on the cross button on the top right of the screen. This action will initialize the **generation of the last files of the ontology**: the ontology vocabulary file, the ontology file, and an ontology specification file with all the items of the ontology.

9. In the destination path, you should have all the ontology files plus the ontology specification file (**ontologySpecificationFile_yyyyMMdd.txt**). The ontology specification file is the file that you have to load in the Ontology Generator in order to continue editing the ontology.

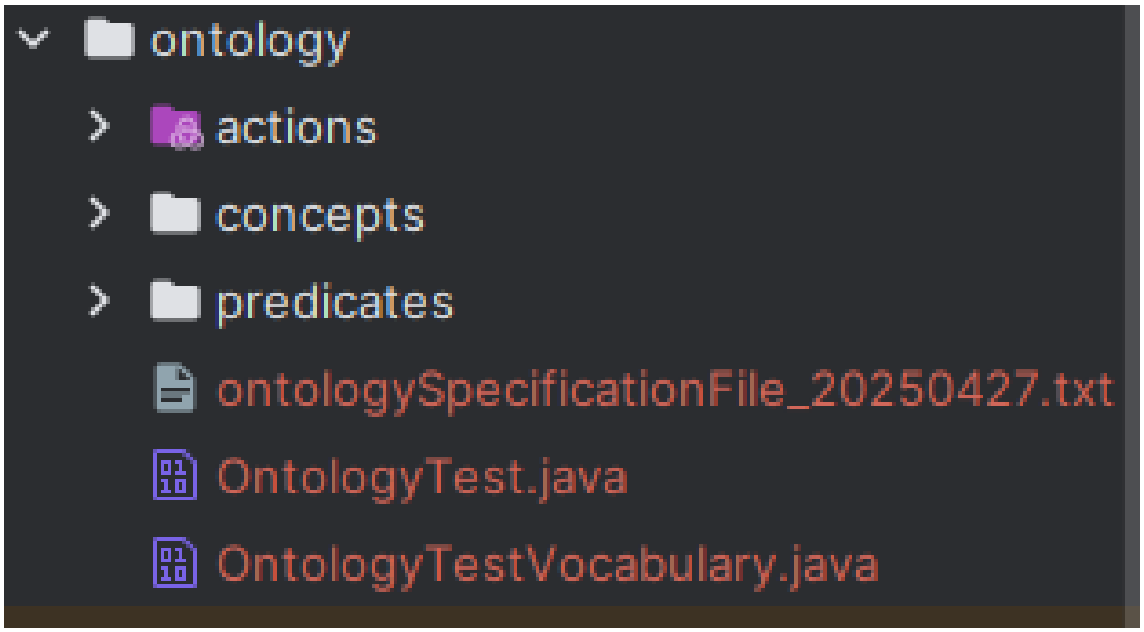


Figure 4.15: Ontology Generator interface - Final structure of the destination path

4.4. Response codes

The Ontology Generator application has different response codes depending on the result of the ontology generation. In case of generation through the **console**, these response codes appear in the **logs of the program**. However, for generation through the **user interface**, the **interface manages the codes** and shows the error messages on the corresponding screen. The response codes are complemented with a descriptive message and they are divided in two groups:

- **Success response code:** The Ontology Generator returns this code to indicate that the ontology has been generated correctly.
  - **OK** (000): *"Operation completed successfully"*
- **Error response codes:** The Ontology Generator returns these codes when an error occurs during generation, they can be divided in two subgroups depending on the source of the error.
  - **Bad request:** Related with errors in the user input (ontologySpecificationFile). To avoid these errors, it is advisable to read chapters 1 and 2 of this document.
    - **KO_SPECIFICATION** (003): *"Error in specification file [description of the error]"*.
    - **KO_INCORRECT_GROUP** (004): *"Incorrect group in specification file: [invalid group]"*.
    - **KO_FILE_NOT_FOUND** (005): *"File not found [path of the specification file]"*.
    - **KO_INCORRECT_CLASS_SPECIFICATION** (006): *"Incorrect specification of class: [name of the class]"*.

- **System error:** Related to an internal service error. These errors are unlikely to occur if the Ontology Generator source code is not modified.
  - **KO_READING** (001): *"Error reading specification file [path of the specification file]"*.
  - **KO_WRITING** (002): *"Error writing class [name of the class]"*.

12.2. Agents Game - API

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

European Master in Software Engineering

Master Thesis

Agents Game: API

Authors: José Rodríguez de Santiago and Adrián Sánchez Rodero

Version control

Version	**Date**	**Change**
v1.0	13/03/2025	First definition of the API
v1.1	10/05/2025	Update the definition of the API (Create unit, End game and Attack) and add new functionalities (Update health and Kill unit)
v1.2	11/05/2025	Added notes on API behaviour and little corrections on

		responses and model names

Table of Contents

1 Notes on API Behavior

All endpoints must be accessed through http://localhost:5002.

If a request is sent to a non-existent endpoint or uses an unsupported HTTP method (only POST is supported), the server will respond with HTTP 404 Not Found and a message indicating "Invalid endpoint or method."

If the request body is malformed, missing required fields, or does not match the expected structure for the target endpoint, the server will respond with HTTP 500 Internal Server Error.

The response body will include an error message indicating the cause of the failure.

For correctly structured and valid requests, the action is queued on the Unity main thread and a response with HTTP 200 OK is returned. The response body contains a message confirming that the action has been queued (e.g., "Map initialization queued.", "Unit movement queued.").

Some endpoints such as /startGame do not require a request body. In these cases, a POST request without a body is considered valid and will be processed correctly.

2 Description of the endpoints

2.1 Initialize the map

Method	POST
Endpoint	/initializeMap
Description	Queues the initialization of the game map with a specified number of players.

Request body	InitializeMapRequest
	{
	“numPlayers”: 6
	}

Response	200 OK: Map initialization queued.

2.2 Start game

Method	POST
Endpoint	/startGame
Description	Queues the start of the game.

Request body	No request body
Response	200 OK: Game start queued.

2.3 End Game

Method	POST
Endpoint	/endGame
Description	Queues the end of the game, displaying results for specified teams.

Request body	EndGameRequest
	{
	“teams”:
	[
	{
	“teamName”: “Red Team”,
	“points”: 150
	},
	{
	“teamName”: “Blue Team”,
	“points”: 120
	}
	]
	}

Response	200 OK: Game end queued.

2.4 Create unit

Method	POST
Endpoint	/createUnit
Description	Queues the creation of a unit at a specified position with given type, ID, and team.
Request body	CreateUnitRequest { “x”: 10, “y”: 10, “id”: “unit_11”, “type”: 1, “team”: 1

	}
Response	200 OK: Unit creation queued.

2.5 Move

Method	POST
Endpoint	/move
Description	Queues movement of a unit to a target location.

Request body	MoveUnitRequest
	{
	“id”: “unit_11”,
	“toX”: 2,
	“toY”: 6
	}

Response	200 OK: Unit movement queued.

2.6 Attack

Method	POST
Endpoint	/attack
Description	Queues an attack between two units, applying damage and updating health.

Request body	AttackUnitRequest
	{
	“attackerId”: “unit_11”,
	“targetId”: “unit_25”,
	“damage”: 2,
	“targetHealthLeft”: 50
	}

Response	200 OK: Attack queued.

2.7 Update health

Method	POST
Endpoint	/updateHealth
Description	Queues a health update for a unit.

Request body	UpdateHealthRequest
	{
	“unitId”: “unit_11”,
	“healthLeft”: 50
	}

Response	200 OK: Health update queued.

2.8 Kill unit

Method	POST
Endpoint	/killUnit
Description	Queues the removal of a specified unit from the game.

Request body	KillUnitRequest
	{
	“unitId”: “unit_25”
	}

Response	200 OK: Unit kill queued.

3 Description of the models

3.1 AttackUnitRequest

Field	Type	Description
attackerId	String	Unique identifier of the unit that performs the attack
targetId	String	Unique identifier of the unit that receives the attack
damage	int	Integer that indicates the damage done by the attack (min 1)
targetHealthLeft	int	Remaining target health (min 0)

3.2 InitializeMapRequest

Field	Type	Description
numPlayers	int	Number of competing teams (2-6)

3.3 EndGameRequest

Field	Type	Description
teams	array<TeamResult>	Team performance data (1 – 6 items)

3.4 TeamResult

Field	Type	Description
teamName	String	Team identifier
points	int	Total points scored (min 0)

3.5 MoveUnitRequest

Field	Type	Description
id	String	Unique identifier of the unit to move
toX	int	New X position of the unit
toY	int	New Y position of the unit

3.6 CreateUnitRequest

Field	Type	Description
x	int	Grid X position of the unit
y	int	Grid Y position of the unit
id	String	Unique identifier of the unit
type	int	Unit type: 1 - Warrior, 2 - Archer, 3 - General, 4 - Healer
team	int	Team affiliation (1-6)

3.7 UpdateHealthRequest

Field	Type	Description
unitId	String	Unique identifier of the unit
healthLeft	int	New health value (min 0)

3.8 KillUnitRequest

Field	Type	Description
unitId	String	Unique identifier of the unit

12.3. Team - Platform: Developer Manual

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

European Master in Software Engineering

Master Thesis

Team - Platform: Developer Manual

Author: Adrián Sánchez Rodero

Version control

Version	Date	Change
v1.0	10/03/2025	First document.
v1.1	18/03/2025	Add AgentsApplication chapter, complete BaseAgent chapter (How to use it? section), and write Set up chapter.
v1.2	27/03/2025	Write GameHttpClient chapter.
v1.3	12/04/2025	Change GameHttpClient and Structure of the project according to the modifications in the API definition.
v1.4	06/05/2025	Write Log standardization chapter.

Table of Contents

1 Set up of the project

This chapter explains the steps to set up the agents project and leave the development environment ready for students to implement their agents and strategies.

The following configurations are specific of Intellij IDEA, in other IDEs the configuration should be similar.

1.1. Basic configuration

This section enumerates the essential steps to prepare the development environment.

1. The first step is to **clone the agents project** from GitHub. You can do this by clicking *File* → *New* → *Project from Version Control*. Then you have to paste the git url of the repository and click *Clone*.

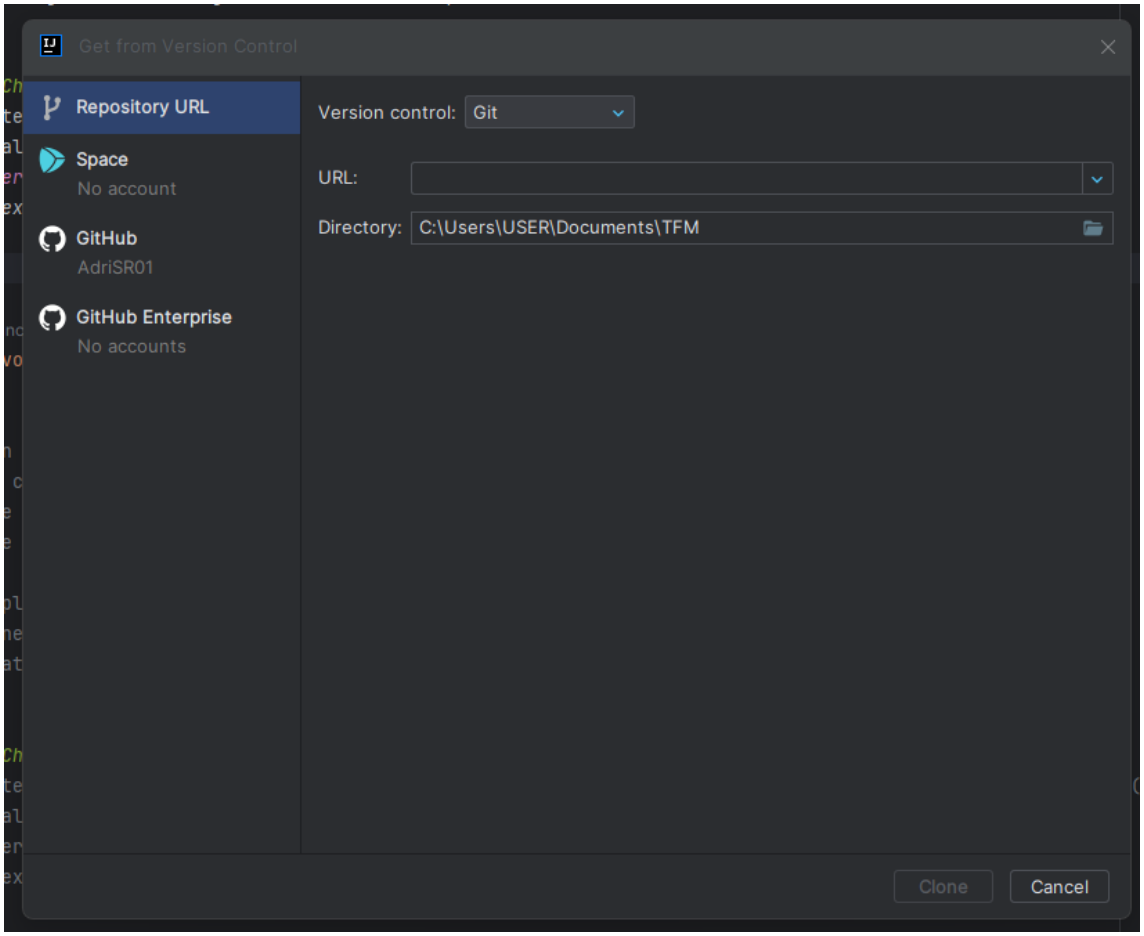


Figure 1.1: Set up - Clone the project

2. Secondly, you have to add the project as a **Maven project**. For this, you have to right click the pom.xml file of the project and click *Add as Maven*

Project.

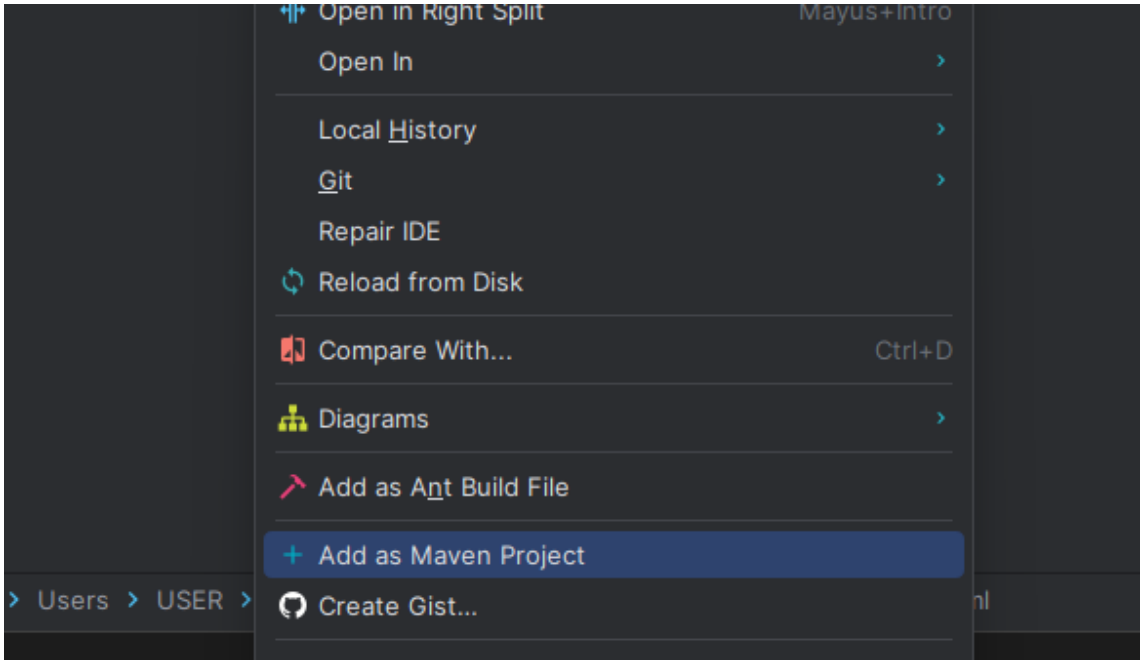


Figure 1.2: Set up - Add as Maven Project

3. If you have done the previous step correctly, the project should appear in the right toolbar of the IDE in the Maven section.

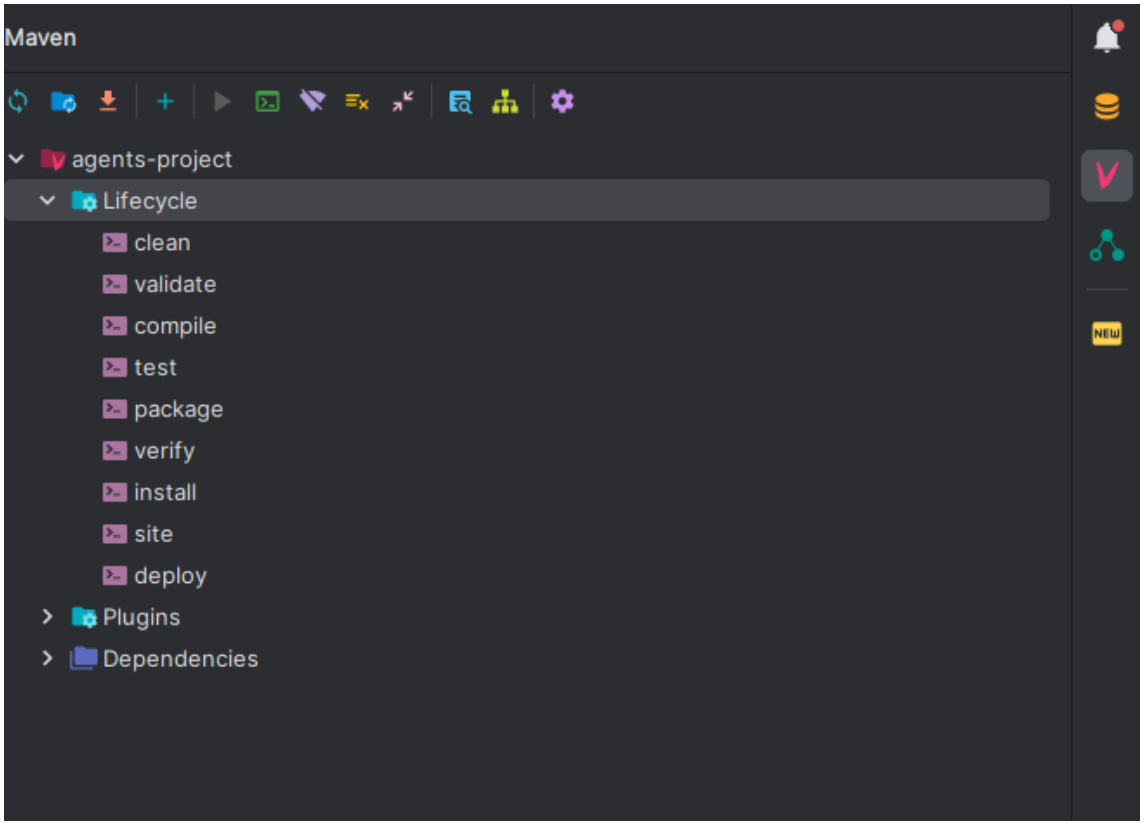


Figure 1.3: Set up - Maven project

4. The next step is to **download the dependencies** of the project. You have to execute the **clean** and **install** scripts that appear under the name of the project in the Maven section (the image in the previous step shows the list of scripts available). Double click on clean and when finished double click on install.

At this point, the project should run without any error. But there are still some actions in the next sections that will be useful during the development.

1.2. Run configurations

We can establish the **running configurations** of the project. We are going to add two configurations: one for the **debug** mode (-d) and another for the **build** mode (-b). These configurations are explained in detail in the third chapter of this document (3.2).

1. Click on the three points on the top bar near the debug icon and select the *Edit* option.

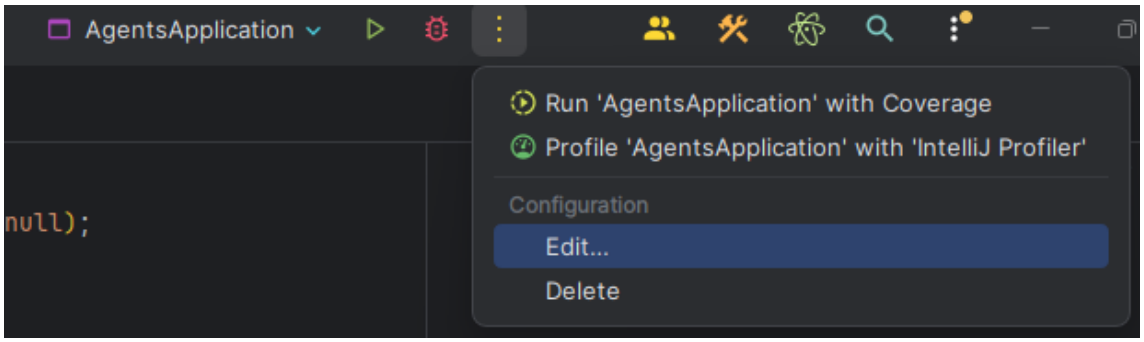


Figure 1.4: Run configurations - Open configurations

2. In the popup, click on the plus icon and select *Application*.

3. Add the debug configuration as follows:

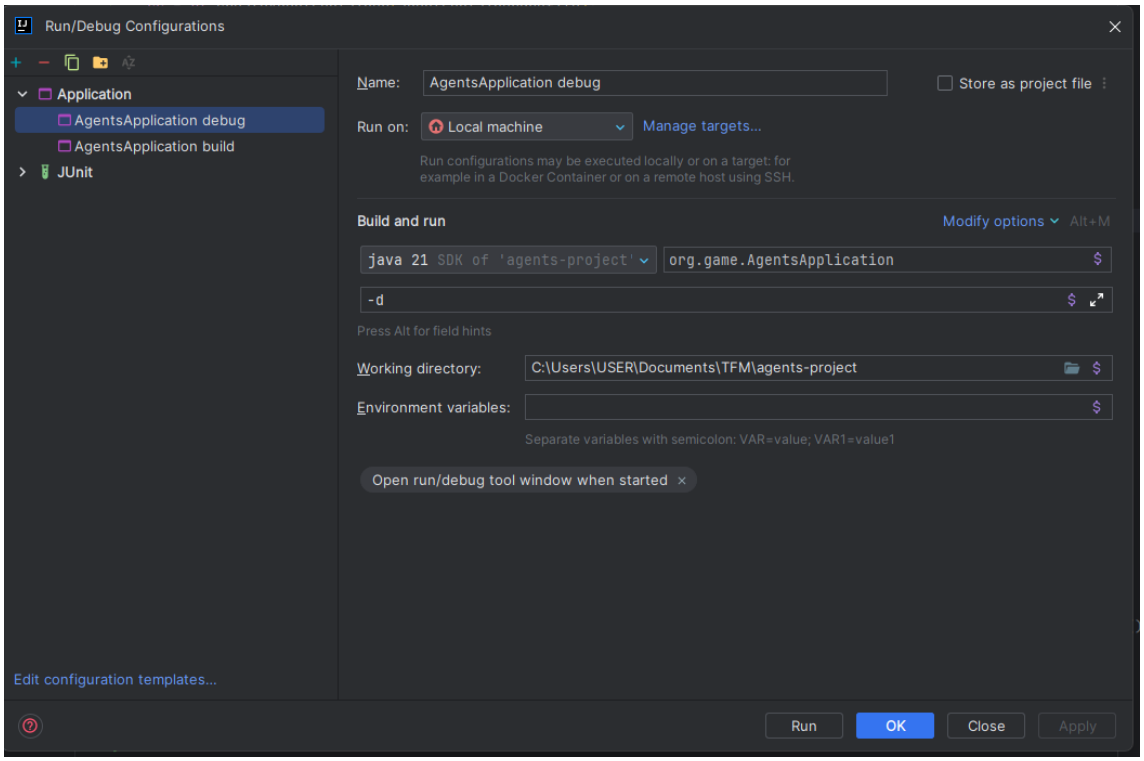


Figure 1.5: Run configurations - Debug configuration

4. Add the build configuration as follows:

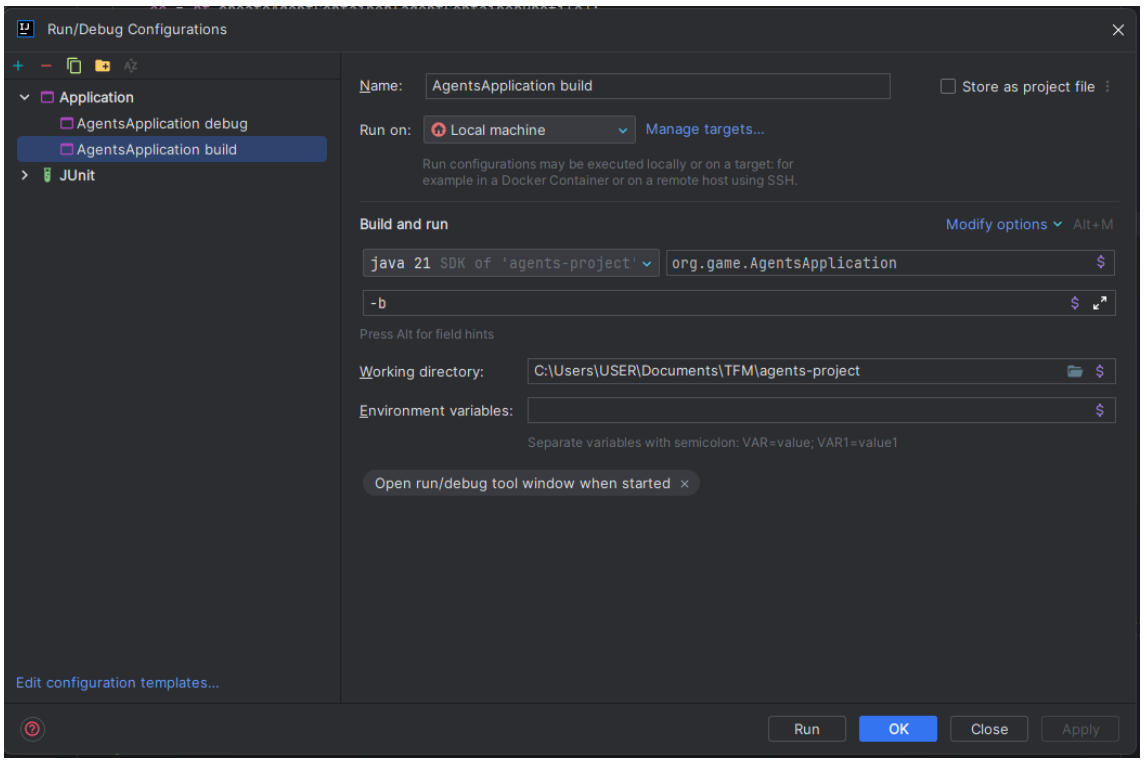


Figure 1.6: Run configurations - Build configuration

5. Click on *Apply* and *OK*.

Now you can run the application by selecting the configuration in the top bar and clicking on the run/debug icon.

1.3. Jar file configuration

You have to configure the generation of the **jar file of your team**. This step is essential to **test your agents** on the platform of other teams.

1. Click on the three points on the right side of the top bar and select the *Project Structure* option.

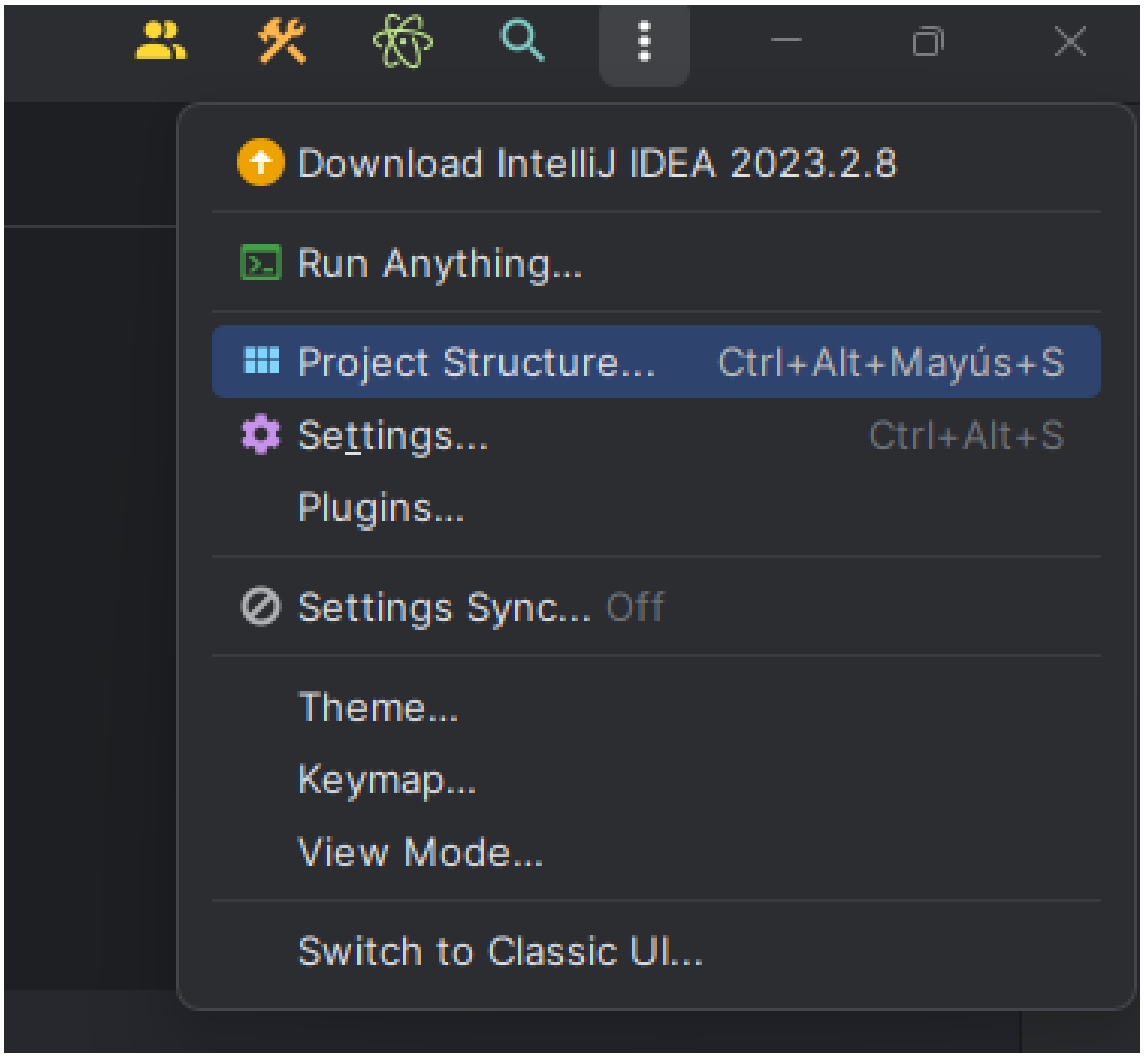


Figure 1.7: Jar configuration - Open Project Structure

2. In the *Artifacts* section, click the plus icon → *JAR* → *From modules with dependencies*.

3. Search for the main class of the project and select it. Then click *OK*.

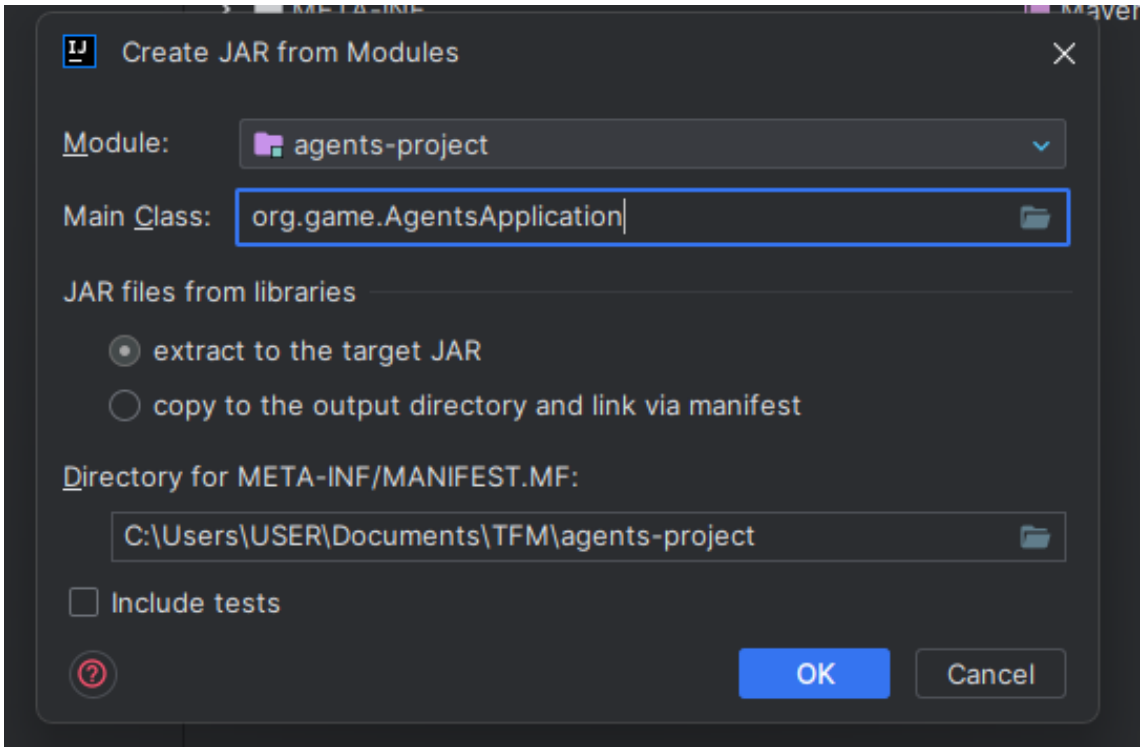


Figure 1.8: Jar configuration - Main class

4. Check the *Include in project build* option of the configuration and change the name of the JAR to **woa-project-teamN.jar** where N must be the number of your team.

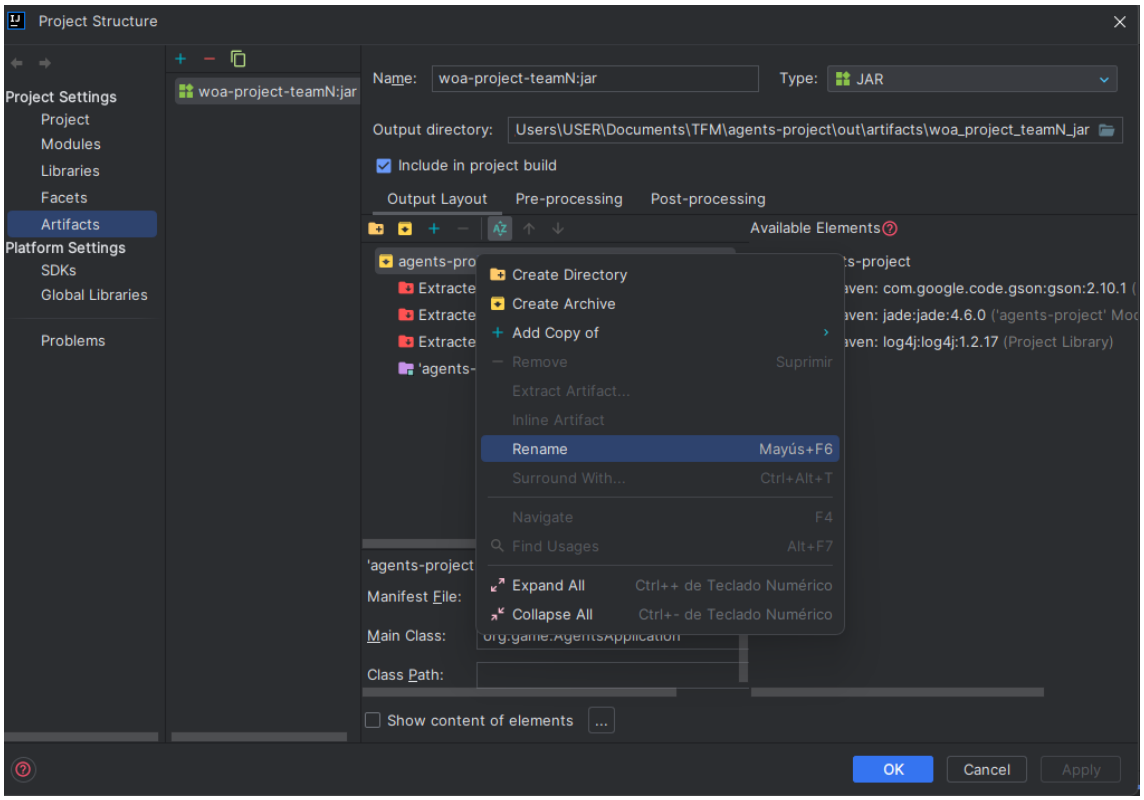


Figure 1.9: Jar configuration - Rename

5. Click *Apply* and *OK*.

If you have configured the artifact correctly, each time you build the project, a jar file will be generated in the **out folder** of your project with the name woa-project-teamN.jar.

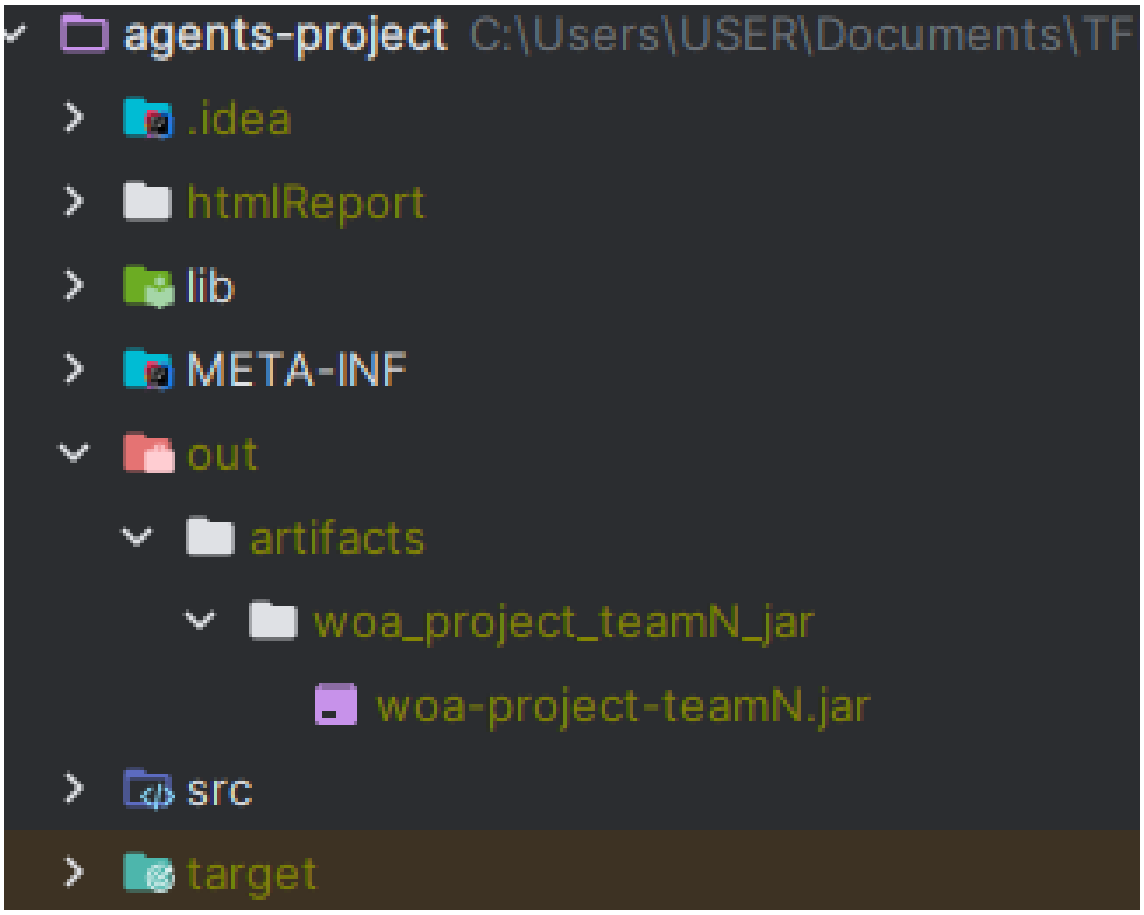


Figure 1.10: Jar configuration - out folder

This is the **jar file that you have to share with other teams** to test your agents on other platforms.

2 Structure of the project

This chapter explains the structure of the Team - Platform project with the functionality that is already implemented for the students.

The structure of the project is the following.

- lib: Folder to add external libraries that cannot be downloaded via Maven.
  - jade-4.6.0.jar: JADE library.
- src.main: Main folder of the project.
  - java.org.game: Source code of the project.
    - agents: Folder with the implementation of the project agents.
      - platform: Folder to store platform agents.
        - AgPlatform: Example of a platform agent.
      - teamN: Folder with the agents of the team.
        - AgTeam: Example of a team agent.
      - BaseAgent: Agent with the common logic for all agents. It is described in detail in chapter 4 of this document.
      - MessagesManager: Interface that defines the methods to send and reply messages between agents.
    - clients: Folder with the different clients for external communications.
      - GameHttpClient: Client to establish the connection with the frontend part of the game. It is described in detail in chapter 5 of this document.
      - InterfaceCommunication: Interface that defines the methods to communicate with the frontend part.
    - config: Folder to store configuration files.
      - AgentLogLayout: Custom log configuration. It is described in detail in chapter 6 of this document.
    - dto: Folder with request classes to communicate with the frontend part.
      - AttackRequest: Definition of the request body required by the API to throw an attack to another unit.
      - ConfigRequest: Definition of the request body required by the API to configure the map.
      - EndGameRequest: Definition of the request body required by the API to end the game and to show final results.
      - KillUnitRequest: Definition of the request body required by the API to kill a unit.

- MoveRequest: Definition of the request body required by the API move a unit.
- TeamResult: Definition of the result of a team required in EndGameRequest.
- UnitRequest: Definition of the request body required by the API to create a new unit.
- UpdateHealthRequest: Definition of the request body required by the API to update the health of a unit.

- models: Optional folder to store the model used in the application.
- ontology: Folder with the ontology for communication between agents. Initially, it contains a mock ontology that students must replace for the real one.
- utils: Optional folder to store auxiliary classes.
  - LogColors: Class with the definitions of the log colors (platform logs, client logs and agents logs).
  - Utils: Class with some constants and methods that can be useful for students in the development of the project.
- AgentsApplication: File that has the main method of the application. This is the file that must be executed to create the agents and start the game. Chapter 3 contains a complete description of this class.

- resources: Folder with project resources.
  - log4j.properties: Properties of log4j logging framework.

- teams: Folder to store the JAR files of other teams.
- pom.xml: File with the project dependencies downloaded via Maven.

The **BaseAgent** and **GameHttpClient** classes are already implemented and students do not need to modify them. The **AgentsApplication** class is almost implemented and only requires students to perform some changes to specify the ontology (3.3) and to add agents from other teams (3.4).

3 AgentsApplication

3.1. Description of methods provided

This section describes the methods (public and private methods) of the AgentsApplication class. These methods are provided to students to build the agents environment.

- ***main(programArguments)***: Main method of the program. It receives the arguments and builds the agents environment.
- ***loadBoot()***: Method that loads the JADE GUI and creates a default profile for the agents.
- ***loadMyPlatformAgents()***: Method that creates and starts the AgPlatform agent inside a container.
- ***loadMyTeamAgents()***: Method that creates and starts the AgTeam agent inside a container.
- ***loadTeamXAgents()***: Method that creates and starts the AgTeam agent of team X inside a container using the corresponding jar in the teams folder.
- ***printHelp()***: Method to print the help text of the program (options -h and --help).

All these methods are static and void.

3.2. How to run the application?

To run the agents application, you can use the following command with the corresponding jar file.

```
java -jar woa-project-teamN.jar [option]
```

where:

- **option**: Argument to indicate the mode to run the agents environment. It can have six values:
  - -h and --help: Displays the application help text.
  - -d and --debug: Run the program loading only local agents from the project.
  - -b and --build: Run the program loading local agents and the agents from the provided jars in the teams folder.

3.3. Necessary actions

There are two actions that the AgentsApplication class requires students to do before running it to test their game-specified code.

- **Ontology:** They should change the mock ontology provided for the real ontology that they will be defining in the Agents Software Development class. There are three places where they have to change it:
  - In the **ontology folder** of the project, they should add the files of the real ontology.
  - In the **loadMyAgents method** of the AgentsApplication class, they should pass the real ontology as argument in the creation of the AgTeam agent.
  - In the **loadMyPlatformAgents method** of the AgentsApplication class, they should pass the real ontology as argument in the creation of the AgPlatform agent.
- **Other teams jar files:** If they want to test the application with the build option (adding agents from other teams), they should follow the steps described in the next section (3.4) to prepare the environment.

3.4. How to add agents from other teams?

This section explains the steps to add and run the application with agents from other teams. First, students should obtain the jar file from other team (**woa-project-teamY.jar**) and then continue with the next steps:

1. Add woa-project-teamY.jar to the **teams folder** of the project.
2. Right-click on the jar file and click the **Add as Library** option.

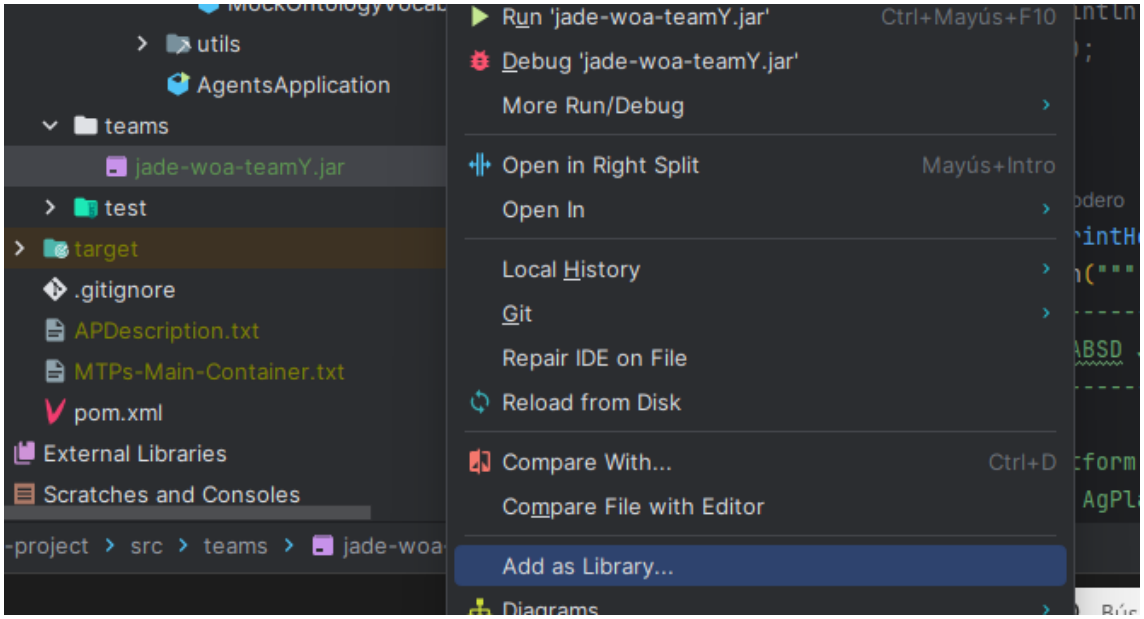


Figure 3.1: How to add agents from other teams?: Step 2

3. Duplicate the method **loadTeamXAgents** of the AgentsApplication class.
4. Change X for Y in the name of the method (**loadTeamYAgents**).
5. Change X for Y in all occurrences within the method.
   - Constant: **TEAM_X** → **TEAM_Y**.
   - Path of AgTeam: **org.game.agents.teamX.AgTeam.class** → **org.game.agents.teamY.AgTeam.class**
   - Catch log: **Error creating agents from tribe X** → **Error creating agents from tribe Y**.

```
private static void loadTeamYAgents() {
    ProfileImpl agentContainerProfile = new ProfileImpl( host: null,  port: 1200,  platformID: null);
    agentContainerProfile.setParameter(Profile.CONTAINER_NAME, TEAM_Y);
    cc = rt.createAgentContainer(agentContainerProfile);

    try {
        //TODO Change ontology: MockOntology for real ontology
        cc.createNewAgent(TEAM_Y, org.game.agents.teamY.AgTeam.class.getName(), new Object[]{MockOntology.getInstance()}).start();
    } catch (StaleProxyException e) {
        System.err.println("Error creating agents from tribe Y!!!");
        System.exit( status: 1);
    }
}
```

Figure 3.2: How to add agents from other teams?: Step 5

6. Call the method loadTeamYAgents in the **main method** of the AgentsApplication class in the **--build branch**.

```
public static void main(String[] args) {

    System.setProperty("java.util.logging.SimpleFormatter.format", ANSI_WHITE + "[%1$tF %1$tT] [%4$-7s] %5$s %n");

    if (args.length == 0 || args[0].equals("-h") || args[0].equals("--help")) {
        printHelp();
        System.exit( status: 0);
    }

    System.out.println("Starting...");
    loadBoot();
    loadMyPlatformAgents();
    Thread.sleep( millis: 2000);

    if (args[0].equals("-d") || args[0].equals("-debug")) {
        loadMyTeamAgents();
    } else if (args[0].equals("-b") || args[0].equals("-build")) {
        loadTeamYAgents();
        loadMyTeamAgents();
    }

    System.out.println("Started successfully AgentsApplication");
}
```

Figure 3.3: How to add agents from other teams?: Step 6

7. Run the program with the --build option.

```
java -jar woa-project-teamN.jar --build
```

*This configuration is specific of Intellij IDEA, in other IDEAs the configuration should be similar.

4 BaseAgent

4.1. What is the BaseAgent?

The BaseAgent is an abstract class that implements the common functionality of all agents in the game. It establishes the **ontology** for the agent, initializes the **agent logger**, and allows the developers to add the **behaviors** of the agents through an abstract method.

Furthermore, it implements the **MessagesManager** interface that contains methods to send or reply messages with a specific content. The MessagesManager interface also defines methods for managing the content of messages. The content of a message can be in two different locations inside a message:

- In the **content** field of the message. This option requires the content to be an action of the ontology (the concepts are also actions).
- In the **content object** field of the message.

The BaseAgent uses the first option in the implementation of the specific methods of the interface to send or reply a message, but it provides the necessary methods to create a message, set and get the content object of a given message.

4.2. How is it implemented?

This section explains the methods and properties that the BaseAgent abstract class provides to his children and the abstract methods that the children must implement.

4.2.1. Properties

The BaseAgent abstract class provides to his children three protected properties.

- **Logger logger**: The agent logger that it is recommended to use it to facilitate the debugging of the code. The logger works as explained in chapter 6.
- **Codec codec**: The codec of the ontology.
- **Ontology ontology**: The ontology that the agent uses for the communications.

4.2.2. Agent specific methods

The BaseAgent abstract class overrides the setup method of the Agent class to set some properties.

- ***setup()***: This method establishes the language and the ontology for communications between agents. The **ontology** must be specified in the **first position of the array of arguments** passed in the creation of the agent,

otherwise, this method will delete the agent. In addition, this method initializes the logger with the agent name.

4.2.3. Abstract methods

The BaseAgent abstract class has one abstract method that his children must implement.

- ***configure()***: This method allows the children to specify configurations of the agent, for example, adding behaviors. It is executed inside the agent setup method.

4.2.4. MessagesManager methods

The BaseAgent abstract class implements the MessagesManager interface and, as a result of this, includes methods to send or reply messages with each type of performative and to fill or extract the content of a message.

The interface defines two types of methods to create and send messages between agents.

- The methods with the pattern ***send[Performative]*** receive a protocol, a list of receivers, and a content (null if not needed). These methods create an ACLMessage with the given parameters and send it.
- The methods with the pattern ***replyWith[Performative]*** receive a message and a content (null if not needed). These methods create a reply to the given message, add the content if specified, and send it.

The descriptions of **all methods** from the MessagesManager interface are as follows:

- ***newMsg(String protocol, List<AID> receivers, int perf)*** → ***ACLMessage***: Create an ACLMessage with the protocol and performative indicated, and set the receivers to the given list. Returns the message ready to be sent.
- ***extractConcept(ACLMessage msg, Class<T>type)*** → ***T***: Extract and return the concept of class T within the given message.
- ***getContentObject(ACLMessage msg, Class<T>type)*** → ***T***: Get and return the content object of class T within the given message.
- ***setContentObject(ACLMessage msg, T object)***: Set as content object of the given message the indicated object.
- ***sendAcceptProposal(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to ACCEPT_PROPOSAL.
- ***replyWithAcceptProposal(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to ACCEPT_PROPOSAL.
- ***sendAgree(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to AGREE.
- ***replyWithAgree(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to AGREE.

- ***sendCancel(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to CANCEL.
- ***replyWithCancel(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to CANCEL.
- ***sendCFP(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to CFP.
- ***replyWithCFP(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to CFP.
- ***sendFailure(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to FAILURE.
- ***replyWithFailure(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to FAILURE.
- ***sendInform(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to INFORM.
- ***replyWithInform(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to INFORM.
- ***sendInformIf(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to INFORM_IF.
- ***replyWithInformIf(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to INFORM_IF.
- ***sendNotUnderstood(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to NOT_UNDERSTOOD.
- ***replyWithNotUnderstood(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to NOT_UNDERSTOOD.
- ***sendPropose(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to PROPOSE.
- ***replyWithPropose(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to PROPOSE.
- ***sendQueryIf(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to QUERY_IF.
- ***replyWithQueryIf(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to QUERY_IF.
- ***sendQueryRef(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to QUERY_REF.
- ***replyWithQueryRef(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to QUERY_REF.
- ***sendRefuse(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REFUSE.
- ***replyWithRefuse(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REFUSE.

- ***sendRejectProposal(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REJECT_PROPOSAL.

- ***replyWithRejectProposal(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REJECT_PROPOSAL.

- ***sendRequest(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to REQUEST.

- ***replyWithRequest(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to REQUEST.

- ***sendSubscribe(String protocol, List<AID> receivers, Concept content)*** → ***boolean***: Set the performative of the created message to SUBSCRIBE.

- ***replyWithSubscribe(ACLMessage msg, Concept content)*** → ***boolean***: Set the performative of the reply message to SUBSCRIBE.

4.3. How to use it?

If we want to create an agent with the functionalities of the BaseAgent, we only have to extend the BaseAgent class. But extending the BaseAgent class implies two more actions:

1. Implement the **configure method**: This method allows us to add behaviors to the agent.

2. In the **agent creation**, it is necessary to pass the **ontology** as the first argument.

The initial project provided to students has two examples of agents that use the BaseAgent functionalities.

- **AgPlatform**: Example of a platform agent.

AgPlatform.java

```java
package org.game.agents.platform;

import ...

4 usages  Adrián Sánchez Rodero
public class AgPlatform extends BaseAgent {

    //1) Instantiate GameHttpClient
    2 usages
    private final GameHttpClient gameClient = new GameHttpClient();

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {

        this.addBehaviour((OneShotBehaviour) () -> {
            AgPlatform agent = (AgPlatform) this.myAgent;

            MessageTemplate template = MessageTemplate.and(
                    MessageTemplate.MatchLanguage(codec.getName()),
                    MessageTemplate.MatchOntology(ontology.getName()));
            template = MessageTemplate.and(template, MessageTemplate.MatchProtocol(PROTOCOL));

            //Wait a message with the protocol InitialCommunication
            ACLMessage msg = agent.blockingReceive(template);
            Concept1 content = agent.extractConcept(msg, Concept1.class);

            logger.info("I have received a message from " + msg.getSender().getLocalName());
            logger.info("The content of the message is: \"" + content.getAt1() + "\"");

            Concept1 contentReply = new Concept1();
            contentReply.setAt1("OK");

            //Reply the message with an AGREE performative
            agent.replyWithAgree(msg, contentReply);
            logger.info("I have sent an AGREE reply to " + TEAM_1);
            logger.info("The content of the reply is \"" + contentReply.getAt1() + "\"");


            //Configure the board and start the game
            agent.play();
        });
    }

    1 usage  Adrián Sánchez Rodero
    private void play() {
        //2) Use the method of the client
        //Initialize the map
        logger.info("I am going to initialize the map with 1 player");
        this.gameClient.initializeMap(new ConfigRequest( numPlayers: 1));

        //Start the game
        logger.info("I am going to start the game");
        this.gameClient.startGame();
    }
}
```

Figure 4.1: AgPlatform class

- **AgTeam**: Example of a team agent.

AgTeam.java

```java
package org.game.agents.teamN;

import ...

2 usages  Adrián Sánchez Rodero
public class AgTeam extends BaseAgent {

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {

        SequentialBehaviour steps = new SequentialBehaviour();
        steps.addSubBehaviour((OneShotBehaviour) () -> {
                AID platformAgent = new AID(PLATFORM_MANAGER, AID.ISLOCALNAME);
                Concept1 content = new Concept1();
                content.setAt1("I am alive");

                //Send a message to AgPlatform with the INFORM performative and InitialCommunication protocol
                ((BaseAgent) this.myAgent).sendInform(PROTOCOL, List.of(platformAgent), content);
                logger.info("I have sent an INFORM message to " + PLATFORM_MANAGER);
                logger.info("The content of the message is: \"" + content.getAt1() + "\"");
        });

        steps.addSubBehaviour((OneShotBehaviour) () -> {
                MessageTemplate template = MessageTemplate.and(
                        MessageTemplate.MatchLanguage(codec.getName()),
                        MessageTemplate.MatchOntology(ontology.getName()));
                template = MessageTemplate.and(template,
                        MessageTemplate.MatchProtocol(PROTOCOL));

                //Wait the response of the InitialCommunication protocol message
                ACLMessage msg = this.myAgent.blockingReceive(template);
                Concept1 content = ((BaseAgent) this.myAgent).extractConcept(msg, Concept1.class);

                logger.info("I have received a message from " + msg.getSender().getLocalName());
                logger.info("The content of the message is: \"" + content.getAt1() + "\"");
        });

        this.addBehaviour(steps);
    }
}
```

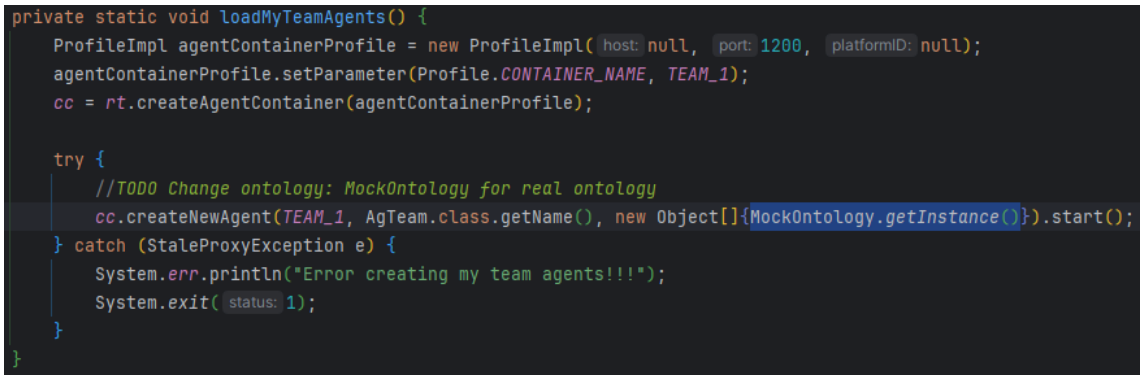
Figure 4.2: AgTeam class

The AgentsApplication class creates these two example agents in the **loadMyPlatformAgents** and **loadMyTeamAgents** methods passing the ontology as the first argument (3.1).

```
private static void loadMyPlatformAgents() {
    ProfileImpl agentContainerProfile = new ProfileImpl( host: null,  port: 1200,  platformID: null);
    agentContainerProfile.setParameter(Profile.CONTAINER_NAME, PLATFORM_MANAGER);
    cc = rt.createAgentContainer(agentContainerProfile);

    try {
        //TODO Change ontology: MockOntology for real ontology
        cc.createNewAgent(PLATFORM_MANAGER, AgPlatform.class.getName(),
                new Object[]{MockOntology.getInstance()}).start();
    } catch (StaleProxyException e) {
        System.err.println("Error creating platform agents!!!");
        System.exit( status: 1);
    }
}
```

Figure 4.3: Creation of AgPlatform in AgentsApplication class



```
private static void loadMyTeamAgents() {
    ProfileImpl agentContainerProfile = new ProfileImpl( host: null,  port: 1200,  platformID: null);
    agentContainerProfile.setParameter(Profile.CONTAINER_NAME, TEAM_1);
    cc = rt.createAgentContainer(agentContainerProfile);

    try {
        //TODO Change ontology: MockOntology for real ontology
        cc.createNewAgent(TEAM_1, AgTeam.class.getName(), new Object[]{MockOntology.getInstance()}).start();
    } catch (StaleProxyException e) {
        System.err.println("Error creating my team agents!!!");
        System.exit( status: 1);
    }
}
```

Figure 4.4: Creation of AgTeam in AgentsApplication class

After creation, these two agents try to establish a conversation: First, AgTeam **sends an INFORM message** using the **InitialCommunication protocol** to AgPlatform. Then AgPlatform receives the messages and **replies with an AGREE**.

There are two ways to add a new agent to the environment.

a. Create the agent in the **AgPlatform** class (if it is a platform agent) or in the **AgTeam** class (if it is a team agent) in the **configure** method. The following box has the code to do that:

```
try {
    this.getContainerController().createNewAgent(agentName,
     ↪  NewAgent.class.getName(), new
     ↪  Object[]{MockOntology.getInstance()}).start();
} catch (StaleProxyException e) {
    System.err.println("Error creating the agent!!!");
    System.exit(1);
}
```

From the above code, it is only necessary to change the name of the agent, use the corresponding agent class and pass the real ontology in the objects array.

b. Create the agent in the **loadMyPlatformAgents** method (if it is a platform agent) or in the **loadMyTeamAgents** (if it is a team agent) method of the

AgentsApplication class. In this case, the necessary code is similar to the creation of AgPlatform and AgTeam agents.

5 GameHttpClient

5.1. What is the GameHttpClient?

The GameHttpClient acts as an **intermediary** between agents and the frontend interface. This client allows the platform agents to **communicate with the interface** to: start the game, end the game and show the final results, create a new unit, move a unit, attack another unit, update the health of a unit, or kill a unit. It performs the necessary HTTP requests and handles the responses.

The GameHttpClient is implemented according the API definition in the **Agents Game - API document**.

5.2. How is it implemented?

This section describes how GameHttpClient is implemented and some peculiarities of the implementation of its unit tests.

5.2.1. Client

The GameHttpClient implements the **InterfaceCommunication interface** that defines the methods to communicate with the frontend part. The InterfaceCommunication interface defines the following methods:

- ***initializeMap(ConfigRequest request)***: Initializes the map with the number of players.
- ***startGame()***: Starts a new game phase, initializing phase-specific actions.
- ***endGame(EndGameRequest request)***: Ends the current game phase and shows final results.
- ***createUnit(UnitRequest request)***: Creates a new unit at the specified coordinates and assigns it to a player.
- ***move(MoveRequest request)***: Moves a unit from one position to another on the game board.
- ***attack(AttackRequest request)***: Performs an attack action between two units and updates the attacked unit's health points.
- ***updateUnitHealth(UpdateHealthRequest request)***: Updates the health of a unit with the new value.
- ***killUnit(KillUnitRequest request)***: Removes a unit from the board.

To implement these methods, GameHttpClient uses the **Gson library** for the request bodies. Gson is a Java library that can be used to convert Java Objects into their JSON representation.

The **request bodies** used in the communications are defined in the dto folder of the project and have the following attributes:

- **AttackRequest**:
  - **attackerId**: Unique identifier of the unit that performs the attack (String).
  - **targetId**: Unique identifier of the unit that receives the attack (String).
  - **damage**: Integer that indicates the damage done by the attack (int).
  - **targetHealthLeft**: Remaining target unit health (int).
- **ConfigRequest**:
  - **numPlayers**: Number of players of the game (int).
- **EndGameRequest**:
  - **teams**: List with the result of each team in the game (List<TeamResult>).
- **MoveRequest**:
  - **id**: Unique identifier of the unit to move (String).
  - **toX**: New X position of the unit (int).
  - **toY**: New Y position of the unit (int).
- **TeamResult**:
  - **teamName**: Team identifier (String).
  - **points**: Total points scored (int).
- **UnitRequest**:
  - **x**: X position of the unit (int).
  - **y**: Y position of the unit (int).
  - **id**: Unique identifier of the unit (String).
  - **type**: Integer that indicates the type of the unit (int).
  - **team**: Team affiliation (int).
- **UpdateHealthRequest**:
  - **unitId**: Unique identifier of the unit (String).
  - **healthLeft**: New health value (int).
- **KillUnitRequest**:
  - **unitId**: Unique identifier of the unit to kill (String).

5.2.2. Unit tests

The unit tests of GameHttpClient (**GameHttpClientTest**) are different from other classes. GameHttpClientTest uses the **wiremock library** to test that the client performs the different requests correctly. This library allows us to create a mock server that responds to the requests emitted by GameHttpClient.

GameHttpClientTest **initializes the mock server** before starting to run the test cases. First, it creates a new WireMockServer in a specified port, and then it starts the server.

```
@BeforeAll
static void init() {
    wireMockServer = new WireMockServer(PORT);
    wireMockServer.start();
}
```

After all tests have been run, it **stops the server**.

```
@AfterAll
static void endTest() {
    wireMockServer.stop();
}
```

In each test case, we have to **mock the response of the server** to the corresponding endpoint. For example, in the next code we are mocking the response of */initializeMap* endpoint to test the initializeMap method of GameHttpClient.

```
wireMockServer.stubFor(post(urlEqualTo(INIT_MAP))
    .withHeader("Content-Type", equalTo("application/json"))
    .willReturn(aResponse().withStatus(200)));
```

And, to **validate** that the server has received a **request at that endpoint**, we have the WireMockServer verify method. In the following code, we are validating that the server has received a POST request at the */initializeMap* endpoint with the corresponding header.

```
wireMockServer.verify(postRequestedFor(urlEqualTo(INIT_MAP))
   .withHeader("Content-Type", equalTo("application/json")));
```

5.3. How to use it?

If we want to communicate with the interface to perform an action, we have to follow these steps:

1. **Instantiate GameHttpClient** in the corresponding class.

   ```
   private final GameHttpClient gameClient = new
    ↪  GameHttpClient();
   ```

2. **Use the method of the client** associated with the action we want to perform passing the required arguments.

   ```
   //Initialize the map
   this.gameClient.initializeMap(new ConfigRequest(1));
   ```

The initial project provided to students has an example of the usage of the GameHttpClient in AgPlatform. This agent, after handling the AgTeam message, uses GameHttpClient to initialize the map and start the game (in the AgPlatform **play method**, we can see the implementation of step 2 above).

AgPlatform.java

```java
package org.game.agents.platform;

import ...

4 usages  Adrián Sánchez Rodero
public class AgPlatform extends BaseAgent {

    //1) Instantiate GameHttpClient
    2 usages
    private final GameHttpClient gameClient = new GameHttpClient();

    1 usage  Adrián Sánchez Rodero
    @Override
    protected void configure() {...}

    1 usage  Adrián Sánchez Rodero
    private void play() {
        //2) Use the method of the client
        //Initialize the map
        logger.info("I am going to initialize the map with 1 player");
        this.gameClient.initializeMap(new ConfigRequest( numPlayers: 1));

        //Start the game
        logger.info("I am going to start the game");
        this.gameClient.startGame();
    }
}
```

Figure 5.1: AgPlatform implementation of step 1 and 2

6 Log standardization

6.1. What is the log standardization?

The log standardization is a functionality to **print messages** on the program console with a standard format. These messages help developers to **debug the code** in case of an error or to check that everything is working correctly.

The logs have a **standard format** that contains the date, the level, the name of the source, and the message.

```
[yyyy-MM-dd HH:mm:ss] LEVEL SourceName: Message
```

There are three possible **levels** for the logs:

- **INFO**: Used to print decision/operation messages. INFO level messages are printed in different colors depending on the source of the message.
  - **Platform agents** print their messages in **white** color.

Figure 6.1: Example of platform log

  - **GameHttpClient** prints its messages in **magenta** color.

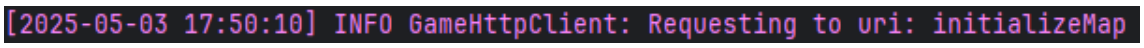


Figure 6.2: Example of GameHttpClient log

  - **Team agents** print their messages in the **color associated with the team**.
- **WARN**: Used to print warning messages. It is associated with the **yellow** color.

Figure 6.3: Example of warning log

- **ERROR**: Used to print error messages. It is associated with the **red** color.

Figure 6.4: Example of error log

6.2. How is it implemented?

The log standardization functionality is implemented using the **log4j logging framework**. This framework allows us to have the three levels of logs already defined in three methods.

- **info(Object message)**: To print information messages.
- **warn(Object message)**: To print warning messages.
- **error(Object message)**: To print error messages.

To customize the logs with the format and color desired, we have created a custom Layout class and override the format method. This implementation is in the **AgentLogLayout class** in the config folder. In this class, the private method **getColorForLevelOrAgent(String name, String level)** is in charge of assigning the color to the logs depending on the **level** and the **logger name**.

- If the level of the message is **ERROR** then the color is red.
- If the level of the mesage is **WARN** then the color is yellow.
- If the logger name is **equal to GameHttpClient** then the color is magenta.
- If the logger name **does not contain the word 'Team'** (Platform agent) then the color is white.
- Otherwise (Team agents), the method extracts the team identifier from the name (Unit5Team2 $\rightarrow$ Team2) and checks if the team already has a color. If it has a color, then the log uses it. If not, assign a color.

The **LogColors class** in the utils folder contains the declaration of the colors. There are **12 different colors reserved for teams**. They are assigned in sequence, and if there are more than 12 teams, then the first color is assigned again.

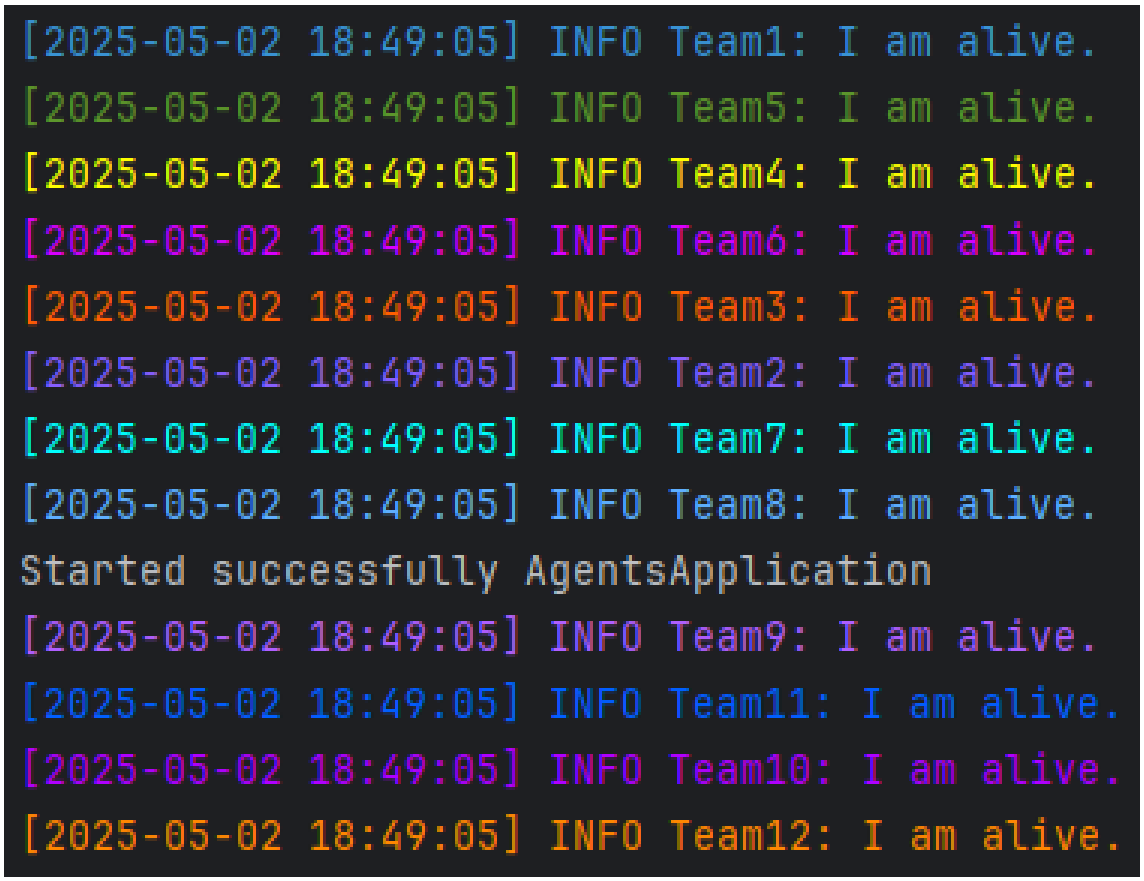


Figure 6.5: Example of 12 team agents logs

Finally, the log4j logging framework needs some properties to indicate the new Layout class. These properties are defined in the **log4j.properties** file of the resources folder.

6.3. How to use it?

To use the log standardization functionality we have to distinguish two cases:

- In an **agent**: The BaseAgent implementation has already integrated the functionality in its **protected logger property**. So, its child classes only need to call the log4j methods (info, warn and error) to print a message. The logger layout will assign the color depending on the level of the message and the agent name. In this aspect, it is **very important** that **team agents names** contain the word **'Team'** following by the identifier of the team (Unit5Team1, Team4Unit1, ManTeam2Unit1...), and the **platform agents names** do **not** contain the **'Team' word** to avoid possible problems.
- In **another class**: In this case, it is necessary to **initialize the logger** with a logger name in the class, **add the color** we want for the new class in the LogColors class and **add a new if statement** in the getColorForLevelOrAgent method of the AgentLogLayout to return the new color. The **GameHttpClient** logger is an example of this.

```
import org.apache.log4j.Logger;
```

Figure 6.6: Example of logger import

```
private final Logger logger = Logger.getLogger( name: "GameHttpClient");
```

Figure 6.7: Example of GameHttpClient logger initialization

```
public static final String CLIENT_COLOR = "\u001B[38;5;213m";   // Bright magenta
```

Figure 6.8: Example of GameHttpClient color

```
private String getColorForLevelOrAgent(String name, String level) {
    switch (level) {
        case "ERROR":
            return LogColors.RED;
        case "WARN":
            return LogColors.YELLOW;
        case "INFO":
        default:
            if (name.equals("GameHttpClient")) {
                return LogColors.CLIENT_COLOR;
            }
            if (!name.contains("Team")) {
                return LogColors.PLATFORM_COLOR;
            }
            String teamId = name.replaceAll( regex: ".*(Team\\d+).*", replacement: "$1");
            return teamColors.computeIfAbsent(teamId, k -> LogColors.TEAM_COLORS[teamColors.size() %
                    LogColors.TEAM_COLORS.length]);
    }
}
```

Figure 6.9: Example of GameHttpClient if statement