.Software design
Team project – Deliverable 2

Team number: **28**

Team members: 5

Name	**Student Nr.**	**E-mail**
Fabiola Loffredi	2617929	f.loffredi@student.vu.nl
Carmen Toyas Sánchez	2622278	c.toyassanchez@student.vu.nl
Roshini Ramasamy	2610864	r.ramasamy@student.vu.nl
Cristiano Milanese	2600169	c.milanese@student.vu.nl
Anna Claire Favié	2626585	a.c.favie@student.vu.nl

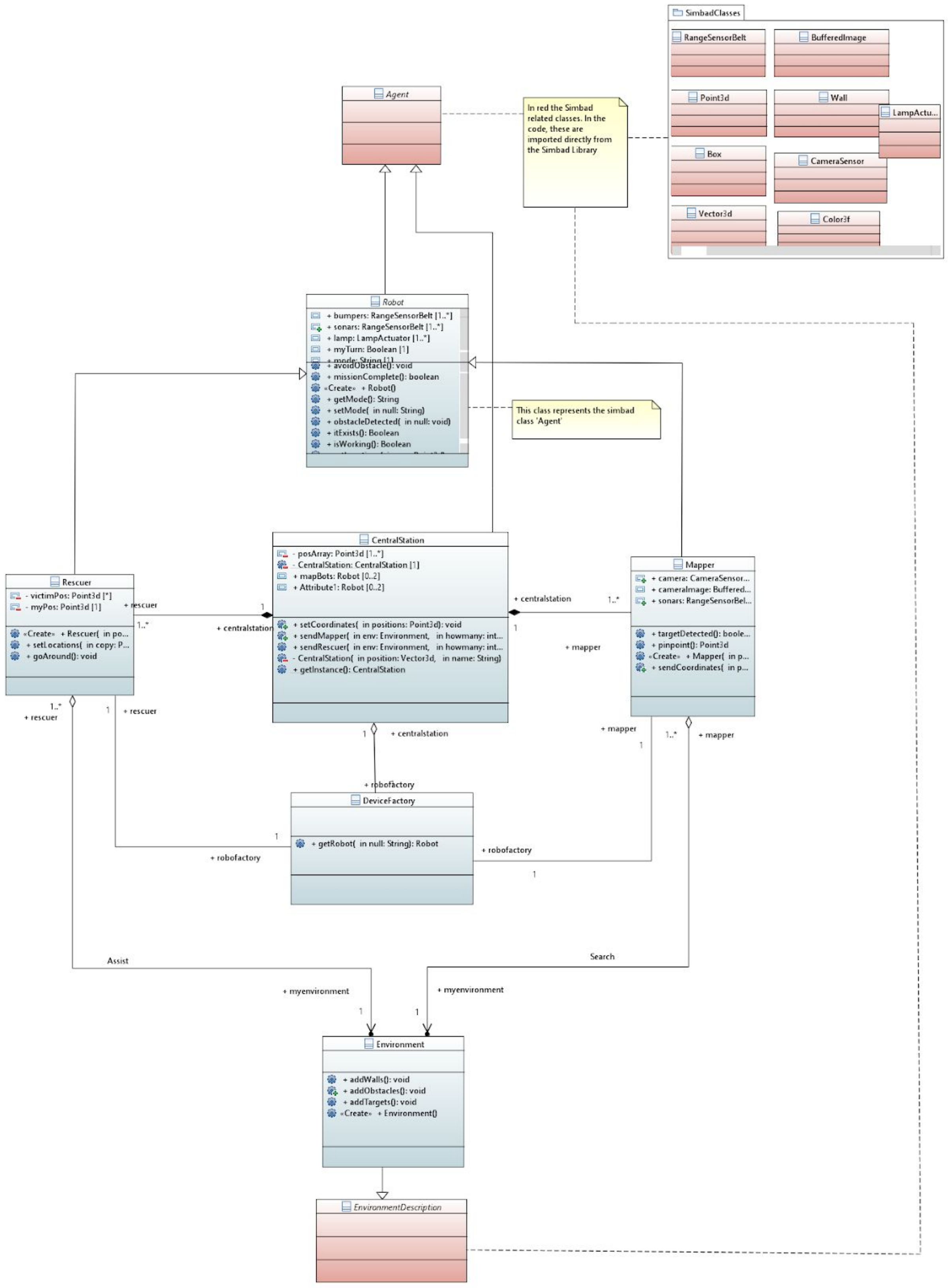
This document has a maximum length of 10 pages.

Table of Contents

1. Class diagram

Author(s): Fabiola Loffredi, Carmen Toyas Sánchez

This class diagram was updated


SimbadClasses
RangeSensorBelt
BufferedImage
Point3d
Wall
LampActu...
Box
CameraSensor
Vector3d
Color3f
Agent
In red the Simbad related classes. In the code, these are imported directly from the Simbad Library
Robot
+ bumpers: RangeSensorBelt [1..*]
+ sonars: RangeSensorBelt [1..*]
+ lamp: LampActuator [1..*]
+ myTurn: Boolean [1]
+ mode: String [1]
+ avoidObstacle(): void
+ missionComplete(): boolean
«Create» + Robot()
+ getMode(): String
+ setMode(in null: String)
+ obstacleDetected(in null: void)
+ itExists(): Boolean
+ isWorking(): Boolean
This class represents the simbad class 'Agent'
CentralStation
- posArray: Point3d [1..*]
- CentralStation: CentralStation [1]
+ mapBots: Robot [0..2]
+ Attribute1: Robot [0..2]
+ setCoordinates(in positions: Point3d): void
+ sendMapper(in env: Environment, in howmany: int...
+ sendRescuer(in env: Environment, in howmany: int...
- CentralStation(in position: Vector3d, in name: String)
+ getInstance(): CentralStation
Rescuer
- victimPos: Point3d [*]
- myPos: Point3d [1]
«Create» + Rescuer(in po...
+ setLocations(in copy: P...
+ goAround(): void
Mapper
+ camera: CameraSensor...
+ cameraImage: Buffered...
+ sonars: RangeSensorBel...
+ targetDetected(): boole...
+ pinpoint(): Point3d
«Create» + Mapper(in p...
+ sendCoordinates(in p...
+ rescuer
1..*
1
+ centralstation
+ centralstation
1..*
1
+ mapper
1..*
+ rescuer
1
+ rescuer
1
+ centralstation
+ mapper
1
1..*
+ mapper
+ robofactory
DeviceFactory
+ getRobot(in null: String): Robot
1
+ robofactory
+ robofactory
1
Assist
Search
+ myenvironment
1
+ myenvironment
1
Environment
+ addWalls(): void
+ addObstacles(): void
+ addTargets(): void
«Create» + Environment()
EnvironmentDescription

CentralStation: This class represents the Central Station which makes use of the **BotFactory**, and manages the two different types of robots. It also stores all the data retrieved by the **Mapper** robot.

It is associated with **DeviceFactory** through a shared association, as the **DeviceFactory** can exist without the central station and to **Rescuer** and **Mapper** through a composition.

Attributes:

- + posArray: Point3d [1..*] : an array of type Point3d that stores the coordinates computed by **Mapper** robots. It is shared with the **Rescuer** robots once a *centralStation* object stops the mappers.
- -CentralStation:CentralStation[1] : this attribute represents the private instance of CentralStation, needed for the design pattern Singleton

Operations:

- + setCoordinates(in positions: Point3d): void : a Point3d operation that is used to retrieve the coordinates from the array and takes a Point3d variable as a parameter
- + sendMapper(in env: Environment, in howmany: int): void : a void operation that sends a **Mapper** type robot in the environment. Takes the *Environment*, and the number of robots to send as parameters
- + sendRescuer(in env: Environment, in howmany: int):void : a void operation that sends a **Rescuer** type robot in the environment. Takes the *Environment*, and the number of robots to send as parameters.
- -CentralStation(in position: Vector3d, in name: String): constructor of the class, extending Robot and therefore the super constructor. It is private in order to have one instance only at any given moment.
- + getInstance(): CentralStation : a static function that returns the instance of *CentralStation*

Robot (Abstract class): an abstract class that defines some common characteristics for other robot subclasses. It is a generalization for **Mapper, Rescuer**, and **CentralStation** and extends the built-in Simbad class **Agent.**

Attributes:

- +bumpers: RangeSensorBelt[1..*] : this attribute represents the instances for bumper sensors of type **RangeSensorBelt**. The number of instances can range from 1 to any other number.
- +sonars: RangeSensorBelt[1..*] : this attribute represents the instances for sonar sensors of type **RangeSensorBelt**. The number of instances can range from 1 to any other number.
- +lamp: LampActuator[1..*] : this attribute represents the only instance of the lamp.
- +myTurn: Boolean[1]:this attribute checks whether a robot can move or not.
- mode: String[1]: represents the state the robot is currently in.

Operations:

- + avoidObstacle(): void : this operation explains the behavior that the robots assume to avoid any obstacle.

- + missionComplete(): boolean : this operation is called when a certain condition, specific to the type of robot, is met (e.g. a robot of type **Mapper** has found all the objectives).
- <<Create>> + Robot(): represents the constructor.
- +getMode:String : this represents the function that retrieves the state of the robot.
- +setMode(in null String): setter function for the robot mode.
- +obstacleDetected(in null: void): these functions control the robot whenever an obstacle is detected.
- +itExists(): Boolean: this function checks whether or not a robot has been created.
- +isWorking(): Boolean: this function is used to check whether a robot has encountered any hardware fault.
- +setLocations(in pos: Point3d): void: this function send the locations of the boxes to CentralStation.

Mapper (Subclass of **Robot**): a class that represents one of the two possible types of robots. This type will simply roam around the environment in search of the red boxes and report their position to the central station. It is connected to **Environment** by a shared aggregation 'Search'.

Attributes: it inherits all the attributes of the class **Robot**. Moreover, it has the additional attributes:

- + camera: CameraSensor :this attribute represents the camera used to search for objectives in the environment
- + cameraImage: BufferedImage : a variable that stores the current image recorded by the camera
- +sonars: RangeSensorBelt[1..*] : this attribute represents the instances for sonars sensors of type **RangeSensorBelt**. The number of instances can range from 1 to any other number.

Operations: it inherits all the operations of **Robot.** Has additional operations:

- + targetDetected(): boolean : this operation is used to determine whether or not an objective has been identified by the camera and returns TRUE or FALSE accordingly
- + pinPoint(): point3d : the operation that records the coordinates of the object based on the coordinates of the robot itself.
- *<<Create>>* + Mapper() : constructor for the class **Mapper**
- + sendCoordinates(): void : a function that stores the coordinates in *CentralStation*

Rescuer (Subclass of **Robot**): a class that represents the other type of robots. This type will go to specific points in the environment according to the coordinates recorded by the mappers. It is connected to **MyEnvironment** by a shared aggregation 'Assists'.

Attributes: it inherits all the attributes of **Robot**

- - victimPos: Point3d[*] : a list of locations of the items retrieved by Mapper

and stored by CentralStation
- -myPos: Point3d[1]: the position of the rescuer

Operations: it inherits all the operations of **Robot**. has the additional operation:

- «Create» + Rescuer() : constructor for the class **Rescuer.**
- +setLocations(in copy Point3d): sets a copy of the list of the boxes locations.
- +goAround():void: a function that lets the robot move around the environment.

Environment (subclass of **EnvironmentDescription** - Simbad Class): a class describing the environment where the Robots interact, that extends the Simbad class **EnvironmentDescription**

Operations:

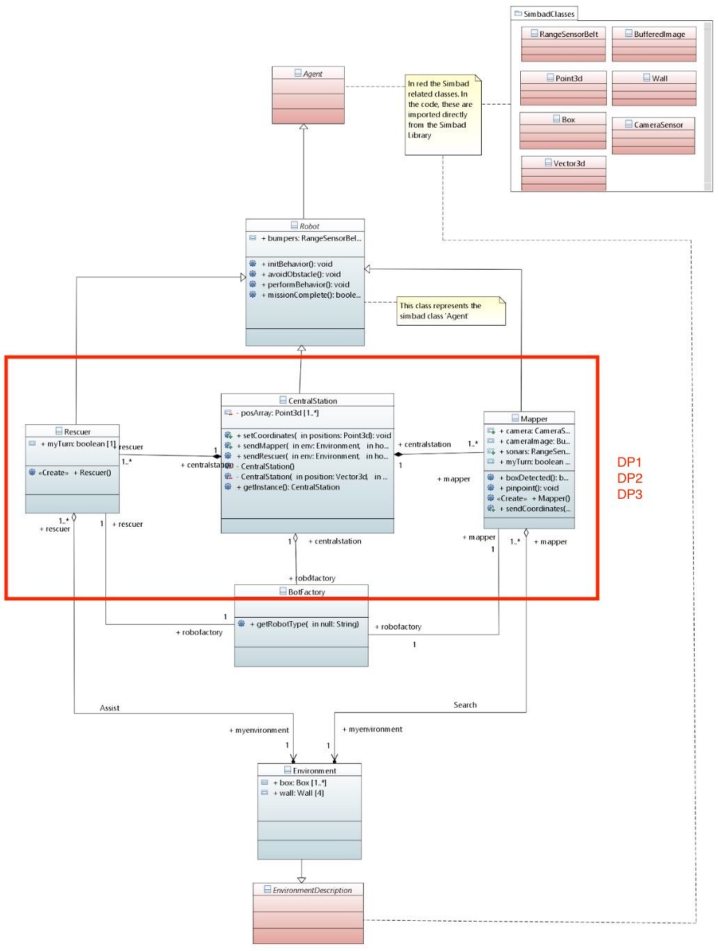
- + addTargets():void : adds the instances of the objects of type **Box.**
- + addWalls:void : adds the four instances of the object of type **Wall**, which are the 4 physical walls that surround the environment.
- addTargets():void: adds the red boxes(targets) to the environment.
- <<Create>> + Environment(): constructor.

In the diagram, in addition to our own classes, we included other classes, identified by the red rectangle, that do not have any kind of specification, rather they are just placeholders for built-in Simbad classes that we could not import in Papyrus. These classes are:

- **Agent**
- **Environment Description**
- **RangeSensorBelt**
- **CameraSensor**
- **BufferedImage**
- **Wall**
- **Box**
- **LampActuator**
- **Point3d** (Java class)
- **Vector3d**(Java class)

2. Applied design patterns

Author(s): Anna Claire Favié and Roshini Ramasamy



ID	DP1
Design pattern	Singleton
Problem	The central station is a centralized body that keeps track of and collects all the information and coordinates of a objectives position and the field. There can be at most one Central station which collects the coordinates at each simulation.
Solution	The central station class is a singleton, this is to make sure that at run time, all the information is stored in only one single object
Intended use	Spawns robots onto the environment, all robots have access at the same time since a static instance of the constructed class central station.
Constraints	Both mappers and rescuers must be referenced to the central station.
Additional remarks	–

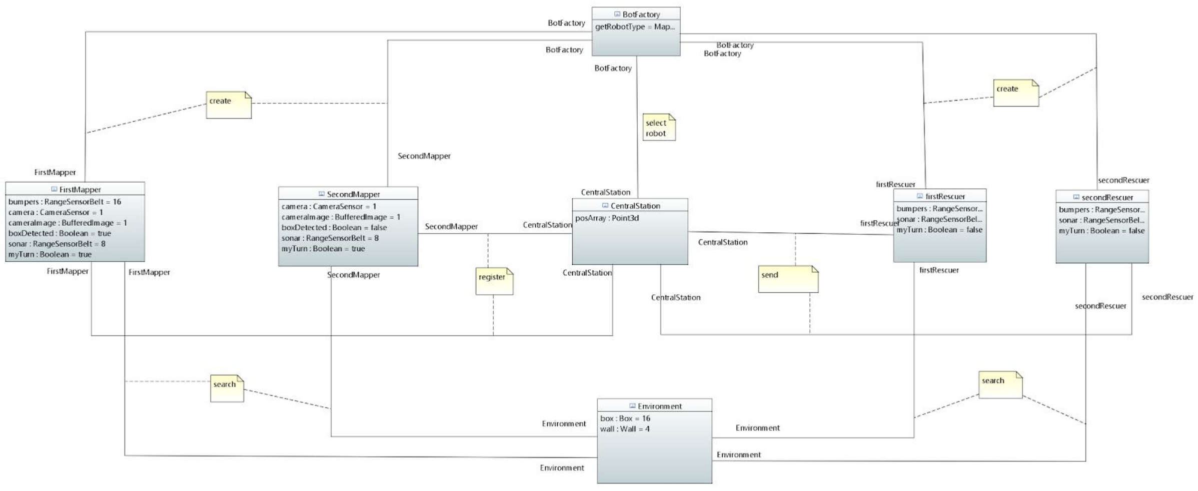
ID	DP2
Design pattern	Factory Method
Problem	Simplify the creation of different objects with a common interface from a different class.
Solution	In Factory Method there has been an object created, this object will be alluded to by using a common class. In our case the object is one of our robots, by using factory method newer

	versions of agents will be easily instantiated by the central station and therefore appended to the existing environment.
Intended use	At any given moment, a class can make use of the BotFactory to create new instances of a particular kind of Robot just by providing its function.
Constraints	In the factory method, you must need to know the tasks performed in order to create relevant instances.
Additional remarks	–

ID	DP3
Design pattern	Observer
Problem	We have two robots of type Mapper and two robots of type Rescuer. The mapper will explore the area and locate the objectives present by using their coordinates and the Rescuers will retrieve the objectives
Solution	Our observer are the rescuers where all the coordinates are received by the Mappers, which are the subjects and are recorded and sent to the central station. The rescuers are robots that are affixed to the objectives. By using setCoordinates(), the central station will store the coordinates and therefore these are available to the rescuers which will now be able to locate the objectives and retrieve them.
Intended use	Since the central station is the observer, it expects the coordinates of the objectives from the mappers. This takes place in the class Central Station.
Constraints	The data being sent to the central station have to be coordinates, of type Point3d.
Additional remarks	–

3. Object diagram

Author(s): Fabiola Loffredi, Carmen Toyas Sánchez



The given Object diagram represents the moment when *centralStation* activates *RoboStation,* which then proceeds to generate two different instances of type **Mapper**: *firstMapper* and *secondMapper*, and two instances of type **Rescuer**. Each robot has 16 bumper sensors (`bumpers: RangeSensorBelt = 16`) and 1 camera (`camera: CameraSensor = 1`), while the **Mapper** Robots also have 8 sonar sensors (`sonar : RAngeSensorBelt = 8`).

The two different types of robot can move in the *environment* only if the variable `myTurn` is set to TRUE: in the case described, only the two *Mapper*s are currently moving (`myTurn : Boolean = true`), while the *Rescuer*s are idle (`myTurn : Boolean = false`).

firstMapper and *secondMapper* to search the *environment* and in the meanwhile, they encounter different obstacles (`box: Box = 16`), which is indicated by `boxDetected : Boolean`. In this case, *firstMapper* encountered an obstacle (`boxDetected : Boolean = true`) and will proceed to avoid it, while *secondMapper* is still looking around (`boxDetected : Boolean = false`).

centralStation awaits for data to be registered, indicated by the empty array `posArray : Point3d = null`.

4. Code generation remarks

Author(s): Cristiano Milanese

The code generation was performed through the Papyrus. We did not encounter any issues in doing so, however, we noticed:

1. As previously mentioned in the class diagram description, we could not import any of the built-in Simbad classes or any particular Java class that was not a basic data type. When the code was generated, the placeholder classes in the diagram where considered as “new” classes, therefore they had their own file to work on. In the implementation phase, we removed these files and moved the generated code that we needed to an existing workspace with all the Simbad Libraries
2. When defining a multiplicity greater than one for the variables for which we needed more than one instance, the code generator would consider them as an array of a given class/type, so we had to manually change those bits of code to make distinct instances.

5. Implementation remarks

Author(s): Cristiano Milanese

With the generated code we tried to improve our system by having two different types of robots: the Mappers, which are in charge of exploring the environment and finding the targets (e.g. victims of a natural disaster) which are represented by red boxes, and the Rescuers, which need to "retrieve" the objectives using the coordinates found by the Mappers. Moreover, we added a CentralStation, which is in charge of organizing/distributing the data among the two robot types, and a BotStation, which generates any of the two types of robot as needed.

The final project wants to expand some concepts that we barely implemented for the moment: Mappers need more efficient algorithms in order to map the environment and Rescuer to reach their targets. Some more time must be spent on setting the accuracy of the locations, playing with spatial coordinates and directions. We met some remarkable difficulties in attaching/detaching agents to the given space, hurdles that brought us the design choice of leaving all robots present in the environment and make them move only when needed. Although complying with the Observer design pattern, our class diagram yielded an implementation that lacks the interfaces needed to make it work.

6. References

References, if needed.