

---

# **SimFleet Documentation**

***Release 1.0.1***

**Javi Palanca**

**Nov 07, 2019**



---

## Contents

---

<b>1</b>	<b>SimFleet</b>	<b>3</b>
1.1	Features . . . . .	3
1.2	Credits . . . . .	3
<b>2</b>	<b>Installation</b>	<b>5</b>
2.1	Stable release . . . . .	5
2.2	From sources . . . . .	5
<b>3</b>	<b>Quickstart</b>	<b>7</b>
3.1	Usage . . . . .	7
3.2	SimFleet entities summary . . . . .	7
3.3	Command-line interface . . . . .	8
3.4	The Config file: Loading Scenarios . . . . .	10
3.5	Graphical User Interface . . . . .	14
<b>4</b>	<b>Developing New Strategies</b>	<b>17</b>
4.1	Introduction . . . . .	18
4.2	Agent Foundations . . . . .	24
4.3	How to Implement your own Strategies . . . . .	27
4.4	How to Implement New Strategies – Recommendations . . . . .	35
<b>5</b>	<b>API Documentation</b>	<b>39</b>
5.1	simfleet package . . . . .	39
<b>6</b>	<b>Contributing</b>	<b>63</b>
6.1	Types of Contributions . . . . .	63
6.2	Get Started! . . . . .	64
6.3	Pull Request Guidelines . . . . .	65
6.4	Tips . . . . .	65
<b>7</b>	<b>Credits</b>	<b>67</b>
7.1	Development Lead . . . . .	67
7.2	Contributors . . . . .	67
<b>8</b>	<b>History</b>	<b>69</b>
8.1	1.0.1 (2019-11-07) . . . . .	69
8.2	1.0.0 (2019-11-05) . . . . .	69

8.3	0.4.1 (2019-01-07) . . . . .	70
8.4	0.4.0 (2018-10-25) . . . . .	70
8.5	0.3.0 (2018-10-01) . . . . .	70
8.6	0.2 (2017-11-15) . . . . .	70
8.7	0.1.3 (2017-11-15) . . . . .	70
8.8	0.1.1 (2017-11-14) . . . . .	70
8.9	0.1.0 (2017-11-03) . . . . .	71
<b>9</b>	<b>Indices and tables</b>	<b>73</b>
	<b>Bibliography</b>	<b>75</b>
	<b>Python Module Index</b>	<b>77</b>
	<b>Index</b>	<b>79</b>

Contents:



Agent-based fleet simulator to test strategies

- Free software: MIT license
- Documentation: <https://simfleet.readthedocs.io>.

## 1.1 Features

- Open Fleets simulator
- Strategy pattern
- Continuous simulator
- Load scenarios
- Multi-agent system built with [SPADE](#)
- XMPP communications

## 1.2 Credits

This package was created with [Cookiecutter](#) and the [audreyr/cookiecutter-pypackage](#) project template.





### 2.1 Stable release

To install SimFleet, run this command in your terminal:

```
$ pip install simfleet
```

This is the preferred method to install SimFleet, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

### 2.2 From sources

The sources for SimFleet can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/javipalanca/simfleet
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/javipalanca/simfleet/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```



### Table of Contents

- *Quickstart*
  - *Usage*
  - *SimFleet entities summary*
    - \* *Description of the Customer Agents*
    - \* *Description of the Transport Agent*
    - \* *Description of the FleetManager Agent*
  - *Command-line interface*
  - *The Config file: Loading Scenarios*
  - *Graphical User Interface*

## 3.1 Usage

Using SimFleet is as easy as running the application in a command line. There are two use modes: a command-line interface and a graphical web-based view. You can run simulations using only the command line or using the easier and intuitive graphical user interface. Running SimFleet without your own developed strategies is possible since the application comes with a set of default strategies. Let's explore how to use both user interfaces.

## 3.2 SimFleet entities summary

In SimFleet there are three types of agent that interact among them during simulations. These are the Fleet Manager agent, the Transport agent, and the Customer agent.

### 3.2.1 Description of the Customer Agents

The Customer agents represent people that need to go from one location of the city (their “current location”) to another (their “destination”) or packages that need to be moved from an origin to a destination. For doing so, each Customer agent requests a single transport service and, once it is transported to its destination, it reaches its final state and ends its execution.

### 3.2.2 Description of the Transport Agent

The Transport agents represent vehicles which can transport Customer agents from their current positions to their respective destinations.

### 3.2.3 Description of the FleetManager Agent

The FleetManager Agent is responsible for putting in contact the Customer agents that need a transport service, and the Transport agents that may be available to offer these services. In short, the FleetManager Agent acts like a transport call center, accepting the incoming requests from customers (Customer agents) and forwarding these requests to the (appropriate) Transport agents. In order to do so, the FleetManager has a registration protocol by which Transport agents subscribe to the Fleet Manager that represents their fleet. This is automatically done when a Transport agent is started.

In the context of SimFleet, a “transport service” involves, once a particular Customer and Transport agents have reached an agreement, the movement of the Transport agent from its current position to the Customer’s position in order to pick the Customer up, and then the transportation of the Customer agent to its destination.

## 3.3 Command-line interface

After installing SimFleet open a command-line and type `simfleet --config config_file.json`. This starts a simulator with the configuration specified at the JSON file and runs the simulator agent. The console will output the default logging information and you can terminate the simulator by pressing Ctrl+C. When you terminate the simulator the results of the simulations are printed.

---

**Hint:** To install an XMPP server visit <https://xmpp.org/software/servers.html> (we recommend [Prosody IM](#))

---

```
$ simfleet --config myconfig.json
2015-10-21 16:29:07.049 | INFO      | simfleet.config:load_config:75 - Reading config_
↳myconfig.json
2015-10-21 16:29:07.062 | INFO      | simfleet.simulator:__init__:71 - Starting_
↳SimFleet (SimFleet)
2015-10-21 16:29:07.064 | INFO      | simfleet.simulator:load_icons:172 - Reading icons
2015-10-21 16:29:07.158 | INFO      | simfleet.directory:setup:40 - Directory agent_
↳running
2015-10-21 16:29:07.159 | INFO      | simfleet.simulator:__init__:91 - Creating 0_
↳managers, 0 transports, 0 customers and 0 stations.
2015-10-21 16:29:07.159 | INFO      | simfleet.simulator:load_scenario:116 - Loading_
↳scenario...
2015-10-21 16:29:07.162 | INFO      | simfleet.route:setup:28 - Route agent running
2015-10-21 16:29:07.162 | WARNING   | simfleet.route:load_cache:74 - Could not load_
↳cache file.
2015-10-21 16:29:07.226 | INFO      | simfleet.simulator:setup:97 - Simulator agent_
↳running
```

(continues on next page)

(continued from previous page)

```
2015-10-21 16:29:07.229 | INFO      | simfleet.simulator:setup:110 - Web interface_
↳running at http://127.0.0.1:9000/app
```

```
^C
```

```
2015-10-21 16:29:21.292 | INFO      | simfleet.simulator:stop:258 -
Terminating... (0.0 seconds elapsed)
Simulation Results
```

```
+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| Simulation Name   | Avg Waiting Time | Avg Total Time | Simulation Time |
↳Max Time | Simulation Finished |
```

```
+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| SimFleet         |                  | 0 |          0 |          0 |
↳ 1000 | False      |
```

```
Fleet Manager stats
```

```
+=====+=====+=====+=====+
| fleet_name | transports_in_fleet | type |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
Customer stats
```

```
+=====+=====+=====+=====+
| name  | waiting_time | total_time | status |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
Transport stats
```

```
+=====+=====+=====+=====+
| name  | assignments | distance | status |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
Station stats
```

```
+=====+=====+=====+=====+
| name  | status  | available_places | power |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
2015-10-21 16:29:21.360 | INFO      | simfleet.simulator:stop:258 -
Terminating... (0.0 seconds elapsed)
Simulation Results
```

```
+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| Simulation Name   | Avg Waiting Time | Avg Total Time | Simulation Time |
↳Max Time | Simulation Finished |
```

```
+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| SimFleet         |                  | 0 |          0 |          0 |
↳ 1000 | False      |
```

```
Manager stats
```

```
+=====+=====+=====+=====+
| fleet_name | transports_in_fleet | type |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
Customer stats
```

```
+=====+=====+=====+=====+
| name  | waiting_time | total_time | status |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
```

```
Transport stats
```

```
+=====+=====+=====+=====+
```

(continues on next page)

(continued from previous page)

```

| name      | assignments | distance  | status    |
+=====+=====+=====+=====+
+=====+=====+=====+=====+
Station stats
+=====+=====+=====+=====+
| name      | status     | available_places | power    |
+=====+=====+=====+=====+
+=====+=====+=====+=====+

```

However, if you don't use some options when running the simulator there will be no default transports nor customers. That's why stats are empty. To run a simulation with some parameters you must fill a configuration file where the simulation scenario is defined.

To show the command line interface options you can enter the `--help` command:

```

$ simfleet --help

Usage: simfleet [OPTIONS]

Console script for SimFleet.

Options:
  -n, --name TEXT           Name of the simulation execution.
  -o, --output TEXT         Filename to save simulation results.
  -of, --oformat [json|excel] Output format used to save simulation results.
                              (default: json)
  -mt, --max-time INTEGER   Maximum simulation time (in seconds).
  -r, --autorun             Run simulation as soon as the agents are ready.
  -c, --config TEXT         Filename of JSON file with initial config.
  -v, --verbose             Show verbose debug level: -v level 1, -vv level
                              2, -vvv level 3, -vvvv level 4
  --help                   Show this message and exit.

```

The output of a simulation shows some statistics of the simulation, with the *Average Total Time*, which represents the average time of customers from the moment they request a transport until they are delivered to their destination, and the *Average Waiting Time*, which is the average time of customers from requesting a transport to being picked up. This information is also shown for each customer along with their status at the end of the simulation.

In the case of transports, the shown information includes the number of assignments of each transport (how many customers it has delivered), the total distance it has traveled and its final status.

This information is going to be useful for the development of new strategies that improve the system balancing or for debugging errors if a transport or a customer gets stuck or any other unexpected situation occurs.

The last but no less important option is the verbosity option. It allows you to specify how verbose you want the simulator to be. The number of `v` letters you pass to the option indicates the level of verbosity (e.g. `-v` is **DEBUG** verbosity and `-vvvv` is the highest level of verbosity where the internal messages of the platform are shown).

### 3.4 The Config file: Loading Scenarios

The ability to load scenarios to SimFleet allows us to repeat the same experiment as many times as we want with the same initial conditions. SimFleet supports to load a *config* file that defines all the fields that you need to load the same information repeatedly. A scenario file must be coded in JSON format.

The most important fields that the scenario file must include are a customers list and a transports list. Each customer must include the following fields:

Customers	
Field	Description
position	Initial coordinates of the customer
destination	Destination coordinates of the customer
name	Name of the customer
password	Password for registering the customer in the platform (optional)
fleet_type	Fleet type that the customer wants to use
icon	Custom icon (in base64 format) to be used by the customer (optional)
strategy	Custom strategy file in the format module.file.Class (optional)

For transports the fields are as follows:

Transports	
Field	Description
position	Initial coordinates of the transport
name	Name of the transport
password	Password for registering the transport in the platform (optional)
speed	Speed of the transport (in meters per second) (optional)
fleet_type	Fleet type that the customer wants to use
fleet	The fleet manager's JID to be subscribed to (optional)
autonomy	The maximum autonomy of the transport (in km) (optional)
current_autonomy	The initial autonomy of the transport (in km) (optional)
icon	Custom icon (in base64 format) to be used by the transport (optional)
strategy	Custom strategy file in the format module.file.Class (optional)

For fleet managers the fields are as follows:

Fleet managers	
Field	Description
position	Initial coordinates of the manager
name	Name of the manager
password	Password for registering the manager in the platform (optional)
fleet_type	Fleet type that the agent manages
icon	Custom icon (in base64 format) to be used by the manager (optional)
strategy	Custom strategy file in the format module.file.Class (optional)

An example of a config file with two customers, two transports and one fleet manager:

```
{
  "fleets": [
    {
      "password": "secret",
      "name": "fleetm1",
      "fleet_type": "drone"
    },
    {
      "password": "secret",
      "name": "fleetm3",
      "fleet_type": "food_delivery"
    },
    {
      "password": "secret",
```

(continues on next page)

(continued from previous page)

```

        "name": "fleetm2",
        "fleet_type": "drone"
    }
],
"transports": [
    {
        "speed": 2000,
        "fleet": "fleetm1@localhost",
        "fleet_type": "drone",
        "position": [40.41192762537371, -3.7105464935302734],
        "password": "secret",
        "name": "drone1"
    },
    {
        "speed": 2000,
        "fleet": "fleetm1@localhost",
        "fleet_type": "drone",
        "position": [40.428655600133546, -3.6993885040283203],
        "password": "secret",
        "name": "drone2"
    },
    {
        "speed": 2000,
        "fleet": "fleetm2@localhost",
        "fleet_type": "drone",
        "position": [40.446424515534666, -3.6612796783447266],
        "password": "secret",
        "name": "drone3"
    },
    {
        "speed": 2000,
        "fleet": "fleetm3@localhost",
        "fleet_type": "food_delivery",
        "position": [40.44635919724081, -3.69140625],
        "password": "secret",
        "name": "bikel"
    },
    {
        "speed": 2000,
        "fleet": "fleetm3@localhost",
        "fleet_type": "food_delivery",
        "position": [40.42035747630869, -3.665142059326172],
        "password": "secret",
        "name": "bike2"
    }
],
"customers": [
    {
        "destination": [40.446163241978304, -3.7075424194335938],
        "position": [40.45171508652634, -3.677501678466797],
        "password": "secret",
        "name": "cl",
        "fleet_type": "drone"
    },
    {
        "destination": [40.4068299938421, -3.670291900634765],
        "position": [40.43087697137461, -3.716297149658203],

```

(continues on next page)



(continued from previous page)

```

        "password": "secret",
        "name": "c2",
        "fleet_type": "drone"
    },
    {
        "destination": [40.43002763221108,-3.6797332763671875],
        "position": [40.45759301026131,-3.664026260375976],
        "password": "secret",
        "name": "c3",
        "fleet_type": "drone"
    },
    {
        "destination": [40.45785423938172,-3.711318969726563],
        "position": [40.440088345478614,-3.680849075317383],
        "password": "secret",
        "name": "f1",
        "fleet_type": "food_delivery"
    },
    {
        "destination": [40.458572614225545,-3.680419921875],
        "position": [40.409770982232956,-3.6928653717041016],
        "password": "secret",
        "name": "f2",
        "fleet_type": "food_delivery"
    }
],
"stations": [
    {
        "name": "station1",
        "password": "secret",
        "position": [40.424559,-3.7002277],
        "places": 2,
        "power": 50,
        "icon": "gas_station"
    }
],
"simulation_name": "Example Config",
"max_time": 1000,
"verbose": 1,
"transport_strategy": "simfleet.strategies.AcceptAlwaysStrategyBehaviour",
"customer_strategy": "simfleet.strategies.AcceptFirstRequestBehaviour",
"fleetmanager_strategy": "simfleet.strategies.DelegateRequestBehaviour",
"directory_strategy": "simfleet.directory.DirectoryStrategyBehaviour",
"station_strategy": "simfleet.station.StationStrategyBehaviour",
"fleetmanager_name": "fleetmanager",
"fleetmanager_password": "fleetmanager_passwd",
"route_name": "route",
"route_password": "route_passwd",
"directory_name": "directory",
"directory_password": "directory_passwd",
"host": "localhost",
"xmpp_port": 5222,
"http_port": 9000,
"http_ip": "127.0.0.1",
"coords": [40.4167754, -3.7037902],
"zoom": 14
}

```

The rest of configuration parameters are referred to general settings of the simulator such as `coords` and `zoom` which allows the user to set up the coordinates and zoom of the city where the simulation is run.

If you want to store the results of simulation in a file you may use the `--output` option (or `-o`) to specify the name of the file where the simulation results will be saved. The `--oformat` (`-of`) allows you to choose the output format between json (default) or excel. It is also useful to use the `--name` (or `-n`) to name the simulation.

Example:

```
$ simfleet --config myconfig.json --name "My Simulation" --output results.xls --  
  ↳oformat excel
```

## 3.5 Graphical User Interface

A much more user-friendly way to use SimFleet is through the built-in graphical user interface. This interface is accessed via any web browser and is designed as a viewer for your running simulations. To open it just visit the address shown on the screen when you run the simulator and access that website.

---

**Hint:** The Simulator agent is who raises the GUI and shows the address in the console output:

```
2015-10-21 16:29:07.229 | INFO          | simfleet.simulator:setup:110 - Web interface_  
  ↳running at http://127.0.0.1:9000/app
```

This address is (in most cases): <http://127.0.0.1:9000/app>

---

Once you visit the GUI address you see an interface like this:

In the GUI you can see a map of the city on the right and a Control Panel with various options on the left:

1. A **Run** button that starts the simulation.
2. A **Clear** button to stop and reset the simulation.
3. Stats of the waiting time and total time of the simulation in real time.
4. A **Download** button to get the stats of the simulation in excel or json format.
5. A collapsable tree view with the transports and customers that are included in the simulation, with a color bullet that indicates their current status.

If the **Run** button is pressed the simulation shows how the transports move to the customers and deliver them to their destinations.

Notice that when a transport picks up a customer, the customer's icon disappears from the map view (since it is inside the transport) and is no longer viewed (it's also not shown when it arrives to its destination). However, you can check at any time your customers status in the tree view of the Control Panel.

The code colors in the tree view indicate the status of a transport or a customer. The legend of colors is as follows:

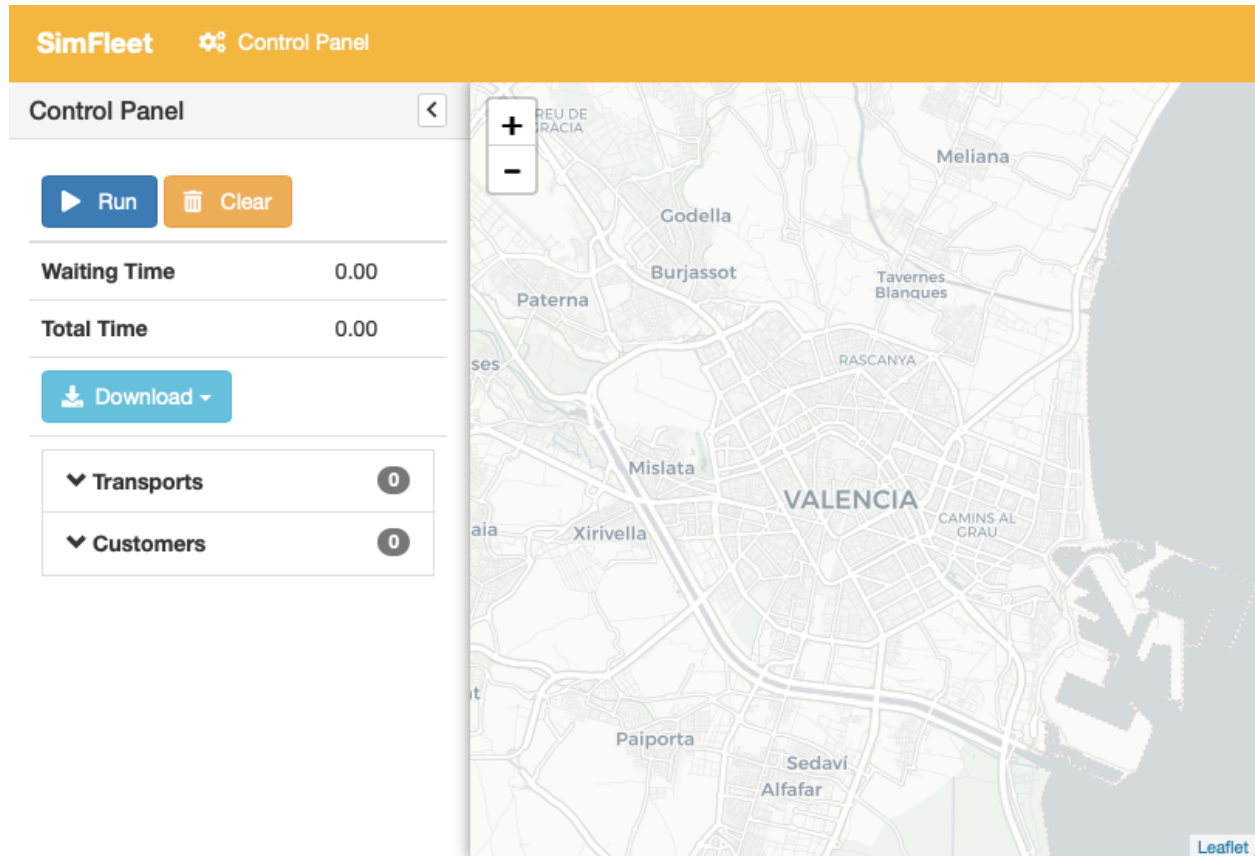


Fig. 1: GUI at startup

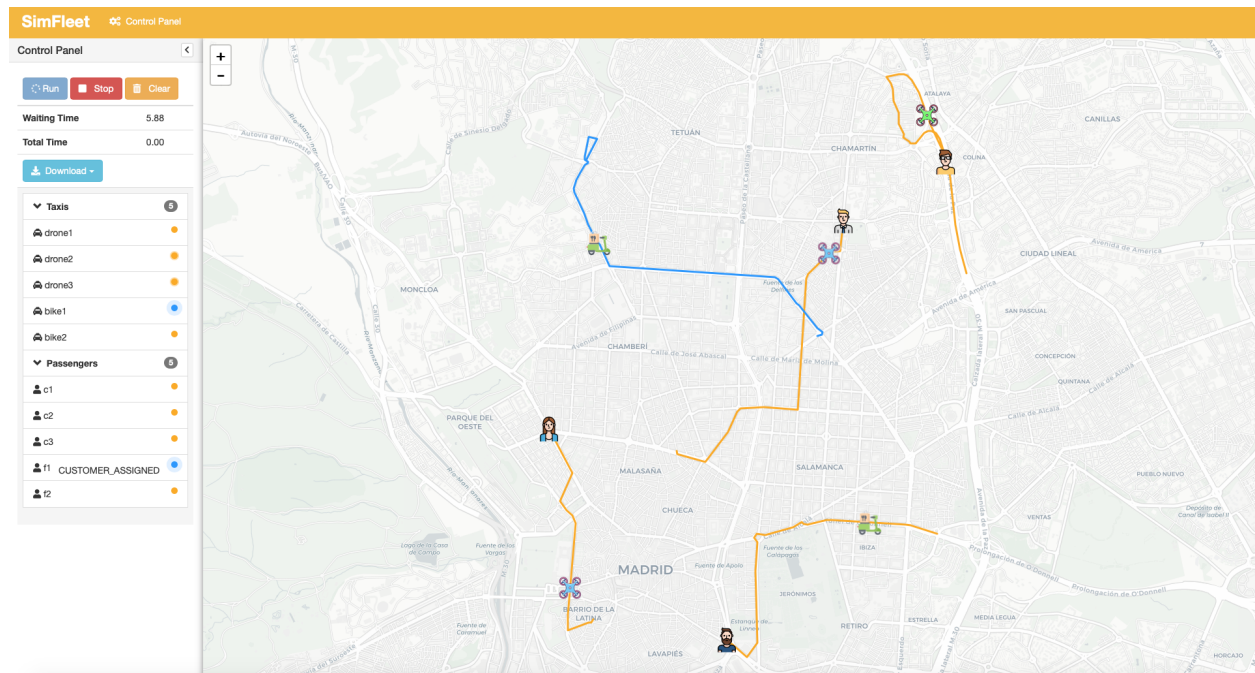










Fig. 2: Simulation in progress

Transports		Customers	
Bullet	Status	Bullet	Status
	WAITING		WAITING
	WAITING FOR APPROVAL		ASSIGNED
	MOVING TO CUSTOMER		IN TRANSPORT
	MOVING TO DESTINATION		IN DESTINATION

---

**Hint:** Every time than a bullet is pulsing means that the agent is moving.

---

When a transport is moving it's also shown in the GUI the path that the transport is following. The color of the path indicates the type of movement that the transport is doing. A yellow path indicates that the transport is going to pick up the customer. On the other hand, a blue path indicates that the transport is taking the customer to his destination.

---

**Note:** A simulation is finished when all transports are free (and waiting for new customers) and all customers are in their destinations (i.e. all bullets are green).

---

---

### Developing New Strategies

---

#### Table of Contents

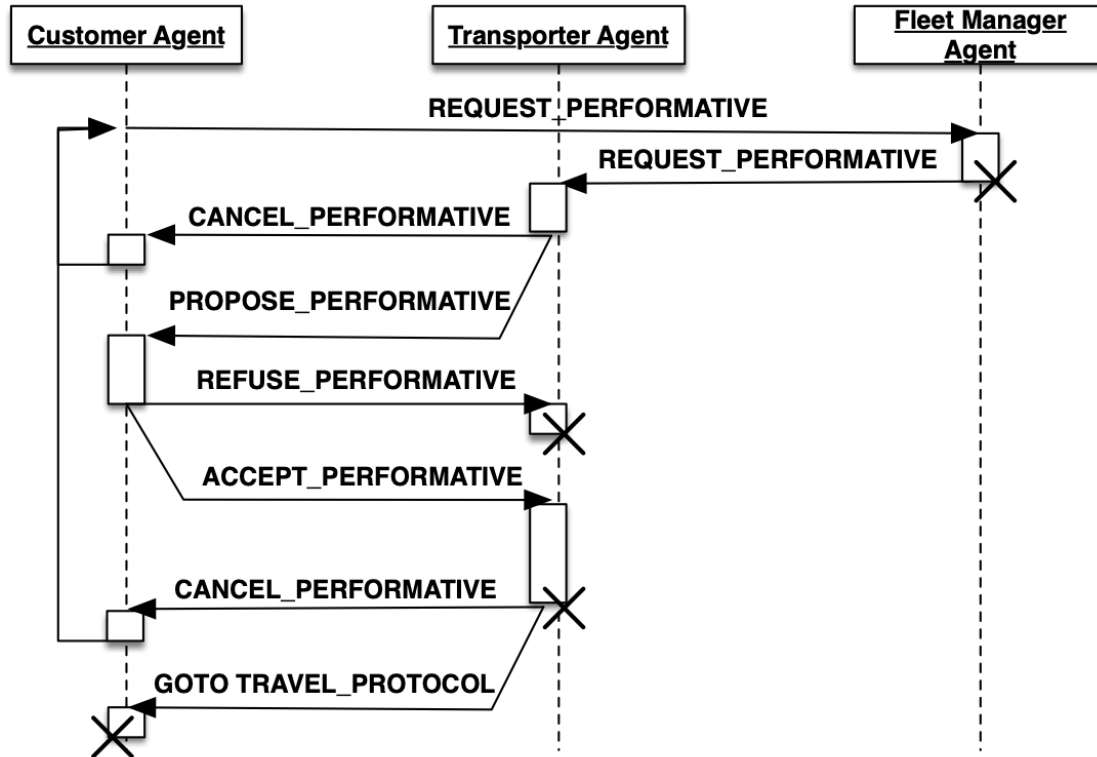
- *Developing New Strategies*
  - *Introduction*
    - \* *Fleet Manager Strategy Behaviour (DelegateRequestBehaviour)*
    - \* *Transport Agents Behaviours*
      - *Strategy Behaviour (AcceptAlwaysStrategyBehaviour)*
      - *Moving Behaviour*
    - \* *Customer Agents Behaviours*
      - *Strategy Behaviour*
      - *Travel Behaviour*
    - \* *The Negotiation Process between Transport and Customer Agents*
  - *Agent Foundations*
    - \* *SPADE*
      - *Agent Model: Behaviours*
      - *Communication API, Messages and Templates*
  - *How to Implement your own Strategies*
    - \* *The Strategy Pattern*
    - \* *The Strategy Behaviour*
      - *Helpers*
    - \* *Developing the FleetManager Agent Strategy*

- *Code*
- *Helpers*
- \* *Developing the Transport Agent Strategy*
  - *Code*
  - *Helpers*
- \* *Developing the Customer Agent Strategy*
  - *Code*
  - *Helpers*
- \* *Other Helpers*
- *How to Implement New Strategies – Recommendations*

## 4.1 Introduction

One of the main features of “SimFleet” is the ability to change the default negotiation strategy of the agents that interact during the simulation: the Fleet Manager agents, the Transport agents and the Customer agents. The overall goal of the negotiation strategy of these three agent types is to decide which Transport agent will transport each Customer agent to its destination, making sure that no Customer agent is left unattended. Additionally, the negotiation strategy may also try to optimize some metrics, such as the average time that Customer agents are waiting to be served, or that the amount of gas spent by Transport in their movements.

The negotiation strategy is based on two main elements. First, it is based on the internal logic of each agent type (FleetManager, Transport and Customer) and, in particular, on their respective *strategy behaviour*, which includes the internal logic of each agent type regarding the negotiation process. And second, it is also based on the so-called *REQUEST* protocol, which comprises the types of messages exchanged among the three agent types during the negotiation. The following diagram presents the protocol in the typical FIPA format, where agents types are depicted as vertical lines and the exchanged message types (or “performatives”) in horizontal arrows:



This chapter introduces first the current, default strategy of each agent type (FleetManager, Transport and Customer) and then explains how to introduce new strategies for any, or all, of them.

#### 4.1.1 Fleet Manager Strategy Behaviour (*DelegateRequestBehaviour*)

The FleetManager Agent includes a single behaviour, which is its strategy behaviour, now described. The goal of the strategy behaviour of the FleetManager Agent is basically to **receive** the “request” messages (*REQUEST\_PERFORMATIVE*) sent by the Customer agents that need a transport service and, for each request, selecting the Transport agent, or agents, that may perform the service, and **forward** the request to them. A *REQUEST\_PERFORMATIVE* message includes the following fields:

```

"customer_id": Id of the Customer agent that performs the request.
"origin":      Current position of the Customer, where the Transport has to pick it_
               ↪ up.
"dest":        Destination of the Customer, where the Transport needs to transport_
               ↪ it.
  
```

The particular set of Transport agents to which the request will be forwarded depends on the *allocation policy* of the FleetManager Agent, which is part of the strategy. In the default strategy behaviour for the FleetManager agent (*DelegateRequestBehaviour*), the allocation policy is the simplest possible: it forwards every incoming request to **all** the Transport agents that are registered in its fleet, regardless of their current statuses or any other consideration (such as, for example, the last time they performed a service, or the distance between them and the Customer agent).

In the default strategy behaviour, the set of incoming messages that may be delivered to the FleetManager Agent is reduced to the requests made by Customer agents, and the behaviour itself does not include multiple states. So, each incoming message is processed in the same way, and leaves the behaviour in the same (unique) state.

Once each request has been forwarded to some (or all) the Transport agents, the goal of the FleetManager Agent for that request is achieved. This is the starting point to the negotiation between the Customer that has issued the request and the Transport agents that have received it, which is described in the following sections.

### 4.1.2 Transport Agents Behaviours

Transport agents incorporate two behaviours: the strategy behaviour and the moving behaviour, now described.

#### Strategy Behaviour (*AcceptAlwaysStrategyBehaviour*)

The goal of the strategy behaviour of a Transport agent is to negotiate with Customer agents which are requesting a transport service the conditions of the service offered by the Transport, in order to achieve an agreement with these Customer agents. When an agreement is reached between a particular Customer and Transport agents, then the Transport agent picks up the Customer agent and transport it to its destination (and starts the Moving behaviour, described below).

The currently implemented, default strategy behaviour is called *AcceptAlwaysStrategyBehaviour*, and has a direct relation with the *REQUEST* protocol explained above. In particular, the behaviour can be thought of as a finite-state machine with some different states specifying the statuses of the Transport agent regarding the strategy behaviour, and some transitions between states, which are triggered either by messages (of the *REQUEST* protocol) received by the Transport agent, or by some other program conditions. This is depicted in the following diagram:

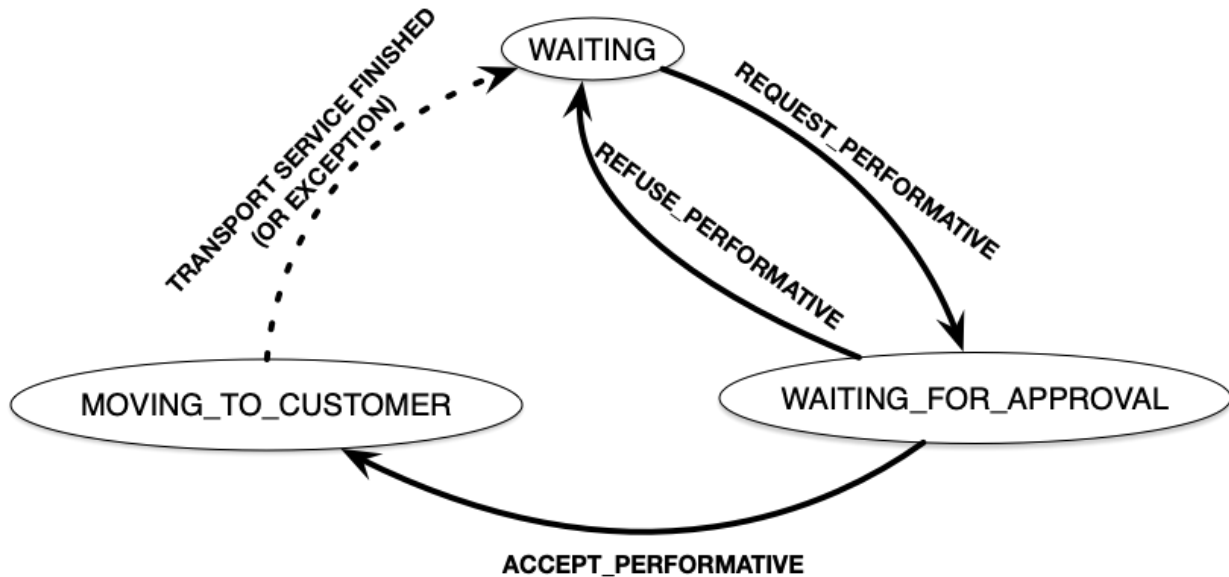


Fig. 1: States and transitions of the strategy behaviour of a Transport agent.

The semantics of each state are now described:

- *TRANSPORT\_WAITING*: In this state, the Transport agent is available (free) and waiting for requests from Customer agents. While in this state, if it receives a request message (*REQUEST\_PERFORMATIVE*) from a particular Customer agent, it will send the Customer a service proposal (*PROPOSE\_PERFORMATIVE*) and it will change its state to *TRANSPORT\_WAITING\_FOR\_APPROVAL*.
- *TRANSPORT\_WAITING\_FOR\_APPROVAL*: In this state, the Transport agent is waiting for the response message from a Customer agent to which it has sent a service proposal message. While in this state, it may receive two alternative answers from the Customer agent: (1) the Customer refuses the service proposal (*REFUSE\_PERFORMATIVE*), in which case the Transport changes its state back to *TRANSPORT\_WAITING*; or (2) the Customer accepts the proposal (*ACCEPT\_PERFORMATIVE*), in which case it will change to the state *TRANSPORT\_MOVING\_TO\_CUSTOMER*.



- *TRANSPORT\_MOVING\_TO\_CUSTOMER*: In this state, the Transport agent and the Customer agent have agreed to perform a transport service, and then the Transport agent starts to travel to the Customer location in order to pick it up. This is the final state of the negotiation between the Transport and a certain Customer agent. When the Transport agent arrives to the Customer's position, the Transport agent executes the helper function *pick\_up\_customer*, which automatically starts the so-called Moving behaviour in the Transport agent, described below. It also sends a message to the Travel behaviour of the Customer agent, which starts that behaviour (this is explained in the next section).

## Moving Behaviour

This behaviour makes the Transport agent to move to the current location of the Customer agent with which it has reached an agreement to perform a transport service. After picking the Customer agent up, the Transport will then transport it to its destination. During that travel, the behaviour informs the Customer agent of where the Transport is and what it is doing (going to pick up the Customer, taking the Customer to its destination, reaching the destination, etc.). All this is performed by sending the Customer agent some messages which belong of another, dedicated protocol called *TRAVEL\_PROTOCOL*.

Once the Transport reaches the Customer agent's destination and the Customer agent is informed about it, the state of the Transport agent (of the strategy behaviour) is here changed to *TRANSPORT\_WAITING*, indicating that it is now free, and hence making the Transport agent available again to receiving new requests from other Customer agents.

**Warning:** This behaviour is internal and automatic, and it is not intended to be modified while developing new negotiation strategies. The same applies to the *TRAVEL\_PROTOCOL* protocol.

## 4.1.3 Customer Agents Behaviours

Customer agents incorporate two behaviours: the strategy behaviour and the travel behaviour, now described.

### Strategy Behaviour

In the course of the *REQUEST* protocol, the request of a transport service made by a Customer agent is answered by one (or several) Transport agents, each of which offering the Customer their conditions to perform such service. The goal of the strategy behaviour of a Customer agent is to select the best of these transport service proposals, according to its needs and/or preferences (e.g., to be picked up faster, to get the nearest available transport, to get the cheapest service, etc.).

The currently implemented default strategy behaviour is called *AcceptFirstRequestBehaviour*. As in the strategy behaviour of the Transport agents above, here we can also consider the strategy as a finite-state machine related to the messages (of the *REQUEST* protocol) received by the Customer agent, as depicted below:

The semantics of each state are now described:

- *CUSTOMER\_WAITING*: In this state, the Customer agent requires a transport service and, periodically, sends a request for that service until one (or many) Transport agent proposals (*PROPOSE\_PERFORMATIVE*) are received. When the Customer accepts a particular proposal (in the current implementation, always the first one it receives while in this state) then it communicates so to the proposing Transport agent, and changes its own status to *CUSTOMER\_ASSIGNED*.
- *CUSTOMER\_ASSIGNED*: In this state, the Customer agent has been assigned to a particular transport, and the transport service is being performed. The Customer side of the transport service is implemented by activating the Travel behaviour, described below, which is started by a message sent by the Transport agent (in its helper function *pick\_up\_customer*). If something goes wrong (for example, an exception is raised during the transport service) or the Transport agent voluntarily wants to cancel the service, then the Transport agent

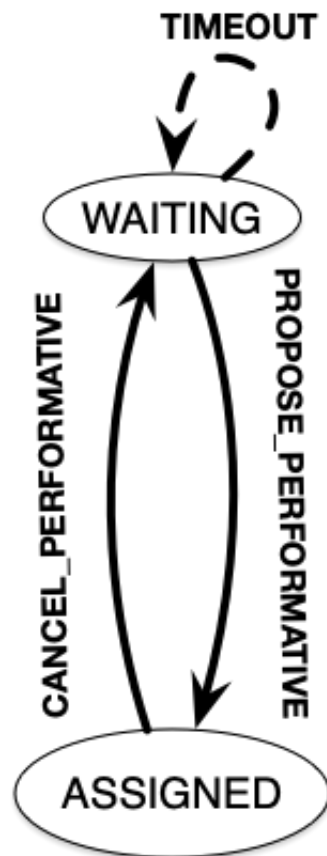


Fig. 2: States and transitions of the strategy behaviour of a Customer agent.

sends a *CANCEL\_PERFORMATIVE* to the Customer agent, which would then change its status back to *CUSTOMER\_WAITING*, initiating the request process again.

## Travel Behaviour

This behaviour is activated (in the Customer agent) when a Transport agent decides to pick up the Customer agent, by means of a message sent by the Transport (inside the Transport agent's helper function *pick\_up\_customer*). This message, as well as other messages sent by the Transport agent to this behaviour, belongs to a protocol called *TRAVEL\_PROTOCOL*.

The messages of the *TRAVEL\_PROTOCOL* drive the transitions between the different states of this behaviour, in the same way that the *REQUEST\_PROTOCOL* does for the strategy behaviour. In particular, the states of this behaviour are: *CUSTOMER\_IN\_TRANSPORT*, when the Transport agent has reached the Customer agent's position and has picked it up; and *CUSTOMER\_IN\_DEST*, when the Transport agent has reached the Customer agent's destination. This would be the final state of the Customer agent.

**Warning:** This behaviour is internal and automatic, and it is not intended to be modified while developing new negotiation strategies. The same applies to the *TRAVEL\_PROTOCOL* protocol.

### 4.1.4 The Negotiation Process between Transport and Customer Agents

After separately explaining the strategy behaviour of Transport and Customer agents, this section tries to relate both behaviours. This is important to understand how these two agent types interact with each other in order to coordinate and reach the overall goals of the simulation.

In particular, there are three key aspects (embedded within the strategy behaviours) which influence the overall coordination process implemented in the simulator, as now described:

- The conditions of a transport service proposal. The current implementation does not consider any special condition other than the Transport agent being free (available to perform the service). Some aspects that could be included in a transport proposal would be, for example, the current location of the transport, the proposed fare, the route to take the Customer agent to its destination, etc.
- The preferences of customers in order to select a particular transport proposal. In the current implementation, the Customer agents always accept the first proposal received from a Transport agent. In a more sophisticated negotiation, some internal goals/conditions of the Customer agent could be taken into account in order to select a "better" proposal. These might include, for example, the expected waiting time until the Transport agent arrives, the amount of money that the service is expected to cost, the brand of the Transport vehicle, etc.
- The possibility of a transport to voluntarily cancel an ongoing transport service after a proposal has been accepted by a customer. This may happen only before the customer has been picked up, that is, while the transport is moving from its initial position to the location where the customer is waiting for it. In the current implementation, a transport service cancellation can only be produced if some exception is raised while the service is being produced (for example, if the software calculating a route for the Transport agent fails to produce a valid route). Since new Customer (and maybe Transport) agents can appear at any time while the simulation is running, a voluntary cancellation of transport services could improve the overall transportation of customers throughout the simulation, allowing for a "dynamic reallocation" of customers to transports, even when transport services were already committed.

## 4.2 Agent Foundations

The architecture of SimFleet is built on top of a multi-agent system platform called SPADE. Although it is not necessary to build new agents in order to develop new coordination strategies (the simulator provides all the necessary agents), it is interesting to know how they work and what methods they provide for the creation of coordination strategies.

Next we will present the SPADE platform and its main features. For more documentation you can visit their website <https://github.com/javipalanca/spade>.

### 4.2.1 SPADE

*SPADE* (Smart Python multi-Agent Development Environment) is a multi-agent system (MAS) platform based on the *XMPP* technology and written in the *Python* programming language. This technology offers by itself many features and facilities that ease the construction of MAS, such as an existing communication channel, the concepts of users (agents) and servers (platforms) and an extensible communication protocol based on XML.

Extensible Messaging and Presence Protocol (XMPP) is an open, XML-inspired protocol for near-real-time, extensible instant messaging (IM) and presence information. The protocol is built to be open and free, asynchronous, decentralized, secure, extensible and flexible. The latter two features allow XMPP not only to be an instant messaging protocol, but also to be extended and used for many tasks and situations (*IoT*, *WebRTC*, *social*, ...). SPADE itself uses some XMPP extensions to provide extended features to its agents, such as remote procedure calls between agents (*Jabber-RPC*), file transfer (*In-Band Bytestreams*), and so on.

In order to fully understand how SPADE works, it is necessary to know how the agents are made up and how they communicate. In the following sections we will summarize the SPADE agent model and its communication API.

#### Agent Model: Behaviours

SPADE agents are threaded-based objects that can be run concurrently and that are connected to a SPADE platform, which internally runs an XMPP server. Each agent must provide an ID and password in order to be allowed to connect to the platform. The agent ID is called JID and has the form of an email: a user name string plus a “@” character plus the IP address of the SPADE server to connect to (e.g. *my\_agent@127.0.0.1*).

The internal components of the SPADE agents that provide their intelligence are the **Behaviours**. A behaviour is a task that an agent can run using some pre-defined repeating pattern. For example, the most basic behaviour type (pattern) is the so-called cyclic behaviour, which repeatedly executes the same method over and over again, indefinitely. This is the way to develop typical behaviours that wait for a perception, reason about it and finally execute an action, and then wait for the next perception.

The following example is a sample of an agent with a cyclic behaviour (*spade.behaviour.CyclicBehaviour* type) that waits for a perception from the keyboard input, reasons on it and executes an action, and continues to do so indefinitely until the user presses Ctrl+C. In order to build a behaviour, you need to inherit from the type of behaviour you want (in the case of this example, the cyclic behaviour is implemented in the class *spade.behaviour.CyclicBehaviour*) and overload the coroutine *run* where the body of the behaviour is implemented. If needed, you can also overload the *on\_start* and *on\_end* coroutines in order to execute actions on the initialization or shutdown of a behaviour, respectively.

```
import spade
import datetime
import time

class MyAgent(spade.agent.Agent):
    class MyBehaviour(spade.behaviour.CyclicBehaviour):
```

(continues on next page)

(continued from previous page)

```

    async def on_start(self):
        print("Initialization of behaviour")

    async def run(self):
        # wait for perception, raw_input is a blocking call
        perception = raw_input("What's your birthday year?")
        # reason about the perception
        age = datetime.datetime.now().year - perception
        # execute an action
        print("You are {age} years old.".format(age=age))

    async def on_end(self):
        print("Shutdown of behaviour")

    async def setup(self):
        # Create behaviour
        behaviour = self.MyBehaviour()
        # Register behaviour in agent
        self.add_behaviour(behaviour)

if __name__ == "__main__":
    a = MyAgent(jid="agent@127.0.0.1", password="secret")
    a.start()
    while True:
        try:
            time.sleep(1)
        except KeyboardInterrupt:
            break
    a.stop()

```

Along with the cyclic repeating pattern (or type), SPADE also provides several other types of behaviours, such as like one-shot behaviours, periodic behaviours, finite-state machine behaviours, etc. It is important to note that SPADE agents can execute many behaviours simultaneously, from the same or different types.

## Communication API, Messages and Templates

Communication is one of the cornerstones of any multi-agent system, and SPADE is no exception. Agents can send and receive messages using a simple API, and more importantly, they can receive them in certain behaviours according to templates they can define.

A `spade.message.Message` is the class that needs to be filled in order to send a message. A Message may be filled with several pieces of information, but the most important fields are the receiver, the content, the performative and the protocol. The receiver must be filled with a *jid* address, which is a string. The content is the (string-based) body of the message. The performative and protocol both add semantic information to the message in the context of a conversation: they are normally used to represent the action and the rules that determine how the agents are going to communicate in a specific semantic context and they are represented as metadata.

---

**Tip:** It is usually recommended to use a representation language for the content of the message. Although semantic languages like OWL or RDF are normally used for this purpose, in this simulator JSON is used instead, for the sake of simplicity.

---

All these fields have a getter and setter function. An example is shown next:

```
import spade

msg = spade.message.Message()
msg.to = "receiver_agent@127.0.0.1"
msg.set_metadata("performative", "request")
msg.set_metadata("protocol", "my_custom_protocol")
msg.body = '{"a_key': 'a_value'}"
```

---

**Hint:** Other metadata fields that can be filled in the message are the content language, the ontology, and so on.

---

The next step is to send the message. This is done with the `send` coroutine provided by a *Behaviour*. For example:

```
import spade

class SenderAgent(spade.agent.Agent):
    class SendBehav(spade.behaviour.OneShotBehaviour):

        async def run(self):
            msg = spade.message.Message()
            msg.to = "receiver@127.0.0.1"
            msg.set_metadata("performative", "inform")
            msg.set_metadata("ontology", "myOntology")
            msg.set_metadata("language", "OWL-S")
            msg.body = "Hello World"

            await self.send(msg) # send the message

    async def setup(self):
        print "MyAgent starting..."
        behav = self.SendBehav()
        self.add_behaviour(behav)
```

The reception of messages is particular in SPADE, since messages can only be received by behaviours, and so SPADE provides each behaviour executed by any agent with its own mailbox, and defines a mechanism in order to configure the particular behaviour that must receive each message, according to the message type. This mechanism is carried out with *Templates*. When an agent receives a new message it checks if the message matches each of the behaviours using a template with which they were registered. If there is a match, the message is delivered to the mailbox of the corresponding behaviour, and will be read when the behaviour executes the `receive` method. Otherwise, the message will be dropped.

---

**Note:** The `receive` coroutine accepts an optional parameter: **timeout=seconds**, which allows the coroutine to be blocking until the specified number of seconds have elapsed. If the timeout is reached without a message being received, then `None` is returned. If the timeout is set to 0, then the `receive()` function is non-blocking and (immediately) returns either a `spade.message.Message` or `None`.

---

A `spade.template.Template` is created using the same API of `spade.message.Message`:

```
import spade

template = spade.template.Template()
template.set_metadata("ontology", "myOntology")
```

---

**Note:** A `spade.template.Template` accepts boolean operators to combine *Templates* (e.g. `my_tpl =`

---

```
Template( template1 & template2))
```

---

At this point we can present a full example on how to build an agent that registers a behaviour with a template and receives messages that match that template:

```
import spade
import asyncio

class RecvAgent(spade.agent.Agent):
    class ReceiveBehav(spade.behaviour.CyclicBehaviour):

        async def run(self):
            await msg = self.receive(timeout=10)

            # Check wether the message arrived
            if msg is not None:
                assert "myOntology" == msg.get_metadata("ontology")
                print("I got a message with the ontology 'myOntology'")
            else:
                print("I waited 10 seconds but got no message")

    async def setup(self):
        recv_behav = self.ReceiveBehav()
        template = spade.template.Template()
        template.set_metadata("ontology", "myOntology")

        self.add_behaviour(recv_behav, template)
```

These are the basics of SPADE programming. You will not need to create all these structures, templates and classes in order to use *SimFleet*, but it is always better to know the foundations before getting down to business.

## 4.3 How to Implement your own Strategies

SimFleet is designed for users to implement and test new strategies that lead to system optimization. The goal of this simulator is to make it easier for users to work with new coordination strategies without having to introduce major modifications to the application. For this purpose, SimFleet incorporates the so-called Strategy design pattern, which is now introduced.

### 4.3.1 The Strategy Pattern

The **Strategy Pattern** is a design pattern that enables selecting an algorithm at runtime. The Strategy Pattern is the best practice when an application incorporates different, alternative versions of an algorithm and we want to be able to select any of these versions to be executed at run time. With this pattern, you can define a separate strategy (implementation of the algorithm) in an object that encapsulates the algorithm. The application that executes the algorithm **must** define an interface that every strategy (implementation) will follow, as it can be seen in the following figure:

Following this implementation, the context object can call the current strategy implementation without knowing how the algorithm was implemented. This design pattern was created, among others, by a group of authors commonly known as the **Gang of Four** (E. Gamma, R. Helm, R. Johnson and J. Vlissides), and it is well presented in [GangOfFour95].

SimFleet uses the *Strategy Pattern* in order to enable users to implement three different strategies (one for the fleet manager agent, one for the transport agent and one for the customer agent) without having to develop new agents or

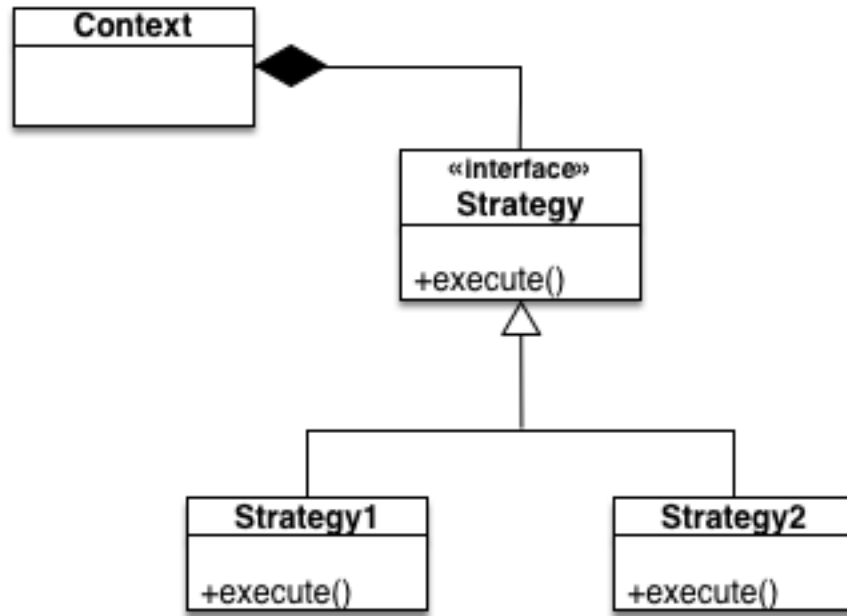


Fig. 3: The Strategy Pattern UML.

entering in the complexity of the simulator. Thanks to this pattern, users can develop their strategies in an external file and pass it as an argument when the simulator is run.

SimFleet implements one interface for each of these three agents, with each interface also providing some helper functions that intend to facilitate the most common actions of each (subclassed) agent. These three interfaces inherit from the `StrategyBehaviour` class and are called: `FleetManagerStrategyBehaviour`, `TransportStrategyBehaviour` and `CustomerStrategyBehaviour`.

### 4.3.2 The Strategy Behaviour

The `StrategyBehaviour` is the metaclass from which interfaces are created for the strategies of each agent in the simulator. It inherits from a `spade.behaviour.CyclicBehaviour` class, so when implementing it, you will have to overload the `run` coroutine that will run cyclically (and endlessly), until the agent stops.

#### Helpers

The Strategy Behaviour provides also some helper functions that are useful in general for any kind of agent in the simulator.

**Danger:** Don't store information in the Behaviour itself since it is a cyclic behaviour and is run by calling repeatedly the `run` coroutine, so the context of the function is not persistent. Use the agent variable that is accessible from any behaviour as `self.agent`. (i.e. you can do `self.agent.set("my_key", "my_value")` and `self.agent.get("my_key")`).

The `set` and `get` functions allow to store persistent information in the agent and to recover it at any moment. The store uses a *key-value* interface to store custom-defined data.



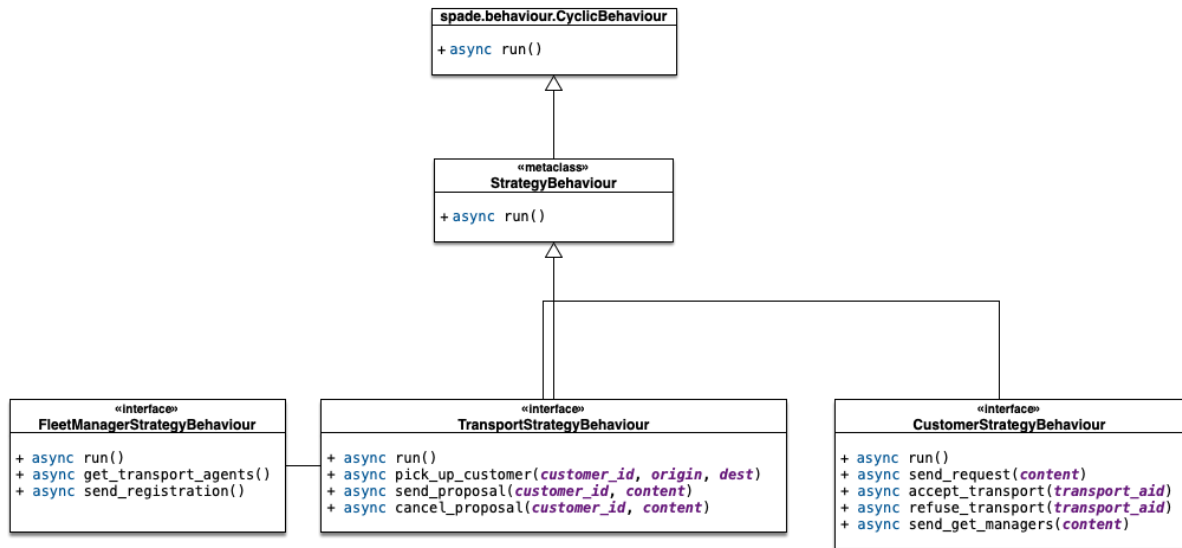


Fig. 4: The StrategyBehaviour class and their inherited interfaces.

There is also a very useful helper function which is the **logger**. This is not a single function but a system of logs which can be used to generate debug information at different levels. There are five levels of logging which are now presented, in order of importance:

- **DEBUG** Used with `logger.debug("my debug message")`. These messages are only shown when the simulator is called with the `-v` option. This is usually superfluous information.
- **INFO** Used with `logger.info("my info message")`. These messages are always shown and are the regular information shown in logs.
- **WARNING** Used with `logger.warn("my warning message")`. These messages are always shown and are used to show warnings to the user.
- **ERROR** Used with `logger.error("my error message")`. These messages are always shown and are used to show errors to the user.
- **SUCCESS** Used with `logger.success("my success message")`. These messages are always shown and are used to show success messages to the user.

In order to use this logger just remember to import the `loguru` library as follows:

```
from loguru import logger
```

### 4.3.3 Developing the FleetManager Agent Strategy

In order to develop a new strategy for the FleetManager Agent, you need to create a class that inherits from `FleetManagerStrategyBehaviour`. Since this is a cyclic behaviour class that follows the *Strategy Pattern* and that inherits from the `StrategyBehaviour`, it has all the previously presented helper functions for communication and storing data inside the agent.

Following the *REQUEST* protocol, the FleetManager agent is supposed to receive every request for a transport service from customers and to carry out the action that your strategy determines (note that, in the default strategy `DelegateRequestBehaviour`, the fleet manager delegates the decision to the transports themselves by redirecting all requests to all their registered transports without any previous, additional reasoning). The code of the `DelegateRequestBehaviour` is presented below.

The place in the code where your fleet manager strategy must be coded is the `run` coroutine. This function is executed in an infinite loop until the agent stops. In addition, you may also overload the `on_start` and the `on_end` coroutines, in order to execute code before the creation of the strategy or after its destruction, if needed.

## Code

This is the code of the default fleet manager strategy `DelegateRequestBehaviour`:

```
from simfleet.fleetmanager import FleetManagerStrategyBehaviour

async def run(self):
    if not self.agent.registration:
        # Register into Directory Agent to make your fleet public
        await self.send_registration()

    msg = await self.receive(timeout=5)
    logger.debug("Manager received message: {}".format(msg))
    if msg:
        # Redirect request to all your registered transports
        for transport in self.get_transport_agents().values():
            msg.to = str(transport["jid"])
            logger.debug("Manager sent request to transport {}".format(transport["name"]
➞))
            await self.send(msg)
```

## Helpers

The fleet manager agent incorporates two helper functions:

- `send_registration`  
Registers its fleet in the Directory agent. This way customers can find their fleet and request for services.
- `get_transport_agents`  
Returns a list of the transports that are registered in that fleet.

### 4.3.4 Developing the Transport Agent Strategy

To develop a new strategy for the Transport Agent, you need to create a class that inherits from `TransportStrategyBehaviour`. Since this is a cyclic behaviour class that follows the *Strategy Pattern* and that inherits from the `StrategyBehaviour`, it has all the previously presented helper functions for communication and storing data inside the agent.

The transport strategy is intended to receive requests from customers, forwarded by its fleet manager agent, and then to send proposals to these customers in order to be selected by the corresponding customer. If a transport proposal is accepted, then the transport begins the process of going to the customer's current position, picking the customer up, and taking the customer to the requested destination.

**Warning:** The process that implies a transport movement is out of the scope of the strategy and should not be addressed by the strategy implementation. This customer-transfer process is automatically triggered when the strategy executes the helper coroutine `pick_up_customer` (which is supposed to be the last action of a transport strategy).

The place in the code where your transport strategy must be coded is the `run` coroutine. This function is executed in an infinite loop until the agent stops. In addition, you may also overload the `on_start` and the `on_end` coroutines, in order to execute code before the creation of the strategy or after its destruction, if needed.

## Code

The default strategy of a transport is to accept any customers' request if the transport is not assigned to any other customer or waiting a confirmation from any customer. This is the code of the default transport strategy `AcceptAlwaysStrategyBehaviour`:

```
from simfleet.transport import TransportStrategyBehaviour

class AcceptAlwaysStrategyBehaviour(TransportStrategyBehaviour):
    async def run(self):
        if self.agent.needs_charging():
            if self.agent.stations is None or len(self.agent.stations) < 1:
                logger.warning("Transport {} looking for a station.".format(self.
↪agent.name))
                await self.send_get_stations()
            else:
                station = random.choice(list(self.agent.stations.keys()))
                logger.info("Transport {} reserving station {}".format(self.agent.
↪name, station))
                await self.send_proposal(station)
                self.agent.status = TRANSPORT_WAITING_FOR_STATION_APPROVAL

        msg = await self.receive(timeout=5)
        if not msg:
            return
        logger.debug("Transport received message: {}".format(msg))
        try:
            content = json.loads(msg.body)
        except TypeError:
            content = {}

        performative = msg.get_metadata("performative")
        protocol = msg.get_metadata("protocol")

        if protocol == QUERY_PROTOCOL:
            if performative == INFORM_PERFORMATIVE:
                self.agent.stations = content
                logger.info("Got list of current stations: {}".format(list(self.agent.
↪stations.keys())))
            elif performative == CANCEL_PERFORMATIVE:
                logger.info("Cancellation of request for stations information.")

        elif protocol == REQUEST_PROTOCOL:
            logger.debug("Transport {} received request protocol from customer/
↪station.".format(self.agent.name))

            if performative == REQUEST_PERFORMATIVE:
                if self.agent.status == TRANSPORT_WAITING:
                    if not self.has_enough_autonomy(content["origin"], content["dest
↪"]):
                        await self.cancel_proposal(content["customer_id"])
                        self.agent.status = TRANSPORT_NEEDS_CHARGING
                else:
```

(continues on next page)

(continued from previous page)

```

        await self.send_proposal(content["customer_id"], {})
        self.agent.status = TRANSPORT_WAITING_FOR_APPROVAL

    elif performative == ACCEPT_PERFORMATIVE:
        if self.agent.status == TRANSPORT_WAITING_FOR_APPROVAL:
            logger.debug("Transport {} got accept from {}".format(self.agent.
↪name,
                                                                    content[
↪"customer_id"])))
            try:
                self.agent.status = TRANSPORT_MOVING_TO_CUSTOMER
                await self.pick_up_customer(content["customer_id"], content[
↪"origin"], content["dest"])
            except PathRequestException:
                logger.error("Transport {} could not get a path to customer {}".
↪. Cancelling..."
                                                                    .format(self.agent.name, content["customer_id"]))
                self.agent.status = TRANSPORT_WAITING
                await self.cancel_proposal(content["customer_id"])
            except Exception as e:
                logger.error("Unexpected error in transport {}: {}".
↪format(self.agent.name, e))
                await self.cancel_proposal(content["customer_id"])
                self.agent.status = TRANSPORT_WAITING
        else:
            await self.cancel_proposal(content["customer_id"])

    elif performative == REFUSE_PERFORMATIVE:
        logger.debug("Transport {} got refusal from customer/station".
↪format(self.agent.name))
        self.agent.status = TRANSPORT_WAITING

    elif performative == INFORM_PERFORMATIVE:
        if self.agent.status == TRANSPORT_WAITING_FOR_STATION_APPROVAL:
            logger.info("Transport {} got accept from station {}".format(self.
↪agent.name,
                                                                    ↪
↪content["station_id"]))
            try:
                self.agent.status = TRANSPORT_MOVING_TO_STATION
                await self.send_confirmation_travel(content["station_id"])
                await self.go_to_the_station(content["station_id"], content[
↪"dest"])
            except PathRequestException:
                logger.error("Transport {} could not get a path to station {}".
↪. Cancelling..."
                                                                    .format(self.agent.name, content["station_id"]))
                self.agent.status = TRANSPORT_WAITING
                await self.cancel_proposal(content["station_id"])
            except Exception as e:
                logger.error("Unexpected error in transport {}: {}".
↪format(self.agent.name, e))
                await self.cancel_proposal(content["station_id"])
                self.agent.status = TRANSPORT_WAITING
        elif self.agent.status == TRANSPORT_CHARGING:
            if content["status"] == TRANSPORT_CHARGED:
                self.agent.transport_charged()

```

(continues on next page)

(continued from previous page)

```

        await self.agent.drop_station()

    elif performative == CANCEL_PERFORMATIVE:
        logger.info("Cancellation of request for {} information".format(self.
→agent.fleet_type))

```

## Helpers

There are some helper coroutines that are specific for the transport strategy:

```

async def send_proposal(self, customer_id, content=None)
async def cancel_proposal(self, customer_id, content=None)
async def pick_up_customer(self, customer_id, origin, dest)

```

The definition and purpose of each of them is now introduced:

- `send_proposal`

This helper function simplifies the composition and sending of a message containing a proposal to a customer. It sends a Message to `customer_id` using the **REQUEST\_PROTOCOL** and a **PROPOSE\_PERFORMATIVE**. It optionally accepts a `content` parameter where you can include any additional information you may want the customer to analyze.

- `cancel_proposal`

This helper function simplifies the composition and sending of a message to a customer to cancel a proposal. It sends a Message to `customer_id` using the **REQUEST\_PROTOCOL** and a **CANCEL\_PERFORMATIVE**. It optionally accepts a `content` parameter where you can include any additional information you may want the customer to analyze.

- `pick_up_customer`

This helper function triggers the **TRAVEL\_PROTOCOL** of a transport, which is the protocol that is used to transport a customer from her current position to her destination. This is a very important and particular function. Invoking this function is normally the last instruction of this strategy, since it means that the purpose of the strategy is accomplished (until the **TRAVEL\_PROTOCOL** ends and the transport is again free and able to receive new requests from some other customers).

The `pick_up_customer` helper receives as parameters the id of the customer and the coordinates of the customer's current position (`origin`) and its destination (`dest`).

### 4.3.5 Developing the Customer Agent Strategy

To develop a new strategy for the Customer Agent, you need to create a class that inherits from `CustomerStrategyBehaviour`. Since this is a cyclic behaviour class that follows the *Strategy Pattern* and that inherits from the `StrategyBehaviour`, it has all the previously presented helper functions for communication and storing data inside the agent.

The customer strategy is intended to ask a fleet manager agent for a transport service, then wait for transport proposals and, after evaluating them, choosing a particular transport proposal which will take the customer to her destination.

The place in the code where your customer strategy must be coded is the `run` coroutine. This function is executed in an infinite loop until the agent stops. In addition, you may also overload the `on_start` and the `on_end` coroutines, in order to execute code before the creation of the strategy or after its destruction, if needed.

## Code

The default strategy of a Customer agent is a dummy strategy that simply accepts the first proposal it receives. This is the code of the default customer strategy `AcceptFirstRequestBehaviour`:

```
from simfleet.customer import CustomerStrategyBehaviour

class AcceptFirstRequestTransportBehaviour(CustomerStrategyBehaviour):

    async def run(self):
        if self.agent.fleetmanagers is None:
            await self.send_get_managers(self.agent.fleet_type)

        msg = await self.receive(timeout=5)
        if msg:
            performative = msg.get_metadata("performative")
            if performative == INFORM_PERFORMATIVE:
                self.agent.fleetmanagers = json.loads(msg.body)
                return
            elif performative == CANCEL_PERFORMATIVE:
                logger.info("Cancellation of request for {} information".
                    ↪format(self.agent.type_service))
                return

        if self.agent.status == CUSTOMER_WAITING:
            await self.send_request(content={})

        msg = await self.receive(timeout=5)

        if msg:
            performative = msg.get_metadata("performative")
            transport_id = msg.sender
            if performative == PROPOSE_PERFORMATIVE:
                if self.agent.status == CUSTOMER_WAITING:
                    logger.debug(
                        "Customer {} received proposal from transport {}".format(self.
                    ↪agent.name, transport_id))
                    await self.accept_transport(transport_id)
                    self.agent.status = CUSTOMER_ASSIGNED
                else:
                    await self.refuse_transport(transport_id)

            elif performative == CANCEL_PERFORMATIVE:
                if self.agent.transport_assigned == str(transport_id):
                    logger.warning(
                        "Customer {} received a CANCEL from Transport {}".format(
                    ↪format(self.agent.name, transport_id))
                    self.agent.status = CUSTOMER_WAITING
```

## Helpers

There are some helper coroutines that are specific for the customer strategy:

```
async def send_get_managers(content=None)
async def send_request(self, content=None)
async def accept_transport(self, transport_aid)
async def refuse_transport(self, transport_aid)
```

The definition and purpose of each of them is now introduced:

- `send_get_managers`

This helper makes a query to the Directory agent to find all the fleet managers that provide a fleet service of type *content*. Thus, you can filter those fleet managers that provide the transport service that you are looking for. It is expected for the user to store the response of this query in the `self.agent.fleetmanagers` variable as a dictionary. This variable will be used by the next helper.

- `send_request`

This helper is useful to make a new request without building the entire message (the function makes it for you). It creates a `Message` with a **REQUEST** performative and sends it to all the fleet manager agents stored in `self.agent.fleetmanagers`. In addition, you can append a content to the request message to be used by the fleet manager agent or the transport agents (e.g. your origin coordinates or your destination coordinates).

- `accept_transport`

This is a helper function to send an acceptance message to a `transport_id`. It sends a `Message` with an **ACCEPT** performative to the selected transport.

- `refuse_transport`

This is a helper function to refuse a proposal from a `transport_id`. It sends a `Message` with an **REFUSE** performative to the transport whose proposal is being refused.

### 4.3.6 Other Helpers

SimFleet also includes a `helpers` module which provides some general support methods that may be useful for any agent. These functions are now introduced:

- `are_close`

This helper function facilitates working with distances in maps. This helper function accepts two coordinates (`coord1` and `coord2`) and an optional parameter to set the tolerance in meters. It returns `True` if both coordinates are closer than the tolerance in meters (10 meters by default). Otherwise it returns `False`.

Example:

```
assert are_close([39.253, -0.341], [39.351, -0.333], 1000) == True
```

- `distance_in_meters`

This helper function returns the distance in meters between two points.

Example:

```
assert distance_in_meters([-0.37565, 39.44447], [-0.40392, 39.45293]) == 3264.7134341427977
```

## 4.4 How to Implement New Strategies – Recommendations

At this point is time for you to implement your own strategies to optimize the problem of dispatching transports to customers. In this chapter we have shown you the tools to create these strategies. You have to create a file (in this example we are using `my_strategy_file.py`) and develop the strategies to be tested following the next template:

```

from simfleet.fleetmanager import FleetManagerStrategyBehaviour
from simfleet.customer import CustomerStrategyBehaviour
from simfleet.transport import TransportStrategyBehaviour

#####
#
#           FleetManager Strategy
#
#####
class MyFleetManagerStrategy(FleetManagerStrategyBehaviour):
    async def run(self):
        # Your code here

#####
#
#           Transport Strategy
#
#####
class MyTransportStrategy(TransportStrategyBehaviour):
    async def run(self):
        # Your code here

#####
#
#           Customer Strategy
#
#####
class MyCustomerStrategy(CustomerStrategyBehaviour):
    async def run(self):
        # Your code here

```

In this file, three strategies have been created for the three types of agent handled by the simulator. We have called these strategies `MyFleetManagerStrategy`, `MyTransportStrategy` and `MyCustomerStrategy`.

To run the simulator with your new strategies the configuration file accepts three parameters with the name of the file (without extension) and the name of the class of each strategy.

```

{
    "fleets": [...],
    "transports": [...],
    "customers": [...],
    "stations": [...],
    "simulation_name": "My Config",
    "max_time": 1000,
    "transport_strategy": "my_strategy_file.MyTransportStrategy",
    "customer_strategy": "my_strategy_file.MyCustomerStrategy",
    "fleetmanager_strategy": "my_strategy_file.MyFleetManagerStrategy",
    ...
    "host": "localhost",
}

```

```
$ simfleet --config my_custom_simulation.json
```

**Warning:** The file must be in the current working directory and it must be referenced *without* the extension (if the file is named `my_strategy_file.py` use `my_strategy_file` when calling the simulator).



Once run the simulator you can test your strategies using the graphical web interface or by inspecting the output of the logs in the command line.



Information on specific functions, classes, and methods.

## 5.1 simfleet package

### 5.1.1 Submodules

#### 5.1.2 simfleet.cli module

Console script for SimFleet.

#### 5.1.3 simfleet.config module

**class** `simfleet.config.SimfleetConfig` (*filename=None, name=None, max\_time=None, verbose=None*)

Bases: `object`

A scenario object reads a file with a JSON representation of a scenario and is used to create the participant agents.

**load\_config** (*filename*)

**num\_customers**

**num\_managers**

**num\_stations**

**num\_transport**

`simfleet.config.hide_passwords` (*item, key=None*)

### 5.1.4 simfleet.customer module

```
class simfleet.customer.CustomerAgent (agentjid, password)
    Bases: spade.agent.Agent

get_pickup_time ()
    Returns the time that the customer was waiting to be picked up since it has been assigned to a transport.

    Returns The time that the customer was waiting to a transport since it has been assigned.

    Return type float

get_position ()
    Returns the current position of the customer.

    Returns the coordinates of the current position of the customer (lon, lat)

    Return type list

get_waiting_time ()
    Returns the time that the agent was waiting for a transport, from its creation until it gets into a transport.

    Returns The time the customer was waiting.

    Return type float

is_in_destination ()
    Checks if the customer has arrived to its destination.

    Returns whether the customer is at its destination or not

    Return type bool

request_path (origin, destination)
    Requests a path between two points (origin and destination) using the RouteAgent service.

    Parameters

- origin (list) – the coordinates of the origin of the requested path
- destination (list) – the coordinates of the end of the requested path

Returns A list of points that represent the path from origin to destination, the distance and the estimated duration

    Return type list, float, float

run_strategy ()
    import json Runs the strategy for the customer agent.

set_directory (directory_id)
    Sets the directory JID address :param directory_id: the DirectoryAgent jid :type directory_id: str

set_fleet_type (fleet_type)
    Sets the type of fleet to be used.

    Parameters fleet_type (str) – the type of the fleet to be used

set_fleetmanager (fleetmanagers)
    Sets the fleetmanager JID address :param fleetmanagers: the fleetmanager jid :type fleetmanagers: str

set_icon (icon)

set_id (agent_id)
    Sets the agent identifier :param agent_id: The new Agent Id :type agent_id: str
```

**set\_position** (*coords=None*)

Sets the position of the customer. If no position is provided it is located in a random position.

**Parameters** **coords** (*list*) – a list coordinates (longitude and latitude)

**set\_route\_agent** (*route\_id*)

Sets the route agent JID address :param route\_id: the route agent jid :type route\_id: str

**set\_target\_position** (*coords=None*)

Sets the target position of the customer (i.e. its destination). If no position is provided the destination is setted to a random position.

**Parameters** **coords** (*list*) – a list coordinates (longitude and latitude)

**setup** ()

Setup agent before startup. This coroutine may be overloaded.

**to\_json** ()

Serializes the main information of a customer agent to a JSON format. It includes the id of the agent, its current position, the destination coordinates of the agent, the current status, the transport that it has assigned (if any) and its waiting time.

**Returns**

a JSON doc with the main information of the customer.

Example:

```
{
  "id": "cphillips",
  "position": [ 39.461327, -0.361839 ],
  "dest": [ 39.460599, -0.335041 ],
  "status": 24,
  "transport": "ghiggins@127.0.0.1",
  "waiting": 13.45
}
```

**Return type** dict

**total\_time** ()

Returns the time since the customer was activated until it reached its destination.

**Returns** the total time of the customer's simulation.

**Return type** float

**class** `simfleet.customer.CustomerStrategyBehaviour`

Bases: `simfleet.utils.StrategyBehaviour`

Class from which to inherit to create a transport strategy. You must overload the `run` coroutine

**Helper functions:**

- `send_request`
- `accept_transport`
- `refuse_transport`

**accept\_transport** (*transport\_id*)

Sends a `spade.message.Message` to a transport to accept a travel proposal. It uses the `REQUEST_PROTOCOL` and the `ACCEPT_PERFORMATIVE`.

**Parameters** **transport\_id** (*str*) – The Agent JID of the transport

**on\_start()**

Initializes the logger and timers. Call to parent method if overloaded.

**refuse\_transport** (*transport\_id*)

Sends an `spade.message.Message` to a transport to refuse a travel proposal. It uses the `REQUEST_PROTOCOL` and the `REFUSE_PERFORMATIVE`.

**Parameters** `transport_id` (*str*) – The Agent JID of the transport

**run()**

Body of the behaviour. To be implemented by user.

**send\_get\_managers** (*content=None*)

Sends an `spade.message.Message` to the `DirectoryAgent` to request a managers. It uses the `QUERY_PROTOCOL` and the `REQUEST_PERFORMATIVE`. If no content is set a default content with the `type_service` that needs :param content: Optional content dictionary :type content: dict

**send\_request** (*content=None*)

Sends an `spade.message.Message` to the `fleetmanager` to request a transport. It uses the `REQUEST_PROTOCOL` and the `REQUEST_PERFORMATIVE`. If no content is set a default content with the `customer_id`, `origin` and `target` coordinates is used.

**Parameters** `content` (*dict*) – Optional content dictionary

**class** `simfleet.customer.TravelBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

This is the internal behaviour that manages the movement of the customer. It is triggered when the transport informs the customer that it is going to the customer's position until the customer is dropped in its destination.

**on\_start()**

Coroutine called before the behaviour is started.

**run()**

Body of the behaviour. To be implemented by user.

### 5.1.5 simfleet.directory module

**class** `simfleet.directory.DirectoryAgent` (*agentjid, password*)

Bases: `spade.agent.Agent`

**run\_strategy()**

Runs the strategy for the directory agent.

**set\_id** (*agent\_id*)

Sets the agent identifier

**Parameters** `agent_id` (*str*) – The new agent id

**setup()**

Setup agent before startup. This coroutine may be overloaded.

**class** `simfleet.directory.DirectoryStrategyBehaviour`

Bases: `simfleet.utils.StrategyBehaviour`

Class from which to inherit to create a directory strategy.

**on\_start()**

Coroutine called before the behaviour is started.

```

run ()
    Body of the behaviour. To be implemented by user.

send_negative (agent_id)
    Sends a message to the current assigned manager/station to cancel the registration.

    Parameters agent_id (str) – the id of the manager/station

send_services (agent_id, type_service)
    Send a message to the customer or transport with the current information of the type of service they need.

    Parameters
        • agent_id (str) – the id of the manager/station
        • type_service (str) – the type of service

class simfleet.directory.RegistrationBehaviour
    Bases: spade.behaviour.CyclicBehaviour

    add_service (content)
        Adds a new service to the store.

        Parameters content (dict) – content to be added

    on_start ()
        Coroutine called before the behaviour is started.

    remove_service (service_type, agent)
        Erase a service from the store.

        Parameters
            • service_type (str) – the service type to be erased
            • agent (str) – an str with the jid of the agent to be erased

    run ()
        Body of the behaviour. To be implemented by user.

    send_confirmation (agent_id)
        Send a spade.message.Message with an acceptance to manager/station to register in the dictionary

```

### 5.1.6 simfleet.fleetmanager module

```

class simfleet.fleetmanager.FleetManagerAgent (agentjid, password)
    Bases: spade.agent.Agent

    FleetManager agent that manages the requests between transports and customers

    clear_agents ()
        Resets the set of transports and customers. Resets the simulation clock.

    run_strategy ()
        Runs the strategy for the transport agent.

    set_directory (directory_id)
        Sets the directory JID address :param directory_id: the DirectoryAgent jid :type directory_id: str

    set_fleet_type (fleet_type)
        Sets the type of service to the fleet :param type_service: type of service :type type_service: str

    set_icon (icon)

```

**set\_id** (*agent\_id*)  
Sets the agent identifier

**Parameters** **agent\_id** (*str*) – The new Agent Id

**set\_registration** (*status*)  
Sets the status of registration :param status: True if the transport agent has registered or False if not :type status: boolean

**setup** ()  
Setup agent before startup. This coroutine may be overloaded.

**class** `simfleet.fleetmanager.FleetManagerStrategyBehaviour`

Bases: `simfleet.utils.StrategyBehaviour`

Class from which to inherit to create a coordinator strategy. You must overload the `_process()` method

**Helper functions:**

- `get_transport_agents()`

**get\_transport\_agents** ()  
Gets the list of registered transports

**Returns** a list of `TransportAgent`

**Return type** list

**on\_start** ()  
Coroutine called before the behaviour is started.

**run** ()  
Body of the behaviour. To be implemented by user.

**send\_registration** ()  
Send a `spade.message.Message` with a proposal to directory to register.

**class** `simfleet.fleetmanager.TransportRegistrationForFleetBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

**accept\_registration** (*agent\_id*)  
Send a `spade.message.Message` with an acceptance to transport to register in the fleet.

**add\_transport** (*agent*)  
Adds a new `TransportAgent` to the store.

**Parameters** **agent** (`TransportAgent`) – the instance of the `TransportAgent` to be added

**on\_start** ()  
Coroutine called before the behaviour is started.

**reject\_registration** (*agent\_id*)  
Send a `spade.message.Message` with an acceptance to transport to register in the fleet.

**remove\_transport** (*key*)  
Erase a `TransportAgent` to the store.

**Parameters** **agent** (`TransportAgent`) – the instance of the `TransportAgent` to be erased

**run** ()  
Body of the behaviour. To be implemented by user.



### 5.1.7 simfleet.helpers module

Helpers module

These functions are useful for the develop of new strategies.

**exception** `simfleet.helpers.AlreadyInDestination`

Bases: `Exception`

This exception is raised when an agent wants to move to a destination where it is already there.

**exception** `simfleet.helpers.PathRequestException`

Bases: `Exception`

This exception is raised when a path could not be computed.

`simfleet.helpers.are_close(coord1, coord2, tolerance=10)`

Checks wheter two points are close or not. The tolerance is expressed in meters.

**Parameters**

- **coord1** (*list*) – a coordinate (longitude, latitude)
- **coord2** (*list*) – another coordinate (longitude, latitude)
- **tolerance** (*int*) – tolerance in meters

**Returns** whether the two coordinates are closer than tolerance or not

**Return type** `bool`

`simfleet.helpers.distance_in_meters(coord1, coord2)`

Returns the distance between two coordinates in meters.

**Parameters**

- **coord1** (*list*) – a coordinate (longitude, latitude)
- **coord2** – another coordinate (longitude, latitude)

**Returns** distance meters between the two coordinates

**Return type** `float`

`simfleet.helpers.kmh_to_ms(speed_in_kmh)`

Convert kilometers/hour to meters/second.

**Parameters** **speed\_in\_kmh** (*float*) – speed in kilometers/hour

**Returns** the speed in meters/second

**Return type** `float`

`simfleet.helpers.random_position()`

Returns a random position inside the map.

**Returns** a point (longitude and latitude)

**Return type** `list`

### 5.1.8 simfleet.protocol module

protocol and performative constants

### 5.1.9 simfleet.route module

**class** `simfleet.route.RouteAgent` (*agentjid, password*)

Bases: `spade.agent.Agent`

The RouteAgent receives request for paths, queries an OSRM server and returns the information. It also caches the queries to avoid overloading the OSRM server.

**class** `RequestRouteBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

This cyclic behaviour listens for route requests from other agents. When a message is received it answers with the path.

**on\_end** ()

Coroutine called after the behaviour is done or killed.

**on\_start** ()

Coroutine called before the behaviour is started.

**run** ()

Body of the behaviour. To be implemented by user.

**get\_route** (*origin, destination*)

Checks the cache for a path, if not found then it queries the OSRM server.

**Parameters**

- **origin** (*list*) – origin coordinate (longitude, latitude)
- **destination** (*list*) – target coordinate (longitude, latitude)

**Returns** a dict with three keys: path, distance and duration

**Return type** dict

**load\_cache** ()

Loads the cache from file.

**persist\_cache** ()

Persists the cache to a JSON file.

**static request\_route\_to\_server** (*origin, destination*)

Queries the OSRM for a path.

**Parameters**

- **origin** (*list*) – origin coordinate (longitude, latitude)
- **destination** (*list*) – target coordinate (longitude, latitude)

**Returns** list, float, float = the path, the distance of the path and the estimated duration

**setup** ()

Setup agent before startup. This coroutine may be overloaded.

### 5.1.10 simfleet.simulator module

**class** `simfleet.simulator.SimulatorAgent` (*config, agentjid='simulator@localhost', password='simulator123j3'*)

Bases: `spade.agent.Agent`

The Simulator. It manages all the simulation processes. Tasks done by the simulator at initialization:

1. Create the XMPP server
2. Run the SPADE backend
3. Run the directory and route agents.
4. Create agents defined in scenario (if any).

After these tasks are done in the Simulator constructor, the simulation is started when the `run` method is called.

**add\_customer** (*agent*)

Adds a new `CustomerAgent` to the store.

**Parameters** *agent* (`CustomerAgent`) – the instance of the `CustomerAgent` to be added

**add\_manager** (*agent*)

Adds a new `FleetManagerAgent` to the store.

**Parameters** *agent* (`FleetManagerAgent`) – the instance of the `FleetManagerAgent` to be added

**add\_station** (*agent*)

Adds a new `StationAgent` to the store.

**Parameters** *agent* (`StationAgent`) – the instance of the `StationAgent` to be added

**add\_transport** (*agent*)

Adds a new `TransportAgent` to the store.

**Parameters** *agent* (`TransportAgent`) – the instance of the `TransportAgent` to be added

**all\_customers\_in\_destination** ()

Checks whether the simulation has finished or not. A simulation is finished if all customers are at their destinations. If there is no customers the simulation is not finished.

**Returns:** `bool`: whether the simulation has finished or not.

**assigning\_fleet\_icon** (*fleet\_type*, *default=None*)

**async\_start\_agent** (*agent*)

**clean\_controller** (*request*)

Web controller that resets the simulator to a clean state.

**Returns** no template is returned since this is an AJAX controller, a dict with status=done

**Return type** dict

**clear\_agents** ()

Resets the set of transports and customers. Resets the simulation clock.

**clear\_stopped\_agents** ()

Removes from the transport and customer sets every agent that is stopped.

**collect\_stats** ()

Collects stats from all participant agents and from the simulation and stores it in three dataframes.

**create\_customer\_agent** (*name*, *password*, *fleet\_type*, *position*, *strategy=None*, *target=None*)

Create a customer agent.

**Parameters**

- **name** (*str*) – name of the agent
- **password** (*str*) – password of the agent

- **position** (*list*) – initial coordinates of the agent
- **fleet\_type** (*str*) – type of the fleet to be or demand
- **target** (*list*, *optional*) – destination coordinates of the agent
- **speed** (*float*, *optional*) – speed of the vehicle

**create\_directory\_agent** (*name*, *password*)

**create\_fleetmanager\_agent** (*name*, *password*, *fleet\_type*, *strategy=None*, *icon=None*)

**create\_station\_agent** (*name*, *password*, *position*, *power*, *places*, *strategy=None*)

Create a customer agent.

#### Parameters

- **name** (*str*) – name of the agent
- **password** (*str*) – password of the agent
- **position** (*list*) – initial coordinates of the agent
- **fleet\_type** (*str*) – type of the fleet to be or demand
- **target** (*list*, *optional*) – destination coordinates of the agent
- **speed** (*float*, *optional*) – speed of the vehicle

**create\_transport\_agent** (*name*, *password*, *fleet\_type*, *fleetmanager*, *position*, *strategy=None*, *speed=None*, *autonomy=None*, *current\_autonomy=None*)

**customer\_agents**

Gets the dict of registered customers

**Returns** a dict of CustomerAgent with the name in the key

**Return type** dict

**download\_stats\_excel\_controller** (*request*)

Web controller that returns an Excel file with the simulation results.

**Returns** a Response of type “attachment” with the file content.

**Return type** Response

**download\_stats\_json\_controller** (*request*)

Web controller that returns a JSON file with the simulation results.

**Returns** a Response of type “attachment” with the file content.

**Return type** Response

**entities\_controller** (*request*)

Web controller that returns a dict with the entities of the simulator and their statuses.

Example of the entities returned data:

```
{
  "customers": [
    {
      "status": 24,
      "transport": "transport2@127.0.0.1",
      "dest": [ 39.463356, -0.376463 ],
      "waiting": 3.25,
      "position": [ 39.460568, -0.352529 ],
      "id": "michaelstewart"
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    }
  ],
  "transports": [
    {
      "status": 11,
      "customer": "michaelstewart@127.0.0.1",
      "assignments": 1,
      "path": [
        [ 39.478328, -0.406712 ],
        [ 39.478317, -0.406814 ],
        [ 39.460568, -0.352529 ]
      ],
      "dest": [ 39.460568, -0.352529 ],
      "position": [ 39.468131, -0.39685 ],
      "speed": 327.58,
      "id": "transport2",
      "distance": "6754.60"
    }
  ],
  "stats": {
    "totaltime": "-1.00",
    "waiting": "3.25",
    "finished": False,
    "is_running": True
  },
  "tree": {
    "name": "Agents",
    "children": [
      {
        "count": "1",
        "name": "Transports",
        "children": [ { "status": 11, "name": " transport2", "icon":
↪ "fa-transport" } ]
      },
      {
        "count": "1",
        "name": "Customers",
        "children": [ { "status": 24, "name": " michaelstewart", "icon
↪ ": "fa-user" } ]
      }
    ]
  },
  "authenticated": False,
  "stations": [
    {
      "status": 24,
      "position": [ 39.460568, -0.352529 ],
      "id": "michaelstewart"
    }
  ],
}

```

**Returns** no template is returned since this is an AJAX controller, a dict with the list of transports, the list of customers, the tree view to be showed in the sidebar and the stats of the simulation.

**Return type** dict

**generate\_tree()**

Generates the tree view in JSON format to be showed in the sidebar.

**Returns** a dict with all the agents in the simulator, with their name, status and icon.

**Return type** dict

**get\_customer\_stats()**

Creates a dataframe with the simulation stats of the customers The dataframe includes for each customer its name, waiting time, total time and status.

**Returns** the dataframe with the customers stats.

**Return type** pandas.DataFrame

**get\_directory()****get\_manager\_stats()**

Creates a dataframe with the simulation stats of the customers The dataframe includes for each customer its name, waiting time, total time and status.

**Returns** the dataframe with the customers stats.

**Return type** pandas.DataFrame

**get\_simulation\_time()**

Returns the elapsed simulation time to the current time. If the simulation is not started it returns 0.

**Returns** the whole simulation time.

**Return type** float

**get\_station\_stats()**

Creates a dataframe with the simulation stats of the customers The dataframe includes for each customer its name, waiting time, total time and status.

**Returns** the dataframe with the customers stats.

**Return type** pandas.DataFrame

**get\_stats()**

Generates the stats of the simulation in JSON format.

Examples:

```
{
  "totaltime": "12.25",
  "waiting": "3.25",
  "finished": False,
  "is_running": True
}
```

**Returns** a dict with the total time, waiting time, is\_running and finished values

**Return type** dict

**get\_stats\_dataframes()**

Collects simulation stats and returns 3 dataframes with the information: A general dataframe with the average information, a dataframe with the transport's information and a dataframe with the customer's information. :returns: avg df, transport df and customer df :rtype: pandas.DataFrame, pandas.DataFrame, pandas.DataFrame

**get\_transport\_stats()**

Creates a dataframe with the simulation stats of the transports. The dataframe includes for each transport its name, assignments, traveled distance and status.

**Returns** the dataframe with the transports stats.

**Return type** `pandas.DataFrame`

**index\_controller(request)**

Web controller that returns the index page of the simulator.

**Returns** the name of the template, the data to be pre-processed in the template

**Return type** `dict`

**init\_controller(request)****is\_simulation\_finished()**

Checks if the simulation is finished. A simulation is finished if the max simulation time has been reached or when the fleetmanager says it.

**Returns** whether the simulation is finished or not.

**Return type** `bool`

**load\_icons(filename)****load\_scenario()**

Load the information from the preloaded scenario through the `SimfleetConfig` class

**manager\_agents**

Gets the dict of registered `FleetManager`

**Returns** a dict of `FleetManagerAgents` with the name in the key

**Return type** `dict`

**print\_stats()**

Prints the dataframes collected by `collect_stats`.

**request\_path(origin, destination)**

Requests a path to the `RouteAgent`.

**Parameters**

- **origin** (*list*) – the origin coordinates (lon, lat)
- **destination** (*list*) – the target coordinates (lon, lat)

**Returns** the path as a list of points, the distance of the path, the estimated duration of the path

**Return type** `list, float, float`

**run()**

Starts the simulation

**run\_controller(request)**

Web controller that starts the simulator.

**Returns** no template is returned since this is an AJAX controller, an empty data dict is returned

**Return type** `dict`

**set\_default\_strategies(fleetmanager\_strategy, transport\_strategy, customer\_strategy, directory\_strategy, station\_strategy)**

Gets the strategy strings and loads their classes. These strategies are prepared to be injected into any new transport or customer agent.

**Parameters**

- **fleetmanager\_strategy** (*str*) – the path to the fleetmanager strategy
- **transport\_strategy** (*str*) – the path to the transport strategy
- **customer\_strategy** (*str*) – the path to the customer strategy
- **directory\_strategy** (*str*) – the path to the directory strategy
- **station\_strategy** (*str*) – the path to the station strategy

**set\_directory** (*agent*)

**set\_icon** (*agent*, *icon*, *default=None*)

**setup** ()

Setup agent before startup. This coroutine may be overloaded.

**station\_agents**

Gets the dict of registered stations

**Returns** a dict of `StationAgent` with the name in the key

**Return type** dict

**stop** ()

Finishes the simulation and prints simulation stats. Tasks done when a simulation is stopped:

1. Stop participant agents.
2. Print stats.
3. Stop Route agent.
4. Stop fleetmanager agent.

**stop\_agents** ()

Stops the simulator and all the agents

**stop\_agents\_controller** (*request*)

Web controller that stops all the customer and transport agents.

**Returns** no template is returned since this is an AJAX controller, a dict with status=done

**Return type** dict

**time\_is\_out** ()

Checks if the max simulation time has been reached.

**Returns** whether the max simulation time has been reached or not.

**Return type** bool

**transport\_agents**

Gets the dict of registered transports

**Returns** a dict of `TransportAgent` with the name in the key

**Return type** dict

**write\_excel** (*filename*)

Writes the collected data by `collect_stats` in an excel file.

**Parameters** **filename** (*str*) – name of the excel file.

**write\_file** (*filename*, *fileformat='json'*)

Writes the dataframes collected by `collect_stats` in JSON or Excel format.



**Parameters**

- **filename** (*str*) – name of the output file to be written.
- **fileformat** (*str*) – format of the output file. Choices: json or excel

**write\_json** (*filename*)

Writes the collected data by `collect_stats` in a json file.

**Parameters** **filename** (*str*) – name of the json file.

**5.1.11 simfleet.station module**

**class** `simfleet.station.ChargeBehaviour` (*start\_at, transport\_id*)

Bases: `spade.behaviour.TimeoutBehaviour`

**charging\_complete** ()

Send a message to the transport agent that the vehicle load has been completed

**run** ()

Body of the behaviour. To be implemented by user.

**class** `simfleet.station.RegistrationBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

**on\_start** ()

Coroutine called before the behaviour is started.

**run** ()

Body of the behaviour. To be implemented by user.

**send\_registration** ()

Send a `spade.message.Message` with a proposal to directory to register.

**set\_registration** (*decision*)

**class** `simfleet.station.StationAgent` (*agentjid, password*)

Bases: `spade.agent.Agent`

**assigning\_place** ()

Set a space in the charging station for the transport that has been accepted, when the available spaces are zero, the status will change to `BUSY_STATION`

**charging\_transport** (*need, transport\_id*)

**deassigning\_place** ()

Leave a space of the charging station, when the station has free spaces, the status will change to `FREE_STATION`

**get\_available\_places** ()

**get\_position** ()

Returns the current position of the station.

**Returns** the coordinates of the current position of the customer (lon, lat)

**Return type** list

**get\_power** ()

**get\_status** ()

**run\_strategy** ()

Sets the strategy for the transport agent.

**set\_available\_places** (*places*)

**set\_directory** (*directory\_id*)

Sets the directory JID address :param directory\_id: the DirectoryAgent jid :type directory\_id: str

**set\_icon** (*icon*)

**set\_id** (*agent\_id*)

Sets the agent identifier

**Parameters** **agent\_id** (*str*) – The new Agent Id

**set\_position** (*coords=None*)

Sets the position of the station. If no position is provided it is located in a random position.

**Parameters** **coords** (*list*) – a list coordinates (longitude and latitude)

**set\_power** (*charge*)

**set\_registration** (*status*)

Sets the status of registration :param status: True if the transport agent has registered or False if not :type status: boolean

**set\_status** (*state='FREE\_STATION'*)

**set\_type** (*station\_type*)

**setup** ()

Setup agent before startup. This coroutine may be overloaded.

**to\_json** ()

Serializes the main information of a station agent to a JSON format. It includes the id of the agent, its current position, the destination coordinates of the agent, the current status, the transport that it has assigned (if any) and its waiting time.

#### Returns

a JSON doc with the main information of the station.

Example:

```
{
  "id": "cphillips",
  "position": [ 39.461327, -0.361839 ],
  "status": True,
  "places": 10,
  "power": 10
}
```

**Return type** dict

**class** `simfleet.station.StationStrategyBehaviour`

Bases: `simfleet.utils.StrategyBehaviour`

Class from which to inherit to create a station strategy. You must overload the `run()` method

#### Helper functions:

- `get_transport_agents()`

**accept\_transport** (*transport\_id*)

Sends a `spade.message.Message` to a transport to accept a travel proposal for charge. It uses the `REQUEST_PROTOCOL` and the `ACCEPT_PERFORMATIVE`.

**Parameters** **transport\_id** (*str*) – The Agent JID of the transport

**on\_start()**

Coroutine called before the behaviour is started.

**refuse\_transport(*transport\_id*)**

Sends an `spade.message.Message` to a transport to refuse a travel proposal for charge. It uses the `REQUEST_PROTOCOL` and the `REFUSE_PERFORMATIVE`.

**Parameters** `transport_id` (*str*) – The Agent JID of the transport

**run()**

Body of the behaviour. To be implemented by user.

**class** `simfleet.station.TravelBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

This is the internal behaviour that manages the inform of the station. It is triggered when the transport informs the station that it is going to the customer's position until the customer is dropped in its destination.

**on\_start()**

Coroutine called before the behaviour is started.

**run()**

Body of the behaviour. To be implemented by user.

### 5.1.12 simfleet.strategies module

**class** `simfleet.strategies.AcceptAlwaysStrategyBehaviour`

Bases: `simfleet.transport.TransportStrategyBehaviour`

The default strategy for the Transport agent. By default it accepts every request it receives if available.

**run()**

Body of the behaviour. To be implemented by user.

**class** `simfleet.strategies.AcceptFirstRequestBehaviour`

Bases: `simfleet.customer.CustomerStrategyBehaviour`

The default strategy for the Customer agent. By default it accepts the first proposal it receives.

**run()**

Body of the behaviour. To be implemented by user.

**class** `simfleet.strategies.DelegateRequestBehaviour`

Bases: `simfleet.fleetmanager.FleetManagerStrategyBehaviour`

The default strategy for the FleetManager agent. By default it delegates all requests to all transports.

**run()**

Body of the behaviour. To be implemented by user.

### 5.1.13 simfleet.strategies\_fsm module

**class** `simfleet.strategies_fsm.FSMTransportStrategyBehaviour`

Bases: `spade.behaviour.FSMBehaviour`

**setup()**

**class** `simfleet.strategies_fsm.TransportMovingState`

Bases: `simfleet.transport.TransportStrategyBehaviour`, `spade.behaviour.State`

```
    on_start()
        Coroutine called before the behaviour is started.

    run()
        Body of the behaviour. To be implemented by user.

class simfleet.strategies_fsm.TransportWaitingForApprovalState
    Bases: simfleet.transport.TransportStrategyBehaviour, spade.behaviour.State

    on_start()
        Coroutine called before the behaviour is started.

    run()
        Body of the behaviour. To be implemented by user.

class simfleet.strategies_fsm.TransportWaitingState
    Bases: simfleet.transport.TransportStrategyBehaviour, spade.behaviour.State

    on_start()
        Coroutine called before the behaviour is started.

    run()
        Body of the behaviour. To be implemented by user.

simfleet.strategies_fsm.passenger_in_transport_callback(old, new)
```

#### 5.1.14 simfleet.transport module

```
class simfleet.transport.RegistrationBehaviour
    Bases: spade.behaviour.CyclicBehaviour

    on_start()
        Coroutine called before the behaviour is started.

    run()
        Body of the behaviour. To be implemented by user.

    send_registration()
        Send a spade.message.Message with a proposal to manager to register.

class simfleet.transport.TransportAgent(agentjid, password)
    Bases: spade.agent.Agent

    class MovingBehaviour(period, start_at=None)
        Bases: spade.behaviour.PeriodicBehaviour

        This is the internal behaviour that manages the movement of the transport. It is triggered when the transport has a new destination and the periodic tick is recomputed at every step to show a fine animation. This moving behaviour includes to update the transport coordinates as it moves along the path at the specified speed.

        run()
            Body of the behaviour. To be implemented by user.

    arrived_to_destination()
        Informs that the transport has arrived to its destination. It recomputes the new destination and path if picking up a customer or drops it and goes to WAITING status again.

    arrived_to_station()
        Informs that the transport has arrived to its destination. It recomputes the new destination and path if picking up a customer or drops it and goes to WAITING status again.
```

**calculate\_km\_expense** (*origin, start, dest=None*)

**cancel\_customer** (*data=None*)

Sends a message to the current assigned customer to cancel the assignment.

**Parameters** *data* (*dict, optional*) – Complementary info about the cancellation

**drop\_customer** ()

Drops the customer that the transport is carrying in the current location.

**drop\_station** ()

Drops the customer that the transport is carrying in the current location.

**get\_autonomy** ()

**get\_position** ()

Returns the current position of the customer.

**Returns** the coordinates of the current position of the customer (lon, lat)

**Return type** list

**inform\_customer** (*status, data=None*)

Sends a message to the current assigned customer to inform her about a new status.

**Parameters**

- **status** (*int*) – The new status code
- **data** (*dict, optional*) – complementary info about the status

**inform\_station** (*data=None*)

Sends a message to the current assigned customer to inform her about a new status.

**Parameters**

- **status** (*int*) – The new status code
- **data** (*dict, optional*) – complementary info about the status

**is\_customer\_in\_transport** ()

**is\_free** ()

**is\_in\_destination** ()

Checks if the transport has arrived to its destination.

**Returns** whether the transport is at its destination or not

**Return type** bool

**move\_to** (*dest*)

Moves the transport to a new destination.

**Parameters** *dest* (*list*) – the coordinates of the new destination (in lon, lat format)

**Raises** `AlreadyInDestination` – if the transport is already in the destination coordinates.

**needs\_charging** ()

**request\_path** (*origin, destination*)

Requests a path between two points (origin and destination) using the RouteAgent service.

**Parameters**

- **origin** (*list*) – the coordinates of the origin of the requested path
- **destination** (*list*) – the coordinates of the end of the requested path

**Returns** A list of points that represent the path from origin to destination, the distance and the estimated duration

**Return type** list, float, float

### Examples

```
>>> path, distance, duration = await self.request_path(origin=[0,0],
↪destination=[1,1])
>>> print(path)
[[0,0], [0,1], [1,1]]
>>> print(distance)
2.0
>>> print(duration)
3.24
```

**run\_strategy()**

Sets the strategy for the transport agent.

**Parameters** **strategy\_class** (TransportStrategyBehaviour) – The class to be used. Must inherit from TransportStrategyBehaviour

**send(msg)**

**set\_autonomy(autonomy, current\_autonomy=None)**

**set\_directory(directory\_id)**

Sets the directory JID address :param directory\_id: the DirectoryAgent jid :type directory\_id: str

**set\_fleet\_type(fleet\_type)**

**set\_fleetmanager(fleetmanager\_id)**

Sets the fleetmanager JID address :param fleetmanager\_id: the fleetmanager jid :type fleetmanager\_id: str

**set\_icon(icon)**

**set\_id(agent\_id)**

Sets the agent identifier

**Parameters** **agent\_id** (str) – The new Agent Id

**set\_initial\_position(coords)**

**set\_km\_expense(expense=0)**

**set\_position(coords=None)**

Sets the position of the transport. If no position is provided it is located in a random position.

**Parameters** **coords** (list) – a list coordinates (longitude and latitude)

**set\_registration(status, content=None)**

Sets the status of registration :param status: True if the transport agent has registered or False if not :type status: boolean :param content: :type content: dict

**set\_route\_agent(route\_id)**

Sets the route agent JID address :param route\_id: the route agent jid :type route\_id: str

**set\_speed(speed\_in\_kmh)**

Sets the speed of the transport.

**Parameters** **speed\_in\_kmh** (float) – the speed of the transport in km per hour

**setup()**

Setup agent before startup. This coroutine may be overloaded.

**step()**

Advances one step in the simulation

**to\_json()**

Serializes the main information of a transport agent to a JSON format. It includes the id of the agent, its current position, the destination coordinates of the agent, the current status, the speed of the transport (in km/h), the path it is following (if any), the customer that it has assigned (if any), the number of assignments it has done and the distance that the transport has traveled.

**Returns**

a JSON doc with the main information of the transport.

Example:

```
{
  "id": "cphillips",
  "position": [ 39.461327, -0.361839 ],
  "dest": [ 39.460599, -0.335041 ],
  "status": 24,
  "speed": 1000,
  "path": [[0,0], [0,1], [1,0], [1,1], ...],
  "customer": "ghiggins@127.0.0.1",
  "assignments": 2,
  "distance": 3481.34
}
```

**Return type** dict

**transport\_charged()****watch\_value(key, callback)**

Registers an observer callback to be run when a value is changed

**Parameters**

- **key** (*str*) – the name of the value
- **callback** (*function*) – a function to be called when the value changes. It receives two arguments: the old and the new value.

**class** `simfleet.transport.TransportStrategyBehaviour`

Bases: `simfleet.utils.StrategyBehaviour`

Class from which to inherit to create a transport strategy. You must overload the `run` coroutine

**Helper functions:**

- `pick_up_customer`
- `send_proposal`
- `cancel_proposal`

**cancel\_proposal(customer\_id, content=None)**

Send a `spade.message.Message` to cancel a proposal. If the content is empty the proposal is sent without content.

**Parameters**

- **customer\_id** (*str*) – the id of the customer

- **content** (*dict*, *optional*) – the optional content of the message

**go\_to\_the\_station** (*station\_id*, *dest*)

Starts a TRAVEL\_PROTOCOL to pick up a customer and get him to his destination. It automatically launches all the travelling process until the customer is delivered. This travelling process includes to update the transport coordinates as it moves along the path at the specified speed.

**Parameters**

- **customer\_id** (*str*) – the id of the customer
- **origin** (*list*) – the coordinates of the current location of the customer
- **dest** (*list*) – the coordinates of the target destination of the customer

**has\_enough\_autonomy** (*customer\_orig*, *customer\_dest*)

**on\_start** ()

Coroutine called before the behaviour is started.

**pick\_up\_customer** (*customer\_id*, *origin*, *dest*)

Starts a TRAVEL\_PROTOCOL to pick up a customer and get him to his destination. It automatically launches all the travelling process until the customer is delivered. This travelling process includes to update the transport coordinates as it moves along the path at the specified speed.

**Parameters**

- **customer\_id** (*str*) – the id of the customer
- **origin** (*list*) – the coordinates of the current location of the customer
- **dest** (*list*) – the coordinates of the target destination of the customer

**run** ()

Body of the behaviour. To be implemented by user.

**send\_confirmation\_travel** (*station\_id*)

**send\_get\_stations** (*content=None*)

**send\_proposal** (*customer\_id*, *content=None*)

Send a `spade.message.Message` with a proposal to a customer to pick up him. If the content is empty the proposal is sent without content.

**Parameters**

- **customer\_id** (*str*) – the id of the customer
- **content** (*dict*, *optional*) – the optional content of the message

## 5.1.15 simfleet.utils module

**class** `simfleet.utils.RequestRouteBehaviour` (*msg: spade.message.Message*, *origin: list*,  
*destination: list*, *route\_agent: str*)

Bases: `spade.behaviour.OneShotBehaviour`

A one-shot behaviour that is executed to request for a new route to the route agent.

**run** ()

Body of the behaviour. To be implemented by user.

**class** `simfleet.utils.StrategyBehaviour`

Bases: `spade.behaviour.CyclicBehaviour`

The behaviour that all parent strategies must inherit from. It complies with the Strategy Pattern.



`simfleet.utils.avg(array)`

Makes the average of an array without Nones. :param array: a list of floats and Nones :type array: list

**Returns** the average of the list without the Nones.

**Return type** float

`simfleet.utils.chunk_path(path, speed_in_kmh)`

Splits the path into smaller chunks taking into account the speed.

**Parameters**

- **path** (*list*) – the original path. A list of points (lon, lat)
- **speed\_in\_kmh** (*float*) – the speed in km per hour at which the path is being traveled.

**Returns** a new path equivalent (to the first one), that has at least the same number of points.

**Return type** list

`simfleet.utils.load_class(class_path)`

Tricky method that imports a class from a string.

**Parameters** **class\_path** (*str*) – the path where the class to be imported is.

**Returns** the class imported and ready to be instantiated.

**Return type** class

`simfleet.utils.request_path(agent, origin, destination, route_id)`

Sends a message to the RouteAgent to request a path

**Parameters**

- **agent** – the agent who is requesting the path
- **origin** (*list*) – a list with the origin coordinates [longitude, latitude]
- **destination** (*list*) – a list with the target coordinates [longitude, latitude]

**Returns**

**a list of points (longitude and latitude) representing the path, the distance of the path in meters, a estimation of the duration of the path**

**Return type** list, float, float

## Examples

```
>>> path, distance, duration = request_path(agent, origin=[0,0], destination=[1,
↪1])
>>> print(path)
[[0,0], [0,1], [1,1]]
>>> print(distance)
2.0
>>> print(duration)
3.24
```

`simfleet.utils.status_to_str(status_code)`

Translates an int status code to a string that represents the status

**Parameters** **status\_code** (*int*) – the code of the status

**Returns** the string that represents the status

**Return type** str

`simfleet.utils.unused_port(hostname)`

Return a port that is unused on the current host.

### 5.1.16 Module contents

Top-level package for SimFleet.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

## 6.1 Types of Contributions

### 6.1.1 Report Bugs

Report bugs at <https://github.com/javipalanca/simfleet/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

### 6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

## 6.1.4 Write Documentation

SimFleet could always use more documentation, whether as part of the official SimFleet docs, in docstrings, or even on the web in blog posts, articles, and such.

## 6.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/javipalanca/simfleet/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 6.2 Get Started!

Ready to contribute? Here's how to set up *simfleet* for local development.

1. Fork the *simfleet* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/simfleet.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv simfleet
$ cd simfleet/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 simfleet tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6. Check [https://travis-ci.org/javipalanca/simfleet/pull\\_requests](https://travis-ci.org/javipalanca/simfleet/pull_requests) and make sure that the tests pass for all supported Python versions.

## 6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_simfleet
```



### 7.1 Development Lead

- Javi Palanca <jpalanca@gmail.com>

### 7.2 Contributors

- Jaume Jordan





### 8.1 1.0.1 (2019-11-07)

- SPADE and pandas version upgraded.
- Stop simulation fixed.
- Aesthetic changes.
- Minor bug fixes.
- Updated documentation.

### 8.2 1.0.0 (2019-11-05)

- Moved from a taxi simulator to a generic fleet simulator.
- Updated documentation.
- Added support for different cities.
- Directory agent now sends all the info.
- Fixed bug of staying the corresponding time in the station when charging.
- Changed logger to loguru library.
- Removed fuel from transport popup, now is current\_autonomy/max\_autonomy.
- Concurrent charging in stations now allowed through TimeoutBehavior.
- Custom icons added.
- Removed agents introduction from GUI.
- Added specific parameters in scenario file (now config file).
- CLI simplified.

- Changed cli to config file.
- Control of free places and status for StationAgent.
- Fuel refill behavior between TransportAgent and StationAgent.
- Refactoring from passenger to Customer
- Refactoring from taxi to Transport
- Refactoring from coordinator to fleet manager

### **8.3 0.4.1 (2019-01-07)**

- Fixed bug when checking if the simulation is finished.

### **8.4 0.4.0 (2018-10-25)**

- Improved the concurrent creation of agents.
- Added stop and clear buttons to the interface.
- Added download button for getting results in excel and json formats.
- Documentation updated.

### **8.5 0.3.0 (2018-10-01)**

- Migrated to SPADE 3.
- Documentation highly improved.
- Helper functions added and refined.
- Javascript framework included: VueJS
- Routes centralized with a Route agent.
- UI improved.

### **8.6 0.2 (2017-11-15)**

- Added scenario loading feature.

### **8.7 0.1.3 (2017-11-15)**

- Fixed minor bugs.

### **8.8 0.1.1 (2017-11-14)**

- Added documentation.

## 8.9 0.1.0 (2017-11-03)

- First release on PyPI.



## CHAPTER 9

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



---

## Bibliography

---

- [GangOfFour95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns, Elements of Reusable Object Oriented Software. Addison-Wesley, 1995.





### S

- `simfleet`, [62](#)
- `simfleet.cli`, [39](#)
- `simfleet.config`, [39](#)
- `simfleet.customer`, [40](#)
- `simfleet.directory`, [42](#)
- `simfleet.fleetmanager`, [43](#)
- `simfleet.helpers`, [45](#)
- `simfleet.protocol`, [45](#)
- `simfleet.route`, [46](#)
- `simfleet.simulator`, [46](#)
- `simfleet.station`, [53](#)
- `simfleet.strategies`, [55](#)
- `simfleet.strategies_fsm`, [55](#)
- `simfleet.transport`, [56](#)
- `simfleet.utils`, [60](#)



## A

- `accept_registration()` (*sim-fleet.fleetmanager.TransportRegistrationForFleetBehaviour* [47](#) *method*), [44](#)  
`accept_transport()` (*sim-fleet.customer.CustomerStrategyBehaviour* *method*), [41](#)  
`accept_transport()` (*sim-fleet.station.StationStrategyBehaviour* *method*), [54](#)  
`AcceptAlwaysStrategyBehaviour` (*class in sim-fleet.strategies*), [55](#)  
`AcceptFirstRequestBehaviour` (*class in sim-fleet.strategies*), [55](#)  
`add_customer()` (*simfleet.simulator.SimulatorAgent* *method*), [47](#)  
`add_manager()` (*simfleet.simulator.SimulatorAgent* *method*), [47](#)  
`add_service()` (*sim-fleet.directory.RegistrationBehaviour* *method*), [43](#)  
`add_station()` (*simfleet.simulator.SimulatorAgent* *method*), [47](#)  
`add_transport()` (*sim-fleet.fleetmanager.TransportRegistrationForFleetBehaviour* *method*), [44](#)  
`add_transport()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`all_customers_in_destination()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`AlreadyInDestination`, [45](#)  
`are_close()` (*in module simfleet.helpers*), [45](#)  
`arrived_to_destination()` (*sim-fleet.transport.TransportAgent* *method*), [56](#)  
`arrived_to_station()` (*sim-fleet.transport.TransportAgent* *method*), [56](#)  
`assigning_fleet_icon()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`assigning_place()` (*simfleet.station.StationAgent* *method*), [53](#)  
`async_start_agent()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`avg()` (*in module simfleet.utils*), [60](#)

## C

- `calculate_km_expense()` (*sim-fleet.transport.TransportAgent* *method*), [56](#)  
`cancel_customer()` (*sim-fleet.transport.TransportAgent* *method*), [57](#)  
`cancel_proposal()` (*sim-fleet.transport.TransportStrategyBehaviour* *method*), [59](#)  
`ChargeBehaviour` (*class in simfleet.station*), [53](#)  
`charging_complete()` (*sim-fleet.station.ChargeBehaviour* *method*), [53](#)  
`charging_transport()` (*sim-fleet.station.StationAgent* *method*), [53](#)  
`chunk_path()` (*in module simfleet.utils*), [61](#)  
`clean_controller()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`clear_agents()` (*sim-fleet.fleetmanager.FleetManagerAgent* *method*), [43](#)  
`clear_agents()` (*simfleet.simulator.SimulatorAgent* *method*), [47](#)  
`clear_stopped_agents()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)  
`collect_stats()` (*sim-fleet.simulator.SimulatorAgent* *method*), [47](#)

`create_customer_agent()` (*simfleet.simulator.SimulatorAgent* method), 47

`create_directory_agent()` (*simfleet.simulator.SimulatorAgent* method), 48

`create_fleetmanager_agent()` (*simfleet.simulator.SimulatorAgent* method), 48

`create_station_agent()` (*simfleet.simulator.SimulatorAgent* method), 48

`create_transport_agent()` (*simfleet.simulator.SimulatorAgent* method), 48

`customer_agents` (*simfleet.simulator.SimulatorAgent* attribute), 48

`CustomerAgent` (class in *simfleet.customer*), 40

`CustomerStrategyBehaviour` (class in *simfleet.customer*), 41

## D

`deassigning_place()` (*simfleet.station.StationAgent* method), 53

`DelegateRequestBehaviour` (class in *simfleet.strategies*), 55

`DirectoryAgent` (class in *simfleet.directory*), 42

`DirectoryStrategyBehaviour` (class in *simfleet.directory*), 42

`distance_in_meters()` (in module *simfleet.helpers*), 45

`download_stats_excel_controller()` (*simfleet.simulator.SimulatorAgent* method), 48

`download_stats_json_controller()` (*simfleet.simulator.SimulatorAgent* method), 48

`drop_customer()` (*simfleet.transport.TransportAgent* method), 57

`drop_station()` (*simfleet.transport.TransportAgent* method), 57

## E

`entities_controller()` (*simfleet.simulator.SimulatorAgent* method), 48

## F

`FleetManagerAgent` (class in *simfleet.fleetmanager*), 43

`FleetManagerStrategyBehaviour` (class in *simfleet.fleetmanager*), 44

`FSMTransportStrategyBehaviour` (class in *simfleet.strategies\_fsm*), 55

## G

`generate_tree()` (*simfleet.simulator.SimulatorAgent* method), 49

`get_autonomy()` (*simfleet.transport.TransportAgent* method), 57

`get_available_places()` (*simfleet.station.StationAgent* method), 53

`get_customer_stats()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_directory()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_manager_stats()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_pickup_time()` (*simfleet.customer.CustomerAgent* method), 40

`get_position()` (*simfleet.customer.CustomerAgent* method), 40

`get_position()` (*simfleet.station.StationAgent* method), 53

`get_position()` (*simfleet.transport.TransportAgent* method), 57

`get_power()` (*simfleet.station.StationAgent* method), 53

`get_route()` (*simfleet.route.RouteAgent* method), 46

`get_simulation_time()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_station_stats()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_stats()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_stats_dataframes()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_status()` (*simfleet.station.StationAgent* method), 53

`get_transport_agents()` (*simfleet.fleetmanager.FleetManagerStrategyBehaviour* method), 44

`get_transport_stats()` (*simfleet.simulator.SimulatorAgent* method), 50

`get_waiting_time()` (*simfleet.customer.CustomerAgent* method), 40

`go_to_the_station()` (*simfleet.transport.TransportStrategyBehaviour* method), 60

## H

`has_enough_autonomy()` (*simfleet.transport.TransportStrategyBehaviour method*), 60

`hide_passwords()` (*in module simfleet.config*), 39

## I

`index_controller()` (*simfleet.simulator.SimulatorAgent method*), 51

`inform_customer()` (*simfleet.transport.TransportAgent method*), 57

`inform_station()` (*simfleet.transport.TransportAgent method*), 57

`init_controller()` (*simfleet.simulator.SimulatorAgent method*), 51

`is_customer_in_transport()` (*simfleet.transport.TransportAgent method*), 57

`is_free()` (*simfleet.transport.TransportAgent method*), 57

`is_in_destination()` (*simfleet.customer.CustomerAgent method*), 40

`is_in_destination()` (*simfleet.transport.TransportAgent method*), 57

`is_simulation_finished()` (*simfleet.simulator.SimulatorAgent method*), 51

## K

`kmh_to_ms()` (*in module simfleet.helpers*), 45

## L

`load_cache()` (*simfleet.route.RouteAgent method*), 46

`load_class()` (*in module simfleet.utils*), 61

`load_config()` (*simfleet.config.SimfleetConfig method*), 39

`load_icons()` (*simfleet.simulator.SimulatorAgent method*), 51

`load_scenario()` (*simfleet.simulator.SimulatorAgent method*), 51

## M

`manager_agents` (*simfleet.simulator.SimulatorAgent attribute*), 51

`move_to()` (*simfleet.transport.TransportAgent method*), 57

## N

`needs_charging()` (*simfleet.transport.TransportAgent method*), 57

`num_customers` (*simfleet.config.SimfleetConfig attribute*), 39

`num_managers` (*simfleet.config.SimfleetConfig attribute*), 39

`num_stations` (*simfleet.config.SimfleetConfig attribute*), 39

`num_transport` (*simfleet.config.SimfleetConfig attribute*), 39

## O

`on_end()` (*simfleet.route.RouteAgent.RequestRouteBehaviour method*), 46

`on_start()` (*simfleet.customer.CustomerStrategyBehaviour method*), 41

`on_start()` (*simfleet.customer.TravelBehaviour method*), 42

`on_start()` (*simfleet.directory.DirectoryStrategyBehaviour method*), 42

`on_start()` (*simfleet.directory.RegistrationBehaviour method*), 43

`on_start()` (*simfleet.fleetmanager.FleetManagerStrategyBehaviour method*), 44

`on_start()` (*simfleet.fleetmanager.TransportRegistrationForFleetBehaviour method*), 44

`on_start()` (*simfleet.route.RouteAgent.RequestRouteBehaviour method*), 46

`on_start()` (*simfleet.station.RegistrationBehaviour method*), 53

`on_start()` (*simfleet.station.StationStrategyBehaviour method*), 54

`on_start()` (*simfleet.station.TravelBehaviour method*), 55

`on_start()` (*simfleet.strategies\_fsm.TransportMovingState method*), 55

`on_start()` (*simfleet.strategies\_fsm.TransportWaitingForApprovalState method*), 56

`on_start()` (*simfleet.strategies\_fsm.TransportWaitingState method*), 56

`on_start()` (*simfleet.transport.RegistrationBehaviour method*), 56

`on_start()` (*simfleet.transport.TransportStrategyBehaviour method*), 60

## P

`passenger_in_transport_callback()` (*in module simfleet.strategies\_fsm*), 56

`PathRequestException`, 45

`persist_cache()` (*simfleet.route.RouteAgent method*), 46

`pick_up_customer()` (*simfleet.transport.TransportStrategyBehaviour* method), 60

`print_stats()` (*simfleet.simulator.SimulatorAgent* method), 51

## R

`random_position()` (in module *simfleet.helpers*), 45

`refuse_transport()` (*simfleet.customer.CustomerStrategyBehaviour* method), 42

`refuse_transport()` (*simfleet.station.StationStrategyBehaviour* method), 55

`RegistrationBehaviour` (class in *simfleet.directory*), 43

`RegistrationBehaviour` (class in *simfleet.station*), 53

`RegistrationBehaviour` (class in *simfleet.transport*), 56

`reject_registration()` (*simfleet.fleetmanager.TransportRegistrationForFleetBehaviour* method), 44

`remove_service()` (*simfleet.directory.RegistrationBehaviour* method), 43

`remove_transport()` (*simfleet.fleetmanager.TransportRegistrationForFleetBehaviour* method), 44

`request_path()` (in module *simfleet.utils*), 61

`request_path()` (*simfleet.customer.CustomerAgent* method), 40

`request_path()` (*simfleet.simulator.SimulatorAgent* method), 51

`request_path()` (*simfleet.transport.TransportAgent* method), 57

`request_route_to_server()` (*simfleet.route.RouteAgent* static method), 46

`RequestRouteBehaviour` (class in *simfleet.utils*), 60

`RouteAgent` (class in *simfleet.route*), 46

`RouteAgent.RequestRouteBehaviour` (class in *simfleet.route*), 46

`run()` (*simfleet.customer.CustomerStrategyBehaviour* method), 42

`run()` (*simfleet.customer.TravelBehaviour* method), 42

`run()` (*simfleet.directory.DirectoryStrategyBehaviour* method), 42

`run()` (*simfleet.directory.RegistrationBehaviour* method), 43

`run()` (*simfleet.fleetmanager.FleetManagerStrategyBehaviour* method), 44

`run()` (*simfleet.fleetmanager.TransportRegistrationForFleetBehaviour* method), 44

`run()` (*simfleet.route.RouteAgent.RequestRouteBehaviour* method), 46

`run()` (*simfleet.simulator.SimulatorAgent* method), 51

`run()` (*simfleet.station.ChargeBehaviour* method), 53

`run()` (*simfleet.station.RegistrationBehaviour* method), 53

`run()` (*simfleet.station.StationStrategyBehaviour* method), 55

`run()` (*simfleet.station.TravelBehaviour* method), 55

`run()` (*simfleet.strategies.AcceptAlwaysStrategyBehaviour* method), 55

`run()` (*simfleet.strategies.AcceptFirstRequestBehaviour* method), 55

`run()` (*simfleet.strategies.DelegateRequestBehaviour* method), 55

`run()` (*simfleet.strategies\_fsm.TransportMovingState* method), 56

`run()` (*simfleet.strategies\_fsm.TransportWaitingForApprovalState* method), 56

`run()` (*simfleet.strategies\_fsm.TransportWaitingState* method), 56

`run()` (*simfleet.transport.RegistrationBehaviour* method), 56

`run()` (*simfleet.transport.TransportAgent.MovingBehaviour* method), 56

`run()` (*simfleet.transport.TransportStrategyBehaviour* method), 60

`run()` (*simfleet.utils.RequestRouteBehaviour* method), 60

`run_controller()` (*simfleet.simulator.SimulatorAgent* method), 51

`run_strategy()` (*simfleet.customer.CustomerAgent* method), 40

`run_strategy()` (*simfleet.directory.DirectoryAgent* method), 42

`run_strategy()` (*simfleet.fleetmanager.FleetManagerAgent* method), 43

`run_strategy()` (*simfleet.station.StationAgent* method), 53

`run_strategy()` (*simfleet.transport.TransportAgent* method), 58

## S

`send()` (*simfleet.transport.TransportAgent* method), 58

`send_confirmation()` (*simfleet.directory.RegistrationBehaviour* method), 43

`send_confirmation_travel()` (*simfleet.transport.TransportStrategyBehaviour* method), 60

`set_managers()` (*simfleet.customer.CustomerStrategyBehaviour* method), 42

*method*), 42  
 send\_get\_stations() (sim-  
     *fleet.transport.TransportStrategyBehaviour*  
     *method*), 60  
 send\_negative() (sim-  
     *fleet.directory.DirectoryStrategyBehaviour*  
     *method*), 43  
 send\_proposal() (sim-  
     *fleet.transport.TransportStrategyBehaviour*  
     *method*), 60  
 send\_registration() (sim-  
     *fleet.fleetmanager.FleetManagerStrategyBehaviour*  
     *method*), 44  
 send\_registration() (sim-  
     *fleet.station.RegistrationBehaviour* *method*),  
     53  
 send\_registration() (sim-  
     *fleet.transport.RegistrationBehaviour* *method*),  
     56  
 send\_request() (sim-  
     *fleet.customer.CustomerStrategyBehaviour*  
     *method*), 42  
 send\_services() (sim-  
     *fleet.directory.DirectoryStrategyBehaviour*  
     *method*), 43  
 set\_autonomy() (simfleet.transport.TransportAgent  
     *method*), 58  
 set\_available\_places() (sim-  
     *fleet.station.StationAgent* *method*), 53  
 set\_default\_strategies() (sim-  
     *fleet.simulator.SimulatorAgent* *method*),  
     51  
 set\_directory() (simfleet.customer.CustomerAgent  
     *method*), 40  
 set\_directory() (sim-  
     *fleet.fleetmanager.FleetManagerAgent*  
     *method*), 43  
 set\_directory() (sim-  
     *fleet.simulator.SimulatorAgent* *method*),  
     52  
 set\_directory() (simfleet.station.StationAgent  
     *method*), 54  
 set\_directory() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_fleet\_type() (sim-  
     *fleet.customer.CustomerAgent* *method*), 40  
 set\_fleet\_type() (sim-  
     *fleet.fleetmanager.FleetManagerAgent*  
     *method*), 43  
 set\_fleet\_type() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_fleetmanager() (sim-  
     *fleet.customer.CustomerAgent* *method*), 40  
 set\_fleetmanager() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_icon() (simfleet.customer.CustomerAgent  
     *method*), 40  
 set\_icon() (simfleet.fleetmanager.FleetManagerAgent  
     *method*), 43  
 set\_icon() (simfleet.simulator.SimulatorAgent  
     *method*), 52  
 set\_icon() (simfleet.station.StationAgent *method*), 54  
 set\_icon() (simfleet.transport.TransportAgent  
     *method*), 58  
 set\_id() (simfleet.customer.CustomerAgent *method*),  
     40  
 set\_id() (simfleet.directory.DirectoryAgent *method*),  
     42  
 set\_id() (simfleet.fleetmanager.FleetManagerAgent  
     *method*), 43  
 set\_id() (simfleet.station.StationAgent *method*), 54  
 set\_id() (simfleet.transport.TransportAgent *method*),  
     58  
 set\_initial\_position() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_km\_expense() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_position() (simfleet.customer.CustomerAgent  
     *method*), 40  
 set\_position() (simfleet.station.StationAgent  
     *method*), 54  
 set\_position() (simfleet.transport.TransportAgent  
     *method*), 58  
 set\_power() (simfleet.station.StationAgent *method*),  
     54  
 set\_registration() (sim-  
     *fleet.fleetmanager.FleetManagerAgent*  
     *method*), 44  
 set\_registration() (sim-  
     *fleet.station.RegistrationBehaviour* *method*),  
     53  
 set\_registration() (simfleet.station.StationAgent  
     *method*), 54  
 set\_registration() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_route\_agent() (sim-  
     *fleet.customer.CustomerAgent* *method*), 41  
 set\_route\_agent() (sim-  
     *fleet.transport.TransportAgent* *method*),  
     58  
 set\_speed() (simfleet.transport.TransportAgent  
     *method*), 58



set\_status() (*simfleet.station.StationAgent* method), 54  
 set\_target\_position() (*simfleet.customer.CustomerAgent* method), 41  
 set\_type() (*simfleet.station.StationAgent* method), 54  
 setup() (*simfleet.customer.CustomerAgent* method), 41  
 setup() (*simfleet.directory.DirectoryAgent* method), 42  
 setup() (*simfleet.fleetmanager.FleetManagerAgent* method), 44  
 setup() (*simfleet.route.RouteAgent* method), 46  
 setup() (*simfleet.simulator.SimulatorAgent* method), 52  
 setup() (*simfleet.station.StationAgent* method), 54  
 setup() (*simfleet.strategies\_fsm.FSMTransportStrategyBehaviour* method), 55  
 setup() (*simfleet.transport.TransportAgent* method), 58  
 simfleet (module), 62  
 simfleet.cli (module), 39  
 simfleet.config (module), 39  
 simfleet.customer (module), 40  
 simfleet.directory (module), 42  
 simfleet.fleetmanager (module), 43  
 simfleet.helpers (module), 45  
 simfleet.protocol (module), 45  
 simfleet.route (module), 46  
 simfleet.simulator (module), 46  
 simfleet.station (module), 53  
 simfleet.strategies (module), 55  
 simfleet.strategies\_fsm (module), 55  
 simfleet.transport (module), 56  
 simfleet.utils (module), 60  
 SimfleetConfig (class in *simfleet.config*), 39  
 SimulatorAgent (class in *simfleet.simulator*), 46  
 station\_agents (*simfleet.simulator.SimulatorAgent* attribute), 52  
 StationAgent (class in *simfleet.station*), 53  
 StationStrategyBehaviour (class in *simfleet.station*), 54  
 status\_to\_str() (in module *simfleet.utils*), 61  
 step() (*simfleet.transport.TransportAgent* method), 59  
 stop() (*simfleet.simulator.SimulatorAgent* method), 52  
 stop\_agents() (*simfleet.simulator.SimulatorAgent* method), 52  
 stop\_agents\_controller() (*simfleet.simulator.SimulatorAgent* method), 52  
 StrategyBehaviour (class in *simfleet.utils*), 60

to\_json() (*simfleet.customer.CustomerAgent* method), 41  
 to\_json() (*simfleet.station.StationAgent* method), 54  
 to\_json() (*simfleet.transport.TransportAgent* method), 59  
 total\_time() (*simfleet.customer.CustomerAgent* method), 41  
 transport\_agents (*simfleet.simulator.SimulatorAgent* attribute), 52  
 transport\_charged() (*simfleet.transport.TransportAgent* method), 59  
 TransportAgent (class in *simfleet.transport*), 56  
 TransportAgent.MovingBehaviour (class in *simfleet.transport*), 56  
 TransportMovingState (class in *simfleet.strategies\_fsm*), 55  
 TransportRegistrationForFleetBehaviour (class in *simfleet.fleetmanager*), 44  
 TransportStrategyBehaviour (class in *simfleet.transport*), 59  
 TransportWaitingForApprovalState (class in *simfleet.strategies\_fsm*), 56  
 TransportWaitingState (class in *simfleet.strategies\_fsm*), 56  
 TravelBehaviour (class in *simfleet.customer*), 42  
 TravelBehaviour (class in *simfleet.station*), 55

U

unused\_port() (in module *simfleet.utils*), 62

W

watch\_value() (*simfleet.transport.TransportAgent* method), 59  
 write\_excel() (*simfleet.simulator.SimulatorAgent* method), 52  
 write\_file() (*simfleet.simulator.SimulatorAgent* method), 52  
 write\_json() (*simfleet.simulator.SimulatorAgent* method), 53

## T

time\_is\_out() (*simfleet.simulator.SimulatorAgent* method), 52