D4.3 – FINAL CPS MODELLING LIBRARY

Deliverable ID	**D4.3**
Deliverable Title	**Final CPS modelling library**
Work Package	**WP4 – Models and algorithms for CPS Swarms**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**02-01-2020**
Status	**Final**
Lead Editor	**Etienne Brosse (SOFTEAM)**
Main Contributors	**Alessandra Bagnato (SOFTEAM), Melanie Schranz, Micha Rappaport (LAKE), René Reiners (FRAUNHOFER)**

Published by the CPSwarm Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731946.

Document History

Version	Date	Author(s)	Description
0.1	2019-07-15	Etienne Brosse (SOFTEAM)	First Table of Content.
0.2	2019-09-06	Melanie Schranz (LAKE)	Add Design Pattern Approach
0.3	2019-09-10	Micha Rappaport (LAKE)	Add Modelling Library Concepts
0.4	2019-10-07	Melanie Schranz (LAKE) Rene Reiners (FRAUNHOFER)	Clean Design Pattern Approach
0.5	2019-10-10	Etienne Brosse (SOFTEAM)	Polishing and cleaning.
0.5.1	2019-10-07	Rene Reiners (FRAUNHOFER)	Clean Design Pattern Approach
0.6	2019-10-10	Etienne Brosse (SOFTEAM)	Polishing and cleaning.
1.0	2020-01-02	Etienne Brosse (SOFTEAM)	Final version

Internal Review History

Review Date	Reviewer	Summary of Comments
2019-12-19	Farshid Tavakolizadeh (FRAUNHOFER)	Comments and minor modifications
2019-12-19	Gianluca Prato (LINKS)	Minor modifications introduced

1 Executive summary

This deliverable, namely "D4.3 - Final CPS modelling library", is a deliverable of the CPSwarm project, funded by the European Commission's Directorate- General for Research and Innovation (DG RTD), under its Horizon 2020 Research and innovation program (H2020).

CPSwarm main's goal consists on developing a workbench that aims to fully design, develop, and validate swarm solution. In this project, Work Package 4 focuses on how CPS swarm can be model. Which aspects are needed? What concepts must be depicted? Which formalizes are the more useful? etc. To illustrate and distribute the result of this research, a set of models are publicly available. These models depict CPS swarm through specific aspect (Hardware architecture, swarm member behaviour, internal interaction, external interaction). Finally, this deliverable shows the different models designed for Cyber-Physical Systems (CPS) Swarm design at M35 of the CPSwarm project.

Table of Contents

2 Introduction

D4.3 – "Final CPS modelling library" is a public document describing the publicly available CPS models designed till M35 in the CPSwarm project.
SOFTEAM, as deliverable leader, initially drafted the document, which has subsequently been enriched by all partners' contributions with existing publicly available CPSs models.

2.1 Scope

This deliverable provides a description of the current models specified within CPSwarm project during preliminary or study phase but also within the three CPSwarm case studies. Based on previous studies – published in D4.1 and D4.2- the models depicted here represent the three relevant aspects of a CPS swarm i.e. CPS designs (Hardware and Behaviour description), the Communication between CPSs and CPS/Human interaction. Models or part of these models of these aspects have been designed and publicly published inside libraries.

2.2 Document organization

The remainder of this deliverable is organized as follows:
Section 3 describes the CPSwarm modelling library, which provides - among other - a set of elements for CPS modelling. Section 4 presents four CPS designs made on top of the CPSwarm modelling library, presented in the previous section. In CPSwarm project, CPS modelling do not only deal with the CPS description, but also with the concept around them. The Human aspect modelling is depicted in Section 5. Finally, Section 6 draws conclusions.

2.3 Related documents

ID	Title	Reference	Version	Date
[D4.1]	Initial CPS Modeling Library	D4.1	1.0	2017-10-06
[D4.2]	Updated CPS Modeling Library	D4.2	1.0	2018-10-06
[D5.3]	Updated CPSwarm Modelling Tool	D5.3	1.0	2018-06-30

3 CPSwarm Modelling library

The modelling of the CPSs considers the hardware as well as their behaviour of the CPSs.

3.1 Hardware Components

One main aspect of modeling a CPS consist in specifying its architecture in terms of hardware components. In CPSwarm context, three kind of hardware components have been defined. This specification is made in two steps: First, the list of internal components (which can be a controller, a sensor, or an actuator component) must be defined. Each of this internal component must expose the data it provides or requires. Figure 1 represents two simple hardware components respectively named CNY70 and Locomotion. The CNY70 is a light sensor providing the light level as a double. The Locomotion actuator needs a 2D Pos to be able to move to the specified location.

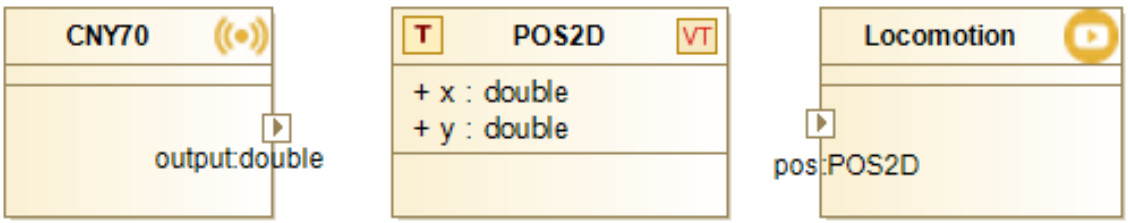

Figure 1: Models of different types Hardware Components.

The second steps in modelling the CPS hardware architecture consists in instantiate each appropriate component and connect them between each other. In Figure 2, the components predefined previously has been instanced twice and each port has been connected to model the data flow between the internal components.

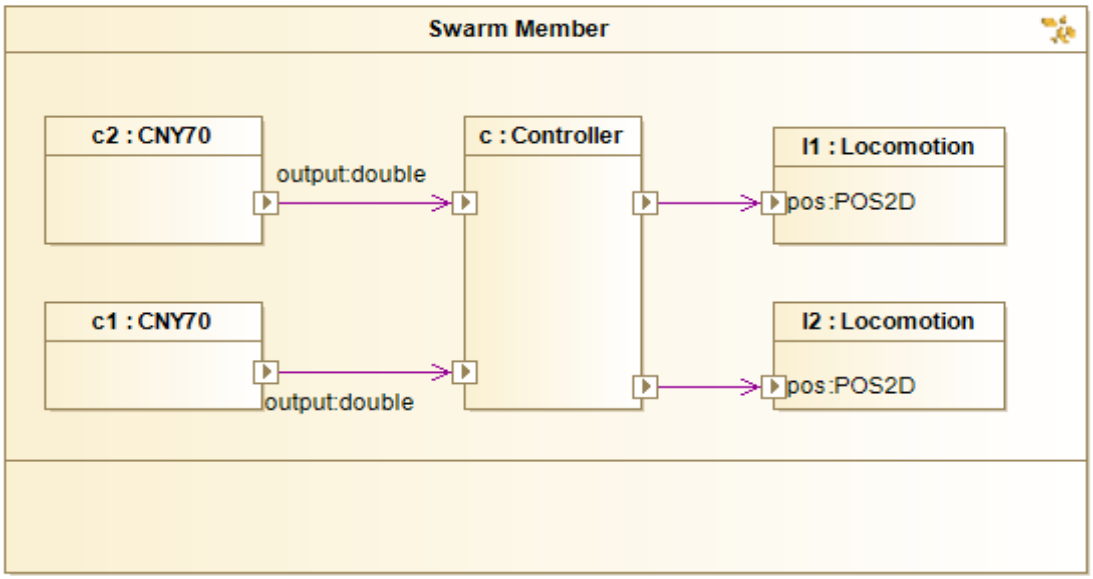

Figure 2: Simple Hardware CPS.

3.2 Software Components

The software component models define the behavior of each CPS. They are modeled in order to describe how the CPS behaves by interacting with the environment and the other CPSs in the swarm. The behaviors are defined in a way that the CPS swarm completes the mission that is intended by the designer of the swarm system. Designing the behavior of individual agents in a swarm is a difficult task because the emergent swarm behavior that should complete a mission effectively is not easily predictable. By modeling the behavior on an abstract level facilitates the process by allowing to execute it on different realism levels. This bottom-up approach allows to iteratively refine the individual behaviors until a global swarm behavior is reached that completes the mission effectively. The formal behavior models allow to speed up the design process by automatically generating the code to be executed on the CPSs, either in simulation or in real-world experiments.
The behavior models are placed within different libraries that contain software artifacts. They are modeled as the states of FSMs. This allows to build a modular open source repository of swarm behaviors to be executed on different type of CPSs. An initial version of the libraries will be available on the CPSwarm Github repository[1]. On the one hand, it features different platform independent swarm behaviors in the Swarm Library (see Deliverable 4.6). On the other hand, it features ROS based Abstraction Libraries for UAVs using the MAVLink protocol[2] and UGVs using the ROS navigation stack[3].

3.3 Abstraction Library

The Abstraction Library allows to access the hardware provided by the CPS. It guarantees the support of controlling several types of sensors such as ultrasonic range sensors, cameras, or GPS, and driving actuators such as grippers, motors and servos. It raises the level of abstraction from a platform-dependent point of view to an application-oriented perspective. Furthermore, the Abstraction Library provides facilities to easily develop high-level routines. It shifts the focus of the developer from coding CPS specific implementations to swarm behavioral executions. This allows to concentrate on describing how the CPSs should behave in order to complete a high-level task or reach an application-specific goal. This is achieved by the Abstraction Library by providing a set of CPS-specific adaptation libraries in order to access platform-specific information of a CPS in a standard and coherent way.

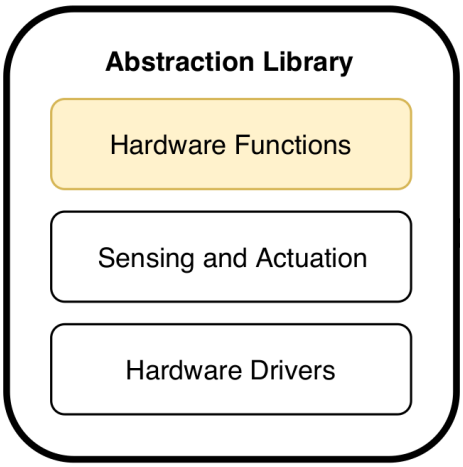

Figure 3: The abstraction library structure.

To achieve this, the Abstraction Library is organized as a composition of three layers as shown in Figure 3 where each layer adds a level of hardware abstraction. First, the bottom most layer of Hardware Drivers gathers the software libraries that are responsible to enable the other layers to access the hardware functionalities. This

1 https://github.com/cpswarm

2 https://mavlink.io

3 https://wiki.ros.org/navigation

layer constitutes the foundation of the Abstraction Library and includes all the drivers for sensors and actuators that are mounted on the CPSs. Second, the Sensing and Actuation layer is responsible for providing sensor information and for controlling the CPSs using their actuators. While the Hardware Drivers layer has a direct connection with the hardware, this layer purely consists of software that contributes to supply a first degree of abstraction by realizing complex functionalities required by the overlying layer. Depending on the available computational power, memory, and hardware resources, this layer can be presented as an independent component or incorporated inside the Hardware Drivers layer. Finally, the topmost layer of hardware functions exhibits a set of high-level functions corresponding to complex routines that a CPS can execute involving a set of sensors and actuators. Each function interacts with the lower layers for sending actuator commands and requesting sensor information. Each function of this layer constitutes a base building block to define a state of the FSMs. A single state can be associated to a specific function of the Abstraction Library that will be executed while the state is active. Therefore, this layer supports two essential features. First, the application of model-driven techniques based on the design of FSMs that will speed up the development process to realize new CPSs behaviors. Second, the mapping of CPS's functionalities to specific software modules that can guarantee the reusability of those functionalities. The hardware functions are defined as UML simple states to be used in the complex behavior FSMs.

3.4 Communication Library

3.4.1 Description

The Communication Library provides a unified interface tools and swarm members can use to interact with each other. It is the duty of the library to ensure that all communications happen with the desired reliability, security level and latency. After evaluating the requirements established by our core use cases and the design goals of a swarm in general, we concluded that interactions have a well-defined set of primitives and actions:

- Swarm members need to be discoverable on the network
- Events and commands need to be sent and received
- Parameters need to be remotely adjustable
- Telemetry needs to be sent back to operators and other subscribers

Communication Library aim is to provide a stable API for all tools that abstracts away the physical layer and the authentication scheme used. To do this, a pluggable architecture was designed for the Communication Library, which separates the logical layer responsible for implementing these primitives and the endpoint implementation capable of sending individual messages over the network. This extensible infrastructure makes it possible to add support for new low-level protocols, physical layers and security schemes without affecting the rest of the system. As a first step, the Zyre protocol was integrated with the library, but as the project progresses, a secure endpoint will be added as well.

The key concepts of the Communication Library are the following:

- High level C++ API;
- Abstracts away the physical and transport layers;
- Responsible for reliable delivery and fault detection;
- Exposes functionality through services;
- Protobuf based serialization;
- The API works with Protobuf objects directly;
- Objects can be reserialized at any points;
- Complex data types can be defined (area, route, etc.);
- Cross-platform;
- Uses only C++ standard library primitives and other cross-platform libraries;
- Compatible with ROS.

So, the Communication Library is used to abstract away the transport and physical layer. Endpoint implementations based on BasicEndpoint only need to implement:

- Starting and stopping the endpoint

- Sending binary messages
- Receiving binary messages
- Tracking the presence of nodes

While we are targeting IP networks (including mesh networks), the library doesn't care about the medium. Zyre based implementation is available now.

Since our primary targets are ROS based devices, support for ROS native facilities needed to be a lot more in-depth. Communications within ROS use a proprietary messaging format and protocol – support for this is not available on non-ROS systems. A bridge node was developed, which can translate between a ROS based system and the rest of the world:

- Publishing any ROS topic as telemetry
- Forwarding events to and from the behavior
- Setting parameters on the ROS Parameter Server
- Using the communication node, applications developed or behaviour generated for ROS based devices can use native ROS facilities and need not care about the presence of the library. The bridge is just another application using the library – it receives no special treatment.
- Using standard ROS facilities to communicate
- Bridge the Key-Value Service to ROS Parameter Server
- Transfer events and telemetry through ROS publish-subscribe
- Bind to ROS resources as defined in a configuration file
- Should be part of the deployment package
- Reloadable without interruption
- Should be one of the first things to install on a node during provisioning
- Cryptographic proof of swarm membership will need to be established
- Network interfaces need to be configured

3.4.2 Modelling

At the modelling level, Communication Library key concepts must be implemented inside CPSwarm modelling library in order to able to generate communication library configuration. For example, in Figure 4 represents a simple drone in which "*reportInterval*" property of type "*UInt*" is published to the whole world. A "Simple Event" named "*launch*" and a "*LocalTargetPositionEvent*" named "*target*_found" are respectively specified as incoming and outcoming events.

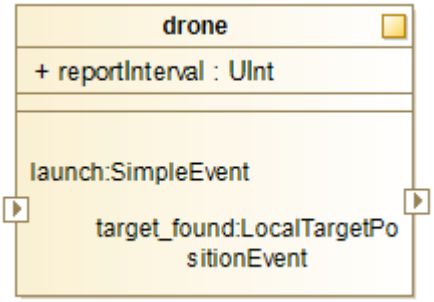

Figure 4: Modelling usage of CPSwarm communication library.

The resulting communication configuration, generated from the previous Figure, is available in Annex A.

4 Available CPS Models

This section describes the status of three CPSs designed on top of the CPSwarm modelling library (Section 3). Each model presents a different state of maturity mainly depending of its complexity and level of detail needed but each of them has been publicly published inside one of the modelling project available at http://forge.modelio.org/projects/cpswarm-modelio38/files. The first one is called "Spiderino" which has been modelled at the beginning of CPSwarm project with a focus on its hardware aspect. Two other CPS named "Drone" and "Rover", which are CPSs used in "Search And Rescue" case study are described in more detail and both Hardware and Behaviour aspects have been sketched. Note that these two models will continue to evolve during CPSwarm project.

4.1 Spiderino

The Spiderino is a low-cost robot for research and educational purposes. As shown in Figure 5, Spiderino is a small spider robot equipped with several embedded sensors.

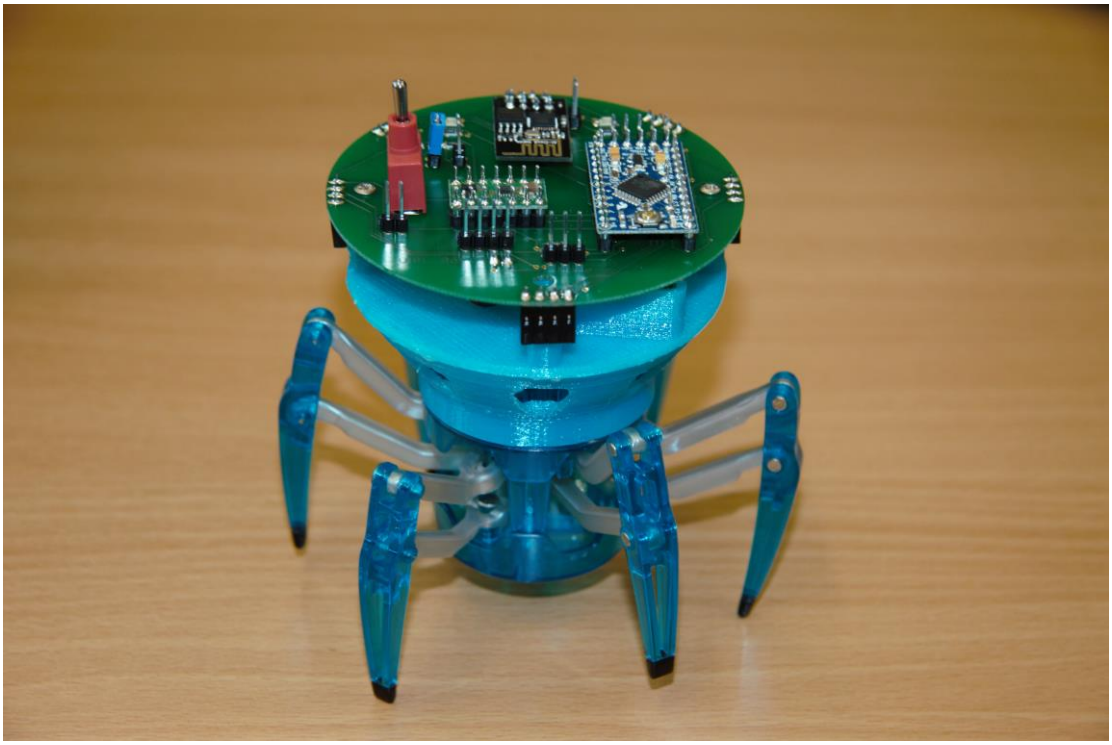

Figure 5: Spiderino robot

The main goal of this model was to test and evaluate CPSwarm Hardware design facilities in a simple project. The result of this designing activity is presented in the following section.

4.1.1 Hardware Design

As presented in Figure 6, a Spiderino is composed of five light sensors – respectively named rs1, rs2, rs3, rs4 and rs5 – of type CNY70. Two motors or locomotion components are also present as a GP2D2 sensor able to calculate the distance between a Spiderino and its environment. Finally, the BEECLUST component holds the Swarm Algorithm/ CPS behaviour which consists in a UML State Machine implementation of one Bee Algorithm.

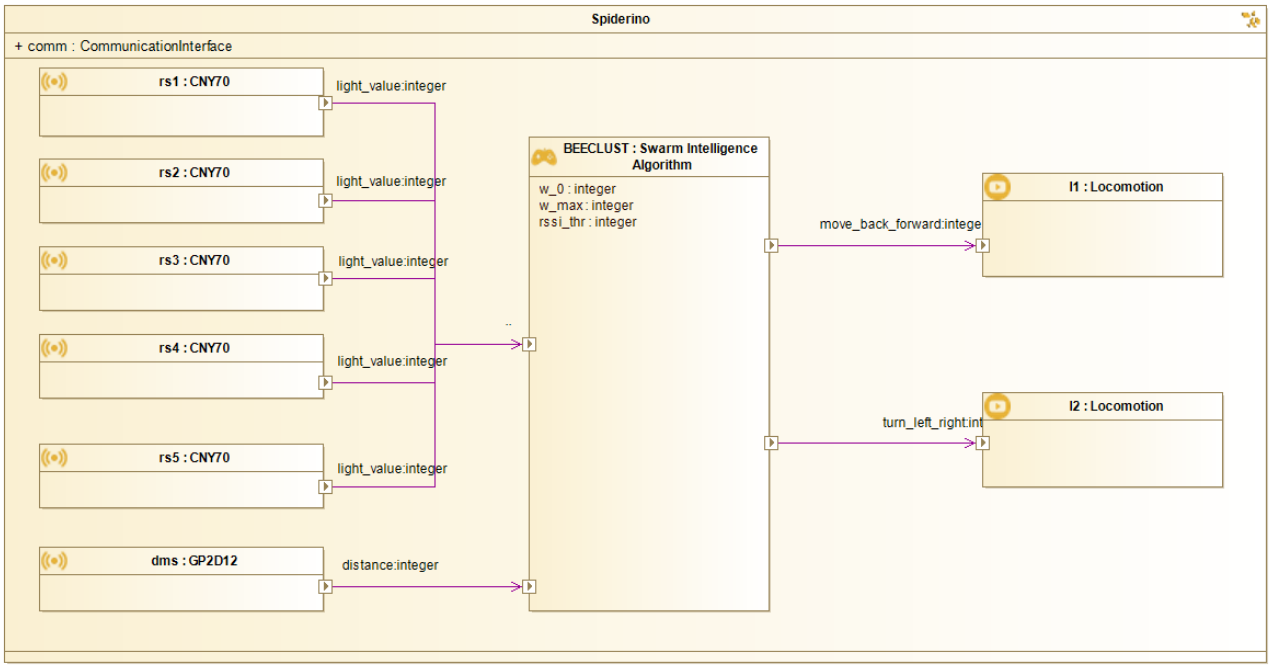

Figure 6: Spiderino hardware architecture

4.2 Drone

Drones in SAR missions are aiming to find target(s) by covering a predefined area. To do so a drone needs to embed different components, described in 4.2.1.

4.2.1 Hardware design

The Drone platform selected for the implementation of the SAR demonstration, shown in Figure 7, is composed by the following devices:

- One PX4 flight stack as a complete Flight Controller solution;
- One nano-pi board as Companion Computer to run the CPSwarm Abstraction Layer with the relevant Library;
- One LiPo battery;
- One Telemetry Radio;
- Three Sonars to avoid collision with another drone or obstacle;
- One "beacon" for measuring the global position of the rover. Two options are available:
  - An Ultra-Wide Band (UWB) node to measure the current position in indoor environments;
  - A Global Positioning System (GPS) module to measure the current position in outdoor environments;
- Four Motor to make the rover able to move;
- One Communication Interface for the communication with the other Rovers and Drones, and the Monitoring Tool;
- One CMOS OV5640 camera: to find target to rescue.

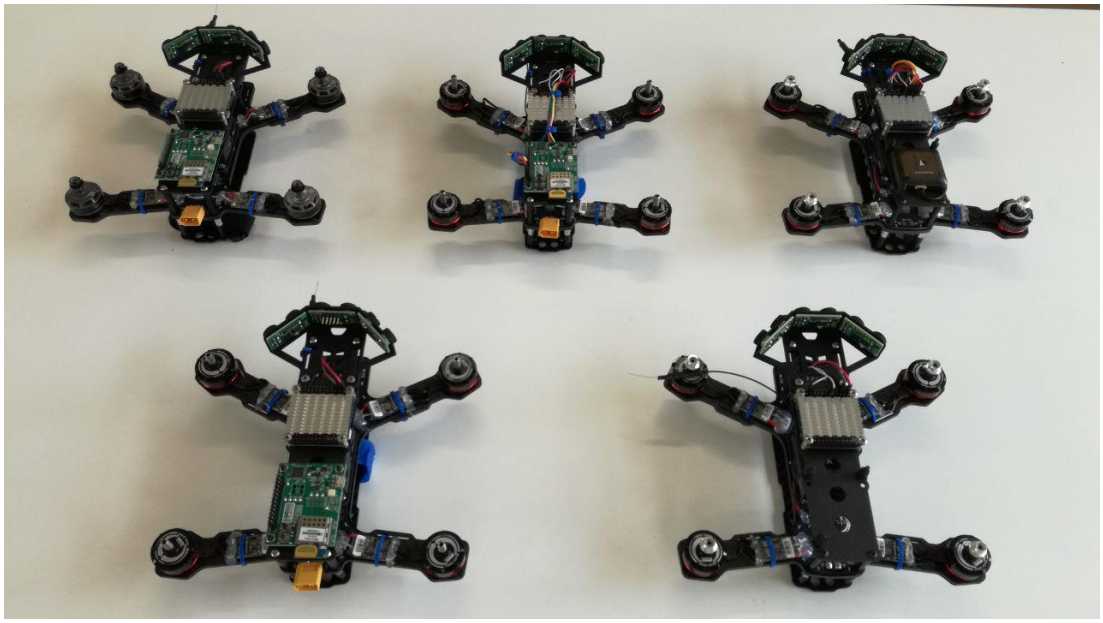

Figure 7: Drone platforms

The following model -cf. Figure 8- is an abstraction view of the Drone hardware described previously. The model depicts:

- The SAR_*Behaviour* component holding CPS behaviour.
- Three sonars, named *ps1*, *ps2* and *ps3*, which gives the distance to the closest object,
- One camera measuring the distance,
- One UWB giving the local position of the device,
- Two locomotion components respectively in charge of moving -forward or backward - and turning - left or right.

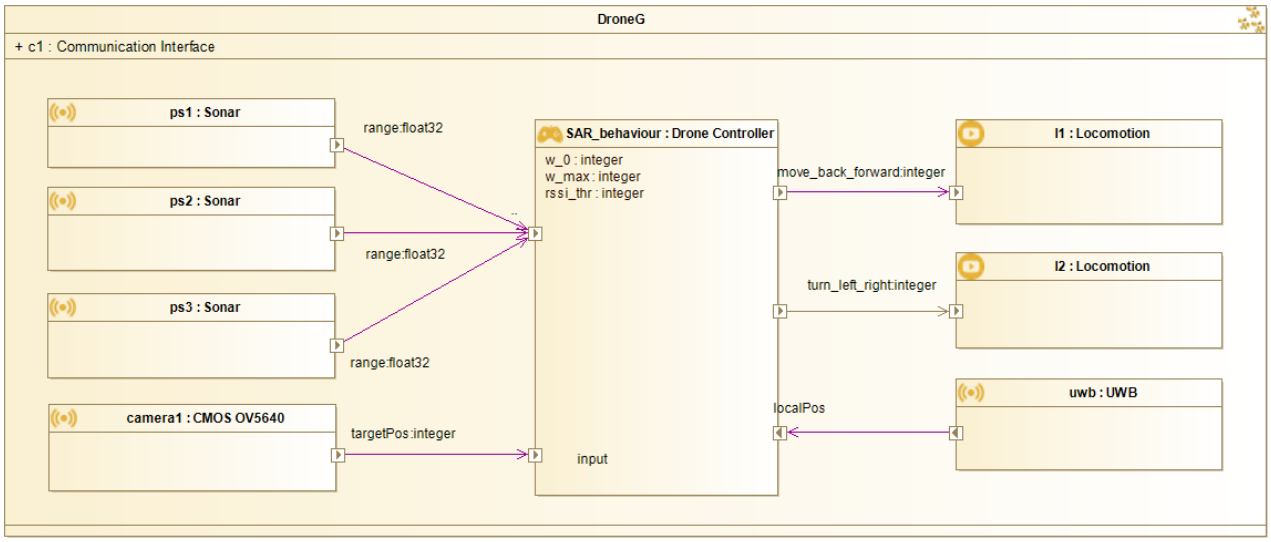

Figure 8: Drone hardware description

4.2.2 Behaviour modelling

The main goal of a Drone is to find target by firstly covering dedicated space. Figure 9 is a UML state Machine representation of how the Drone will achieve its goal. As for the rover, the drone behaviour starts by the "StartUp" State before the "Idle" one. The mission is started once the drone receives a signal from the monitoring tool (outside the swarm). The Drone takes off and start covering dedicated space. If the drone finds a target, it broadcast the event to the Rovers, which negotiate among them which one has to be assigned to reach the target. The drone keeps tracking the target until it receives a rescued signal from the assigned rover. Then the drone restarts to look for a new target.

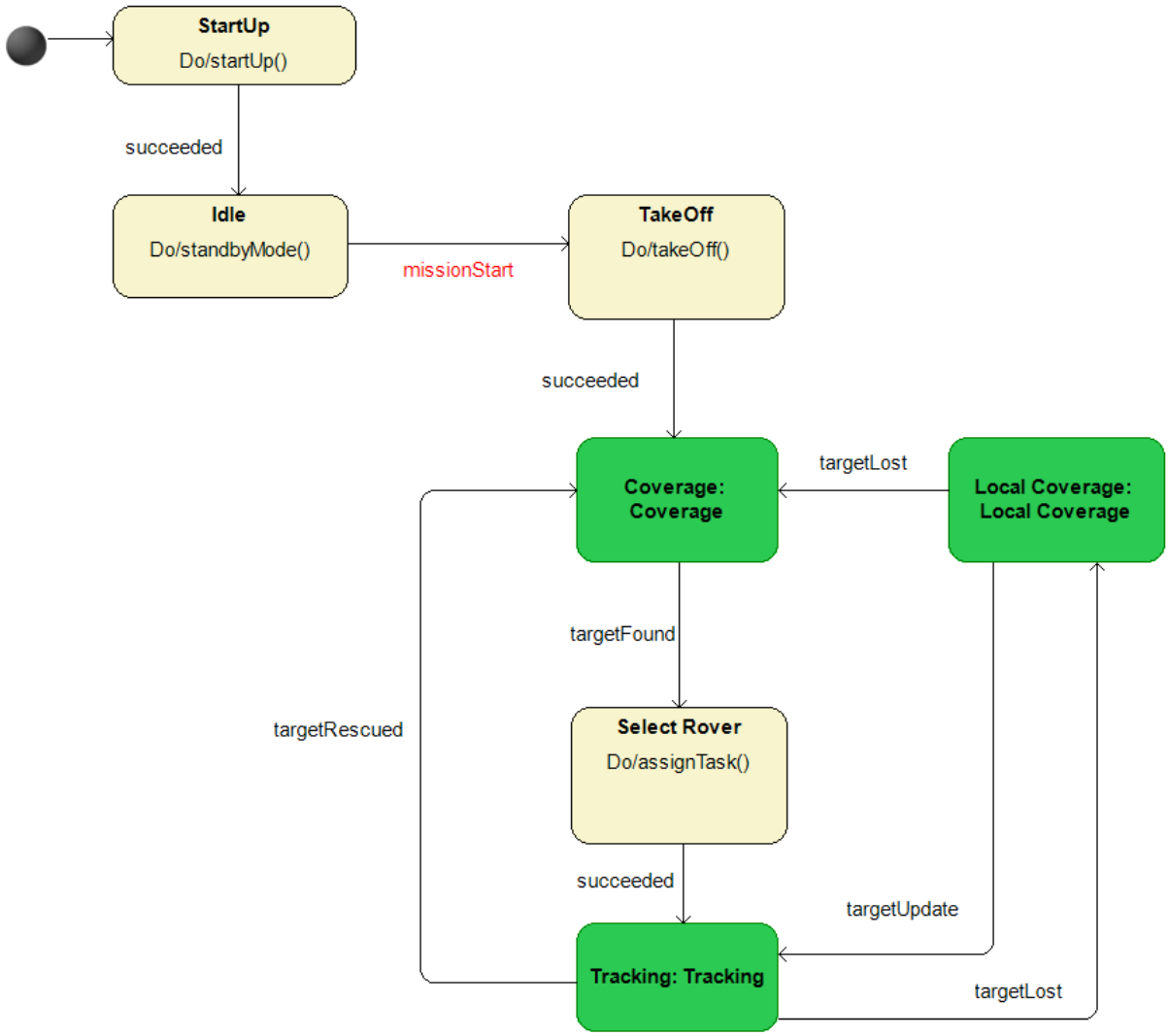

Figure 9: Drone behaviour

4.3 Rover

Heterogeneous swarms of ground robots/rovers and drones are considered within the CPSwarm project to conduct certain missions, such as in Search and Rescue (SAR) tasks. The current section shows both Hardware and Behaviour aspects of a rover/ground robot.

4.3.1 Hardware Design

The Rover platform selected for the implementation of the SAR demonstration, shown in Figure 10, is composed by the following devices:

- One Pixhawk Flight Controller with ArduPilot Autopilot Software Suite;
- One nano-pi board as Companion Computer to run the CPSwarm Abstraction Layer with the relevant Library;
- One LiPo battery;
- One Telemetry Radio;
- One Sonar to avoid collision with other rovers or obstacles;
- One "beacon" for measuring the global position of the rover. Two options are available:
    - An Ultra-Wide Band (UWB) node to measure the current position in indoor environments;
    - A Global Positioning System (GPS) module to measure the current position in outdoor environments;
- One Motor to make the rover able to move;
- One Communication Interface for the communication with the other Rovers and Drones, and the Monitoring Tool.

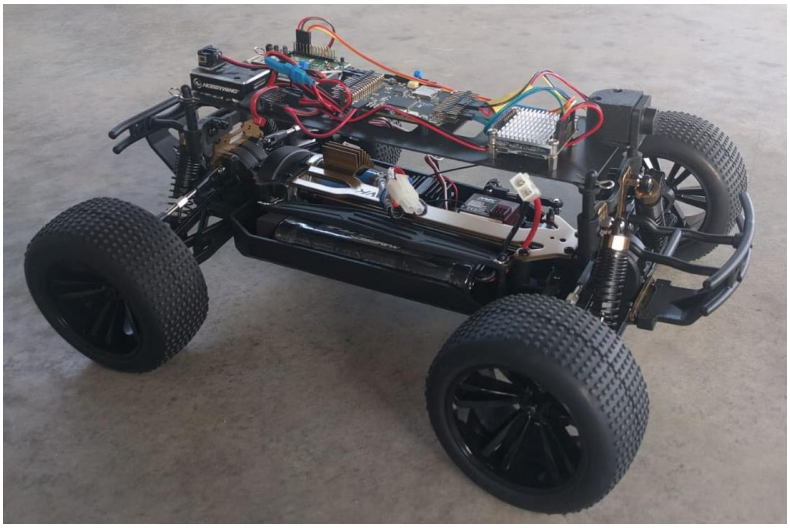

Figure 10: Rover platform

The following model -cf. Figure 11- is an abstraction view of the Drone hardware described previously. The model depicts:

- The *rover behaviour* component holds the CPS behaviour.
- The *s* sonar that gives the distance to the object,
- One UWB giving the local position of the device,
- one motor components respectively in charge of moving -forward or backward - and turning - left or right.

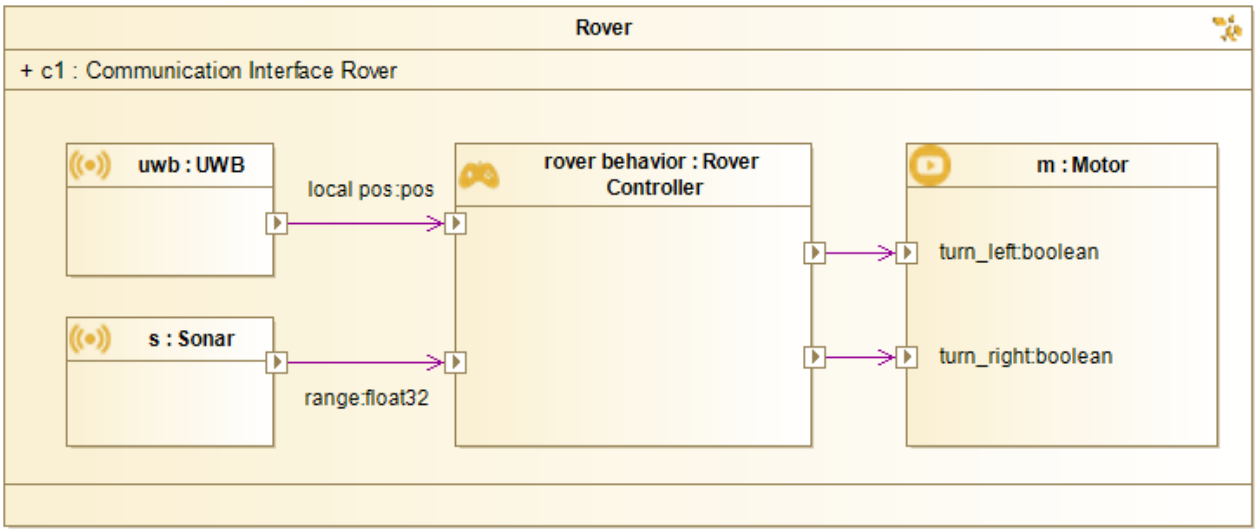

Figure 11: Rover hardware description

4.3.2 Behaviour Modelling

Rovers in SAR missions are aiming to guide a target to an exit. To do so the behaviour shown in Figure 12 has been made. Firstly, the rover is started and passes to the "Idle" mode waiting for a Drone to find a target. Once a target is found by a Drone, the assigned rover has to "move to the target". Then it guides the assigned target to an exit before coming back to the "Idle" state waiting another target to be found.

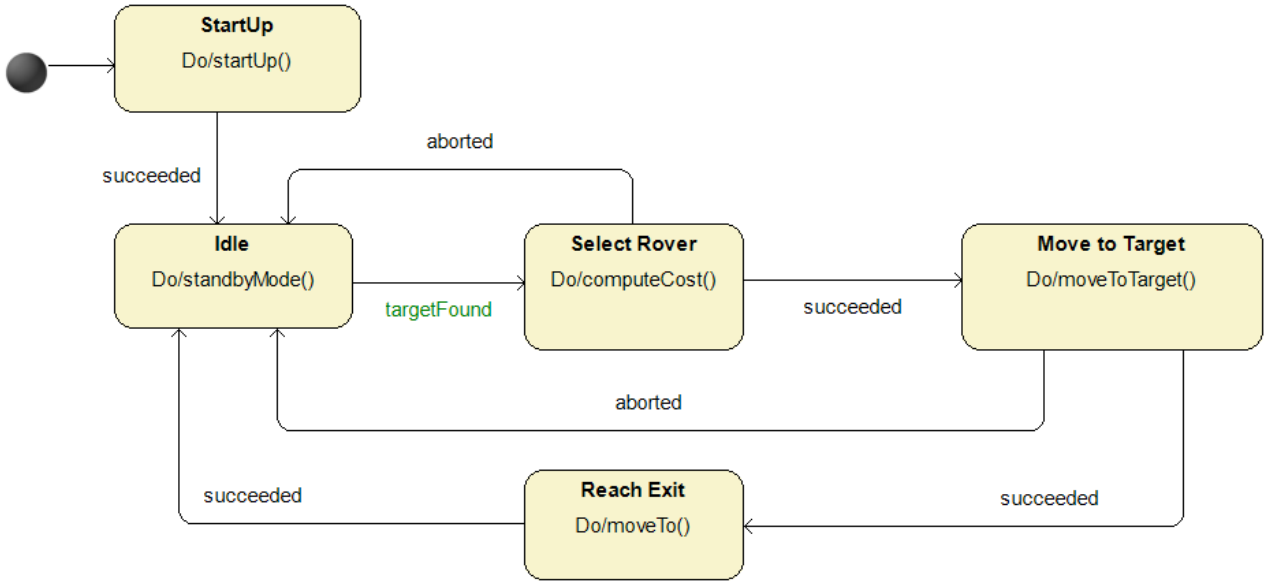

Figure 12 Rover behaviour

4.4 Turtlebot

Heterogeneity in swarm of CPS does not mean that each type of swarm member must be completely different from the others. Swarm member can share common hardware structure and behaviour parts. In Logistics scenario, two type of CPS, named scout and worker, have some modelling component in common. The current section shows both Hardware and Behaviour aspects of a scout CPS which are specialized from general turtlebot platform.

4.4.1 Hardware Design

The components that make up the hardware architecture, shown in Figure 13, are:

- A turtlebot 2 with a kobuki base and three heights of hexagons;
- An Intel NUC i5-8 Gb of RAM and 128GB SSD. Dual band Wireless (802.11ac) and Bluetooth 4.2;
- A RPLidar A2 sonar with 360 degrees, 12-18m range detection and 8000 Samples per time);
- A FLIR Chameleon CM3.camera for scouts or a SKF cahb 10 linear actuator (elevator) for workers.

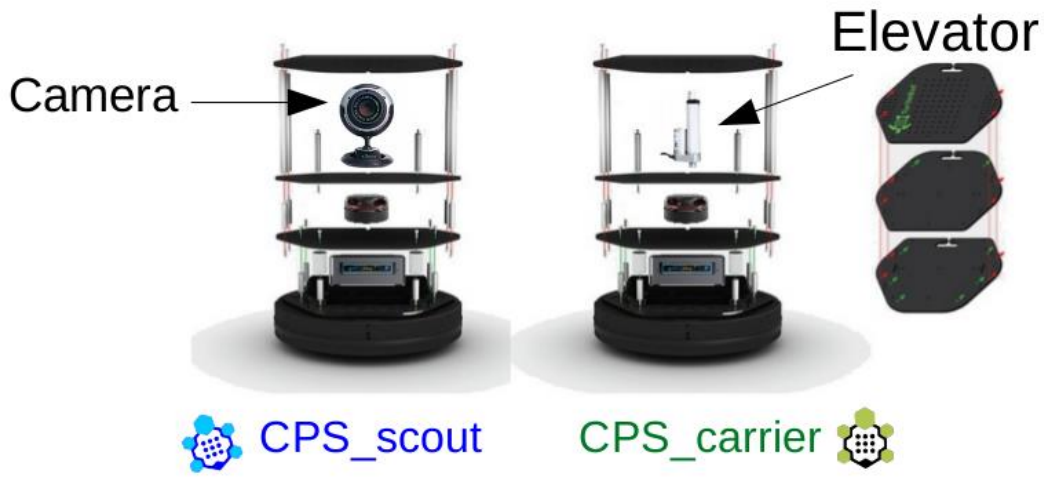

Figure 13: Turtlebot platform

The following model -cf. Figure 14 - is an abstraction view of the scout hardware described previously. The model depicts:

- The Scout Controller component holding CPS behaviour;
- The RPLidar A2 sonar that gives the distance to the closest object;
- One FLIR Chameleon camera scanning QR code;
- One locomotion components in charge of moving the CPS.

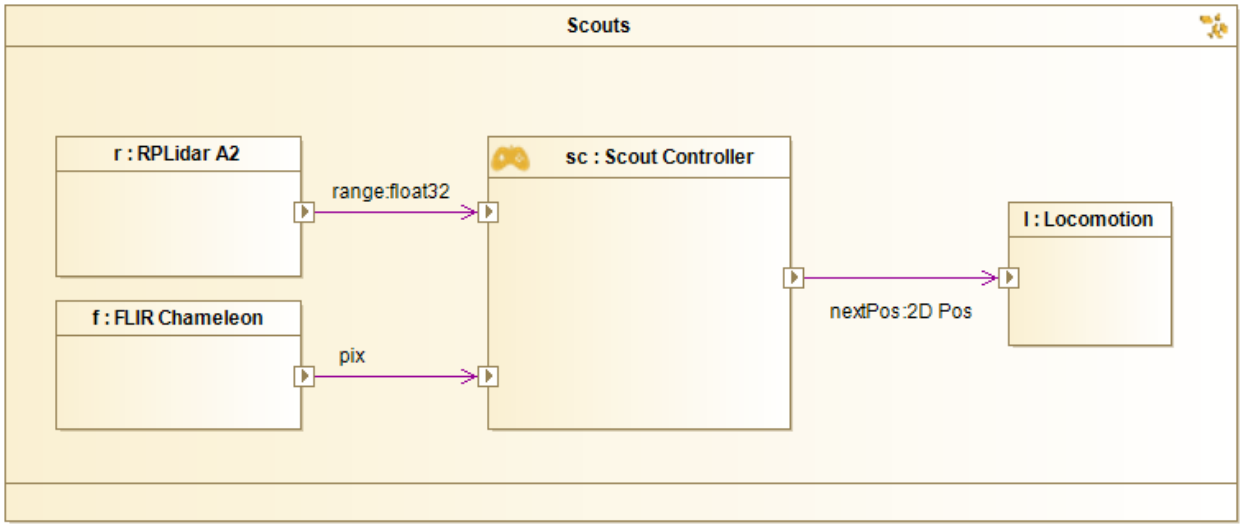

Figure 14: Scout hardware description

4.4.2 Behaviour Modelling

Scouts, in Logistics scenario, are aiming to localize cart that workers will move. To do so the behaviour, shown in Figure 15Figure 12 ,has been made. Firstly, the scout is started and passes to the "Idle" mode waiting for an operator start the mission. Once a cart is found by a scout, it assigns a worker to it before coming back to the "Scouting" state waiting another cart to be found.

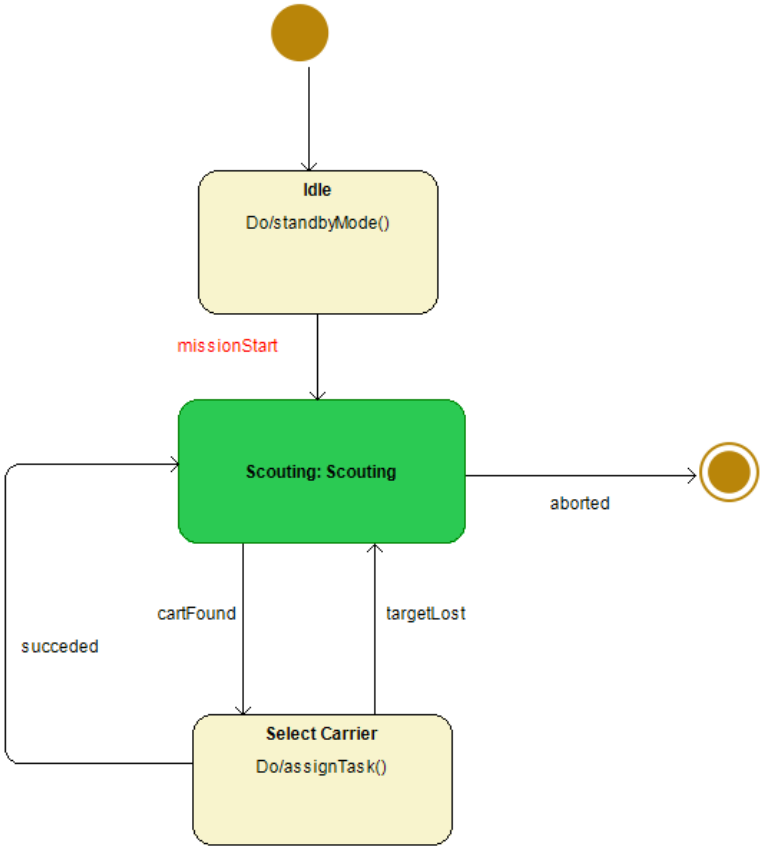

Figure 15: Scout behaviour

5 Design Patterns for Human2Swarm Interaction

Additionally, to the CPS models and behavior libraries, we provide a library with design patterns that describe rules and guidelines for the usage of CPS swarms. The concept of design patterns, how to build them and to structure them, were already described in Deliverable D4.2 – Updated CPS Modelling Library.

The design pattern library contains safety regulations related to the use case in logistics. Thus, a set of design patterns explains what we need to take care of when deploying a swarm of UGVs in a logistics environment. The library can be found on the web via https://patterns.fit.fraunhofer.de/cpswarm/index.php/browse-patterns and will be part of the final CPSwarm workbench source code documentation on git.

The library has following layout:

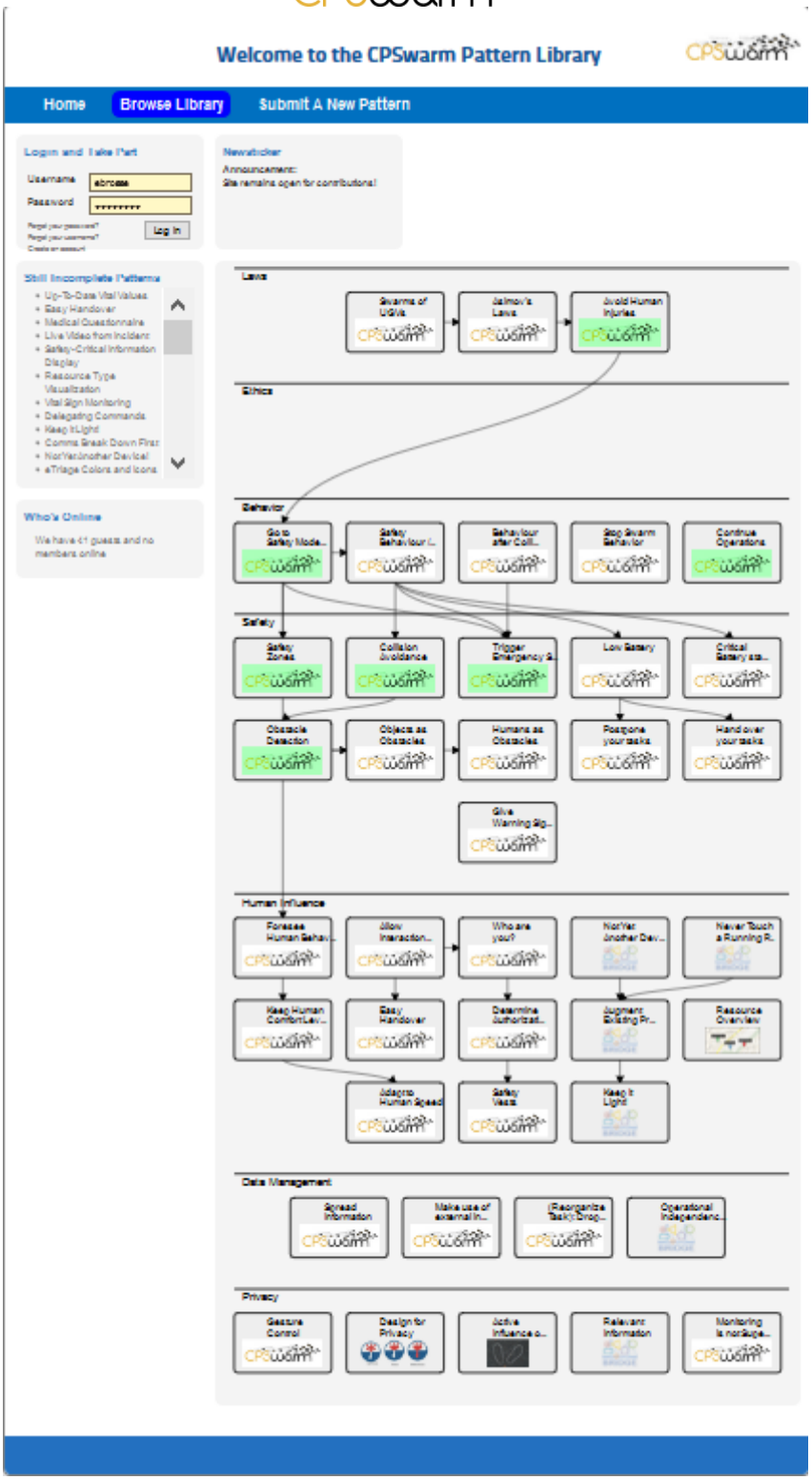

Figure 16: CPSwarm Pattern Library Layout

5.1 Organization of the Design Patterns

The design patterns are organized in the following categories, whereby the green-marked patterns indicate their implementation throughout the project and the ones different with another logo (not CPSwarm) were reused from other projects. The categories are not in a strict hierarchical order but introduce a view from very global aspects, like laws "down to" data management. As baseline, privacy considerations are discussed. The

structure at the moment suits best to follow general rules and guidelines leading to safety patterns and patterns dealing with human behavior. In later iterations, the order may change for the sake of readability.

5.1.1 Laws

Patterns in the laws category, as shown in Figure 17, deal with general rules on how autonomous vehicles must behave in any way. From the "Avoid Human Injuries" patterns, the behaviour and configuration of the warehouse swarm system is derived. The pattern itself is derived from a consideration of "Asimov's Laws".

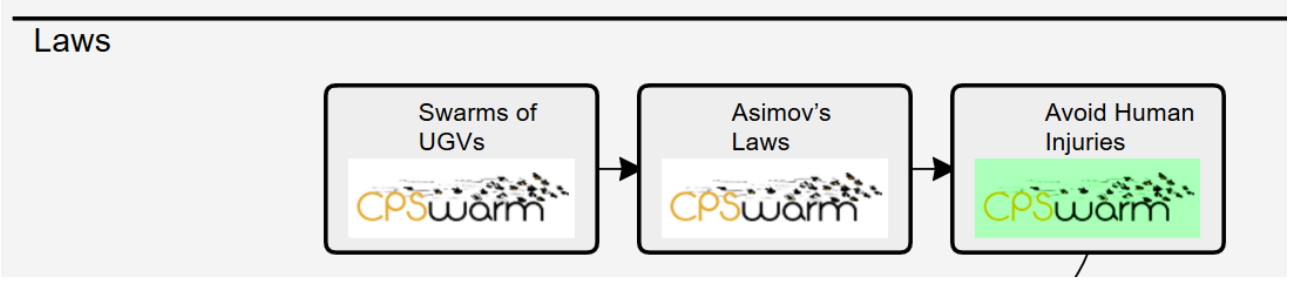

Figure 17: CPSwarm Laws Patterns

5.1.2 Behaviour

These patterns, depicted in Figure 18, describe general behaviours and qualities of the swam, like when to "Continue Operations", going to "Safety Mode" in case of an uncertain or critical situation. Other behaviours after accidents or collisions or emergency stops are also discussed for further aspects of swarm scenarios.

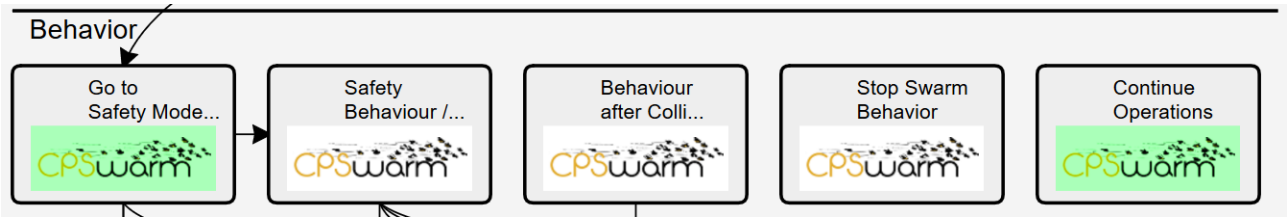

Figure 18: CPSwarm Behaviour Patterns

5.1.3 Safety

The Safety patterns, depicted in Figure 19, deal with questions on how to preserve human safety by keeping distances, detecting obstacles or going to emergency mode behaviours and changing task organizations due to safety restrictions.

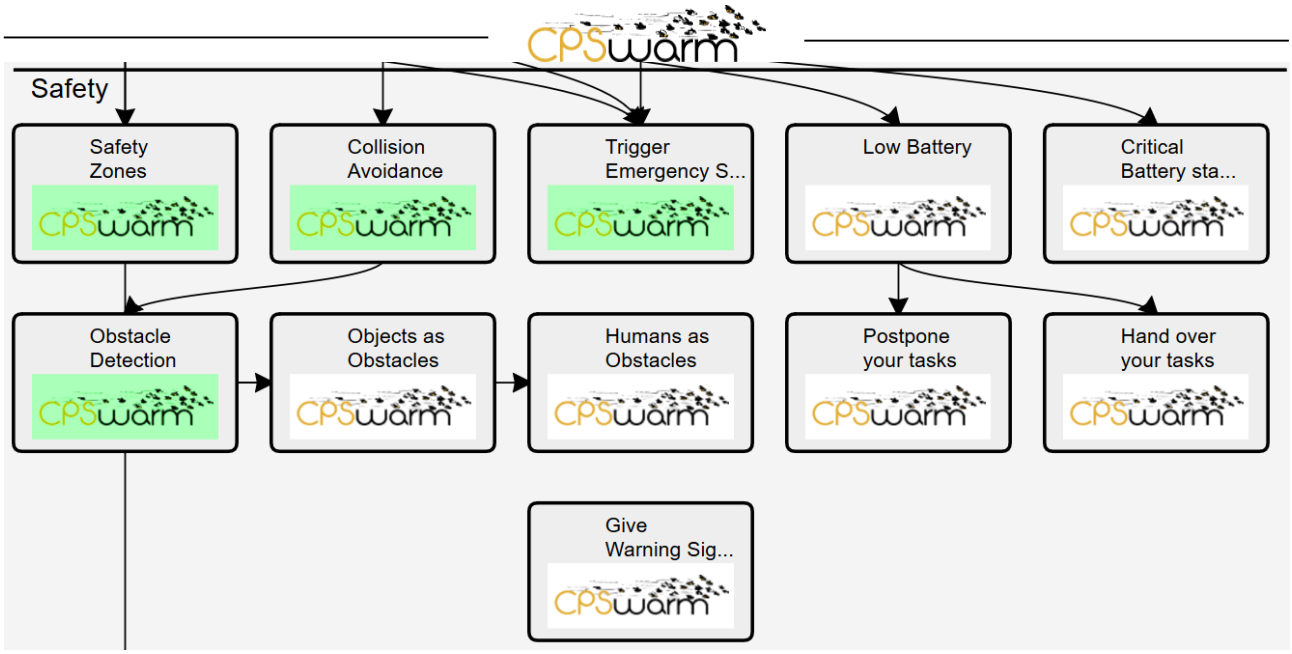

Figure 19: CPSwarm Safety Patterns

5.1.4 Human Influence

This category, shown in Figure 20, deals with aspects on how humans that are present beside swarm should be integrated in the planning and execution of tasks as well as directly influencing the swarm's behaviour. For instance, changes in current operations may become necessary or the human needs measures to interact with the swarm.

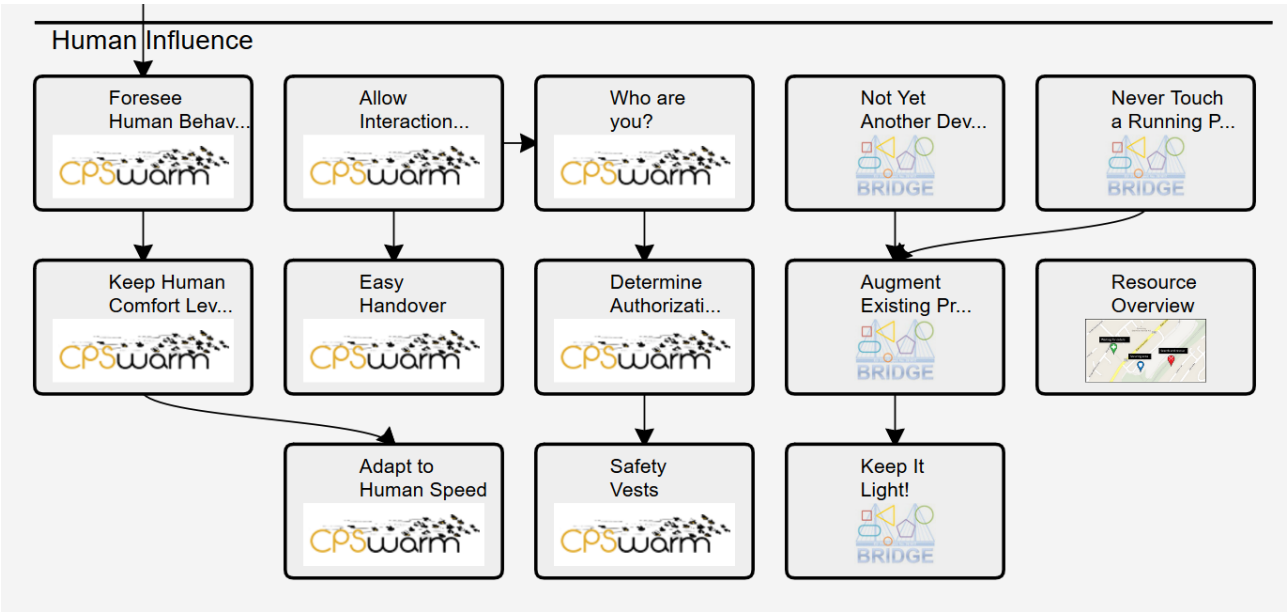

Figure 20: CPSwarm Human Influence Patterns

5.1.5 Data Management

This category, as depicted in Figure 21, includes a first set of more technical patterns on how to deal with questions related to data acquisition, exchanging and using data for operations planning. Here, future implementations leading to best practices should help collecting and extending the patterns library at this point.

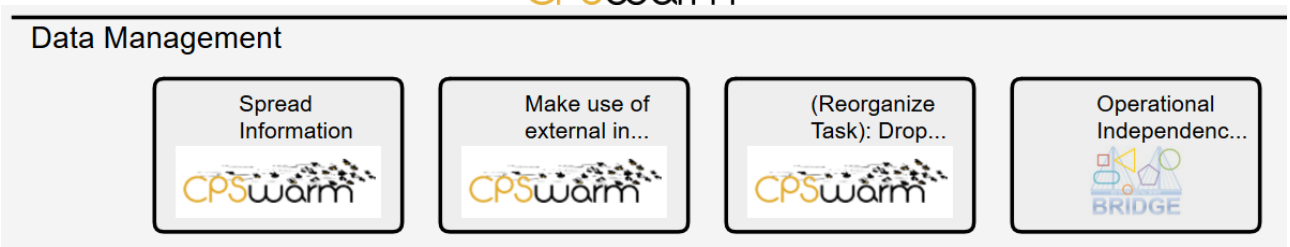

Figure 21: CPSwarm Data Management Patterns

5.1.6 Privacy

Patterns related to privacy related questions, as shown in Figure 22, for implementation that occur in any kind of information system where (sensor) data is collected and analyzed. Here, designs and decisions can be made early in the engineering process.

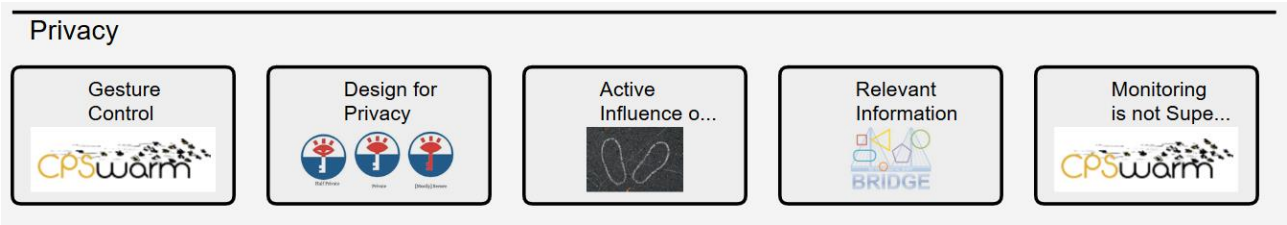

Figure 22: CPSwarm Privacy Patterns

5.2 Implemented Design Patterns

As already mentioned, the green-colored patterns where implemented as part of the logistic swarm use case. In the following tables you can find the individual description of the design patterns, that can be found online in the following form:

Avoid Human Injuries

Pattern Context / Usage

It is mandatory to detect humans in order to avoid injuries (cf. safety regulations at ISO 1525-1997: https://webstore.ansi.org/Standards/DIN/DINEN15251997).

Problem Summary

Collisions between UGVs and humans are highly dangerous and may cause severe injuries or even causalities. Collisions with humans MUST be avoided at any cost.

Problem Details and Forces

Identify humans

Solution Summary

To avoid injuries several possibilities exist: • Work in a safety mode (reduced velocity) • Emergency stop (if some danger is detected) • Keep a safety distance within a safety zone

Solution Illustration

Solution Details and Consequences

Camera and distance sensors will be used to identify humans.

Related Patterns

- Go to Safety Mode / Safe Behavior

Figure 23: Avoid Human Injuries Pattern

Name	Collision Avoidance
Pattern Context	UGVs are moving autonomously in areas where obstacles may occur dynamically and in an unplanned / unforeseen way.
Problem Summary	In case an obstacle is hit, the UGV or obstacle may be damaged seriously. Eventually, the mission must be aborted.
Problem Details	The mission may be interrupted or even aborted. Obstacles or the UGV could be damaged in such a way that the mission is delayed or cannot be completed. Damaged UGVs may be needed to taken out of the system / swarm. Damaged obstacles may need to be discarded and replaced. These consequences may lead to delays, injuries and increased costs.
Solution Summary	Find mechanisms to detect obstacles early enough such that the UGV can react accordingly.
Solution Illustration	
Solution Details	Reaction could be: slowing down, stopping, evade or even retreat. It recommended to report such that an operator (system or human) can react and find a solution to remove the obstacles. Alternatively, the strategy of the UGVs may be adapted by, e.g., updating the map and find new routes.
Related Pattern	Obstacle Detection, Avoid Collisions with Objects, Avoid Collisions with Humans
References	

Table 1: Collision Avoidance Description

Name	Safety Zones
Pattern Context	In case an obstacle or human gets in the way of the route of the UGV, the behavior needs to be adapted such that the UGV can avoid collision or check whether the environment changes again, e.g., because a human step out of the way or the obstacle is removed.
Problem Summary	Depending on the distance to obstacles different strategies to react need to be applied reaching from slowing down to full stop.
Problem Details	
Solution Summary	Define safety zones depending on the distance and adjust the UGVs behavior accordingly
Solution Illustration	
Solution Details	Defined safety zones from the center of the robot (cf. ISO 1525-1997): .5m: danger area (reduced speed) .25m safety area (emergency stop)
Related Pattern	
References	

Table 2: Safety Zones Description

Name	Emergency Stop
Pattern Context	While a UGV is in operation, unforeseen events may require an emergency stop of operation for any reason.
Problem Summary	In an unforeseen situation, the operator needs a fast and direct way to stop all operations immediately.
Problem Details	
Solution Summary	Implement an emergency stop switch at the operator's management system and on the UGVs itself.
Solution Illustration	It needs to be ensured that the emergency stop operation can be triggered at any time, remotely or at the UGVs directly.
Solution Details	According to the regulations (cf. ISO 1525-1997) at every UGV, there must be a reachable red button, the same holds for dedicated spots in the area. Push buttons in red are commonly used and labeled appropriately.
Related Pattern	
References	

Table 3: Emergency Stop Description

Name	Continue Operations
Pattern Context	After an emergency situation is resolved, operations need to go back to normal. Ideally, the last state is resumed.
Problem Summary	A trigger is needed to reuse the last stable state and continue from there.
Problem Details	
Solution Summary	Implement a remote trigger for operators to resume operations for all stopped UGVs. A trigger on a single UGV can be appropriate, too.
Solution Illustration	
Solution Details	
Related Pattern	
References	

Table 4: Continue Operation Description

Name	Go to Safety Mode / Safe Behavior
Pattern Context	In case of uncertainty about the safety of current operations, e.g., caused by unauthorized personnel or the operation during maintenance, the UGVs need to continue operations if safety can be guaranteed.
Problem Summary	The UGV does not necessarily have to stop operations but must be prepared to perform an emergency stop for all movements.
Problem Details	
Solution Summary	Perform all operations in a slow way such that emergency stops are possible at all times and moving parts cannot harm any human around. Force needs to be decreased in such a way that contact to an obstacle causes immediate stop. This includes 1) Evade / find new route 2) Keep distance
Solution Illustration	
Solution Details	
Related Pattern	
References	[1] safelog-project.eu [2] Lasota, Song, Shah, A Survey of Methods for Safe Human-Robot Interaction

Table 5: Safe Behaviour Description

<table>
<tr><th>Name</th><th>Obstacle Detection</th></tr>
<tr><td>Pattern Context</td><td>UGVs need mechanisms for “Collision Avoidance”. Depending on the environment, different approaches have special advantages or drawbacks that need to be considered when implementing a solution.</td></tr>
<tr><td>Problem Summary</td><td>Depending on the environment and kind of obstacles, different technological solutions need to be considered.</td></tr>
<tr><td>Problem Details</td><td>Environmental parameters o be considered are the following:
- Light conditions
- Line of sight
- Moving speed
- Translucence of the obstacle
- Distance to the obstacle
- Stationary vs. moving obstacle
- Labelling by 2D tags
- Labelling by RFID
- Labelling by NFC</td></tr>
<tr><td>Solution Summary</td><td>For each parameter, dedicated sensor systems are most appropriate. Eventually, combinations of approaches are necessary to meet different environmental parameters.</td></tr>
<tr><td>Solution Illustration</td><td></td></tr>
<tr><td>Solution Details</td><td>
<table>
<tr><th>Parameter</th><th>Technical Solutions</th><th>Limitations</th></tr>
<tr><td>Light Conditions</td><td>- IR camera vs photo electric</td><td>- Too light / too dark
- Too transparent</td></tr>
<tr><td>Distance</td><td>- ultrasonic, sonar, radar, photo-electric, photo cell</td><td>- range < 80cm
- object colour, environmental light,
- object material
- object form</td></tr>
<tr><td>Stationary</td><td>All appropriate (see above)</td><td></td></tr>
<tr><td>Moving</td><td>- high measuring frequency sensors (all)</td><td>- Frequency of movement < frequency of measuring</td></tr>
</table>
</td></tr>
<tr><td>Related Pattern</td><td>Object as Obstacles, Humans as Obstacles</td></tr>
<tr><td>References</td><td></td></tr>
</table>

Table 6: Obstacle Detection Description

6 Conclusions

This deliverable presents the status of the available CPS models at M35 of CPSwarm project. Even if the presented models (CPSwarm library, CPS hardware and behavior aspects, communication, and human in the loop) can, of course, still be improved but they have been already used inside both research and industrial case studies at different level.

Annex A

```
#
# Configure endpoint
#
endpoint = {
    name = "drone"        # Possibly non-unique name for the local node
    deviceClass = "drone"   # Discoverable device class
    type = "zyre"           # Endpoint type
    parameters = {          # Endpoint parameters, which for Zyre endpoints can be:
        # ifname = "eth0"       # Network interface to bind to
        # port = 34000          # Port to use for UDP beacons
    }
}

#
# Configure bridged services
#
services = {
    #
    # Outgoing events
    #
    # The bridge will subscribe to these topics and forward
    # received messages as events to the swarm.
    #
    #  - message: fully qualified name of the underlying
    #             ROS message type (must have a field named
    #             header with message type swarmros/EventHeader)
    #  - source:  ROS topic to forward events from
    #
    outgoingEvents = (
        {
            message = "cpswarm_msgs/LocalTargetPositionEvent",
            source = "target_found"
        }
    )

    #
    # Incoming events
    #
    # The bridge will listen to these events and republish
    # them under ROS topics.
    #
    # NOTE: Only one handler per event name can be added. If
    #       desired, handlers with the same message type can
    #       republish to the same topic.
    #
    #  - suffix:  will be published under events/<suffix>
    #  - message: fully qualified name of the underlying
    #             ROS message type (must have a field named
    #             header with message type swarmros/EventHeader)
    #  - name:    discoverable event name
    #
    incomingEvents = (
    {
            suffix = "launch",
            message = "swarmros/SimpleEvent",
            name = "launch"
        }
    )

    #
```

```
    # Published parameters
    #
    # The bridge will load the value of these parameters from
    # the ROS Parameter Server, then publish them both as
    # ROS topics and as remotely available key-value targets.
    #
    # NOTE: Both suffixes and parameter names must be unique. If
    #       desired, two parameter publishers can reference the
    #       same ROS parameter path. Parameters must have a valid
    #       value before the bridge is started.
    #
    #  - suffix:  will be published under parameters/<suffix>
    #  - message: fully qualified name of the underlying
    #             ROS message type (can by any type)
    #  - name:    discoverable key-value path
    #  - path:    ROS parameter path
    #  - rw:      whether set requests are accepted
    #
    publishedParameters = (
    #    {
    #        suffix = "reportInterval",
    #        message = "swarmros/UInt",
    #        name = "reportInterval",
    #        path = "example/reportInterval",
    #        rw = true
    #    }
    )
}
```

References

Acronyms

Acronym	Explanation
ANN	Artificial neural networks
API	Application Programming Interface
CPS	Cyber Physical System
FREVO	FRamework for EVOlutionarydesign
GPS	Global Positioning System
GUI	Graphical User Interface
LTE	Long Term Evolution
ROS	Robot Operating System
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
UWB	Ultra Wide Band
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
ZMQ	ZeroMQ