D4.2 – UPDATED CPS MODELLING LIBRARY

Deliverable ID	**D4.2**
Deliverable Title	**Updated CPS modelling library**
Work Package	**WP4 – Models and algorithms for CPS Swarms**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**28-09-2018**
Status	**Final**
Lead Editor	Etienne Brosse **(SOFTEAM)**
Main Contributors	Melanie Schranz, Micha Rappaport **(LAKE)**, René Reiners **(FRAUNHOFER)**, Alessandra Bagnato **(SOFTEAM)**

Published by the CSPwarm 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	2018-07-03	Etienne Brosse (SOFTEAM)	First Table of Content.
0.2	2018-09-04	Melanie Schranz (LAKE)	Content for chapters 5 and 6.
0.3	2018-09-06	Melanie Schranz, Micha Rappaport (LAKE)	Adapt content for chapter 5 – communication modeling.
0.4	2018-09-10	Etienne Brosse, Alessandra Bagnato (SOFTEAM)	Content for chapters 3 and 4.
0.5	2018-09-13	Melanie Schranz (LAKE), René Reiners (FRAUNHOFER)	Adapt content for chapter 6 – design pattern library
0.6	2018-09-17	Etienne Brosse (SOFTEAM)	Clean up and finalization.
0.6.1	2018-09-21	Davide Conzon (ISMB)	Chapter 4 updates
0.7	2018-09-21	Etienne Brosse (SOFTEAM)	Consolidated version
0.7.1	2018-09-25	Melanie Schranz (LAKE)	Update according to internal review comments
0.7.2	2018-09-28	Etienne Brosse (SOFTEAM)	Update according to internal review comments
1.0	2018-09-28	Etienne Brosse (SOFTEAM)	Final versione, ready to be submitted to EC

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-09-21	Junhong Liang (FRAUNHOFER)	Approved with few comments
2018-09-21	Davide Conzon (ISMB)	Approved with few comments

1 Executive summary

This deliverable, namely "D4.2 - Updated 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 will be 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 till M20 of the CPSwarm project.

Contents

2 Introduction

D4.2 – "Updated CPS modelling library" is a public document describing the publicly available CPS models designed till M20 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 a previous study – published in D4.1- 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 three 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. In Section 5, the communication modelling between CPSs are described. The Human aspect modelling is depicted in Section 6. Finally, Section 7 draws conclusions and provides a first feedback.

2.3 Related documents

ID	Title	Reference	Version	Date
[D5.3]	Updated CPSwarm Modelling Tool	D5.3	1.0	2018-06-30
[D3.2]	Updated System Architecture and Design Specification	D3.2	1.0	2018-06-28
[D4.1]	Intitial CPS Modeling Library	D4.1	1.0	2017-10-06

3 CPSwarm Modelling library

As described in deliverable D3.2, the CPSwarm Modelling library is composed of a set of predefined elements, which can be reused in CPSwarm design context. In this deliverable the authors will focus on the elements reusable in CPS design and not – for example – for environment or fitness function specification. Three categories of elements are available:

- Hardware Component;
- Behavior;
- Software Library routine which abstracts Hardware functionalities, allowing its management and control.

Each of these categories is described in the following sections.

3.1 Hardware Component

The hardware component category regroups all the element available for the CPS hardware description. As depicted in Figure 1, this category has been decomposed in three sub categories respectively named "Actuator", "Controller" and "Sensor". In each sub categories a set of components representing the physical devices are listed which can be used to build a CPS.

Figure 1 Hardware component available in the CPSwarm modelling library

In the Sensor category, several components are modelled. For example the "CNY70" component is an optical sensor able to provide light intensity value. This sensor is part of the "Spiderino" robot described in Section 4.1. More precise description of the Spiderino hardware is shown in Figure 6.

3.2 Behaviour

The behaviour category regroups all "swarm" algorithms available for the CPS behavioural description cf Figure 2.

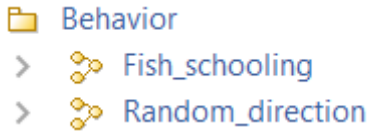

Figure 2 Behaviour available in the CPSwarm modelling library

Each swarm algorithm or behaviour has been described as a UML State Machine and added to the CPSwarm modelling library. Figure 3 depicts a simple Swarm algorithm named "Random_direction". The Swarm Member/CPS with this behaviour will move to a random position which is not out of the specified bounds. This algorithm starts by computing a future position for the swarm member aka CPS. Then if the CPS is **not out of the environment bounds**, it moves to the calculated position. If the computed position is **out of the environment bounds** a new direction is computed.

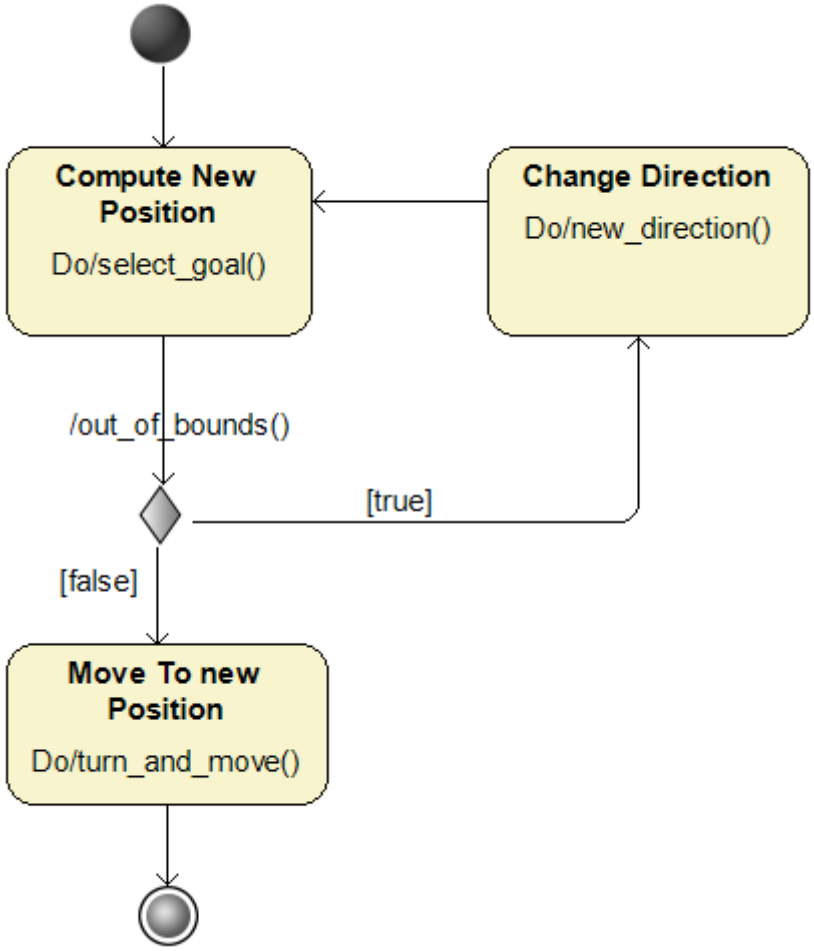

Figure 3 Random direction behaviour description

Note that the three states – respectively named "Compute New Position", "Change Direction", and "Move To New Position" - defined in this simple Swarm algorithm refer to services - respectively named "select_goal", "new_direction", and "turn_and_move"- defined in the Abstract Library, which will be described in the next section.

3.3 Abstraction Library

The third category of elements presented in this deliverable consists of a model representation of the Abstraction Library as defined in D5.3. According to this deliverable, "*the Code Generator relies on the Abstraction Library that provides a set of platform-independent functionalities*". These functionalities or services has to be abstracted as model usable for the Behaviour modelling (as presented in Figure 3). Figure 4 shows part of the Abstract Library in which different packages have been defined to sort the functionalities. Under each folder a set of Node are available. For example, CPS folder presents two nodes respectively named "Drone" and "Rover". Finally, functionalities are defined under the Nodes.

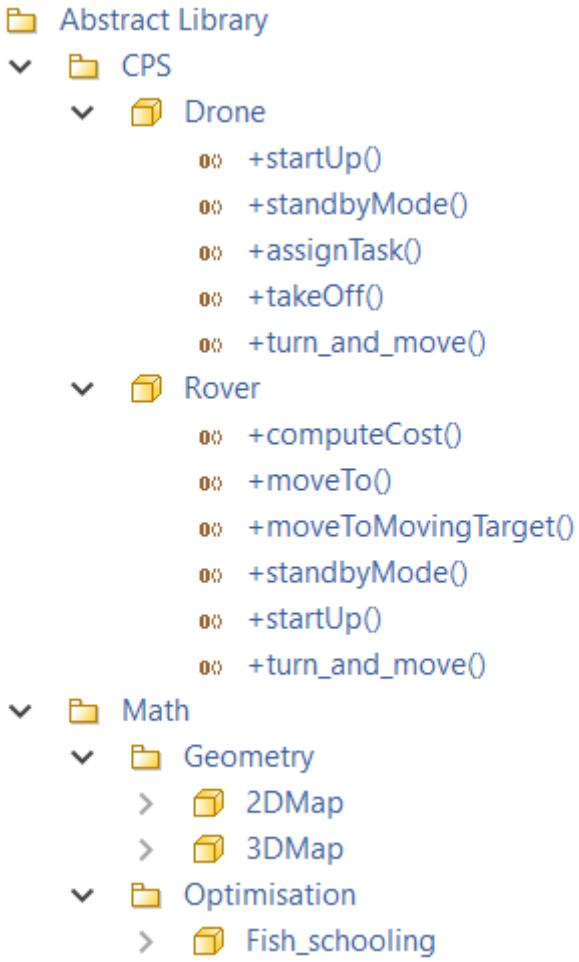

Figure 4 Modelling representation of the Abstract Library

4 Available CPSs Model

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-modelio37/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. A Communication Interface component is also part of this design. This component is responsible of the communication – see Section 5 - between the Spiderino swarm. Finally, the BEECLUST component holds the Swarm Algorithm/ CPS behaviour which consists in a UML State Machine implementation of one Bee Algorithm.

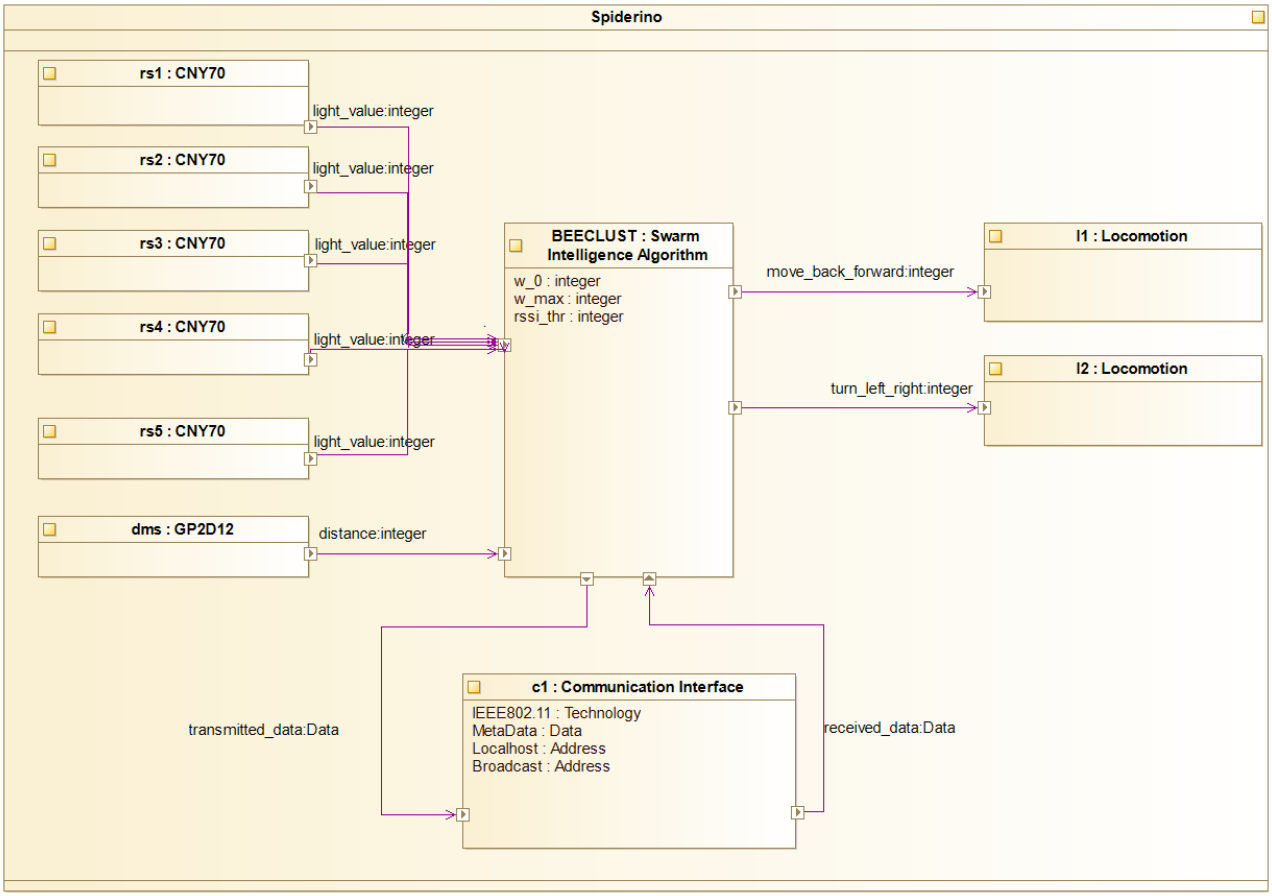

Figure 6 Spiderino hardware architecture

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.

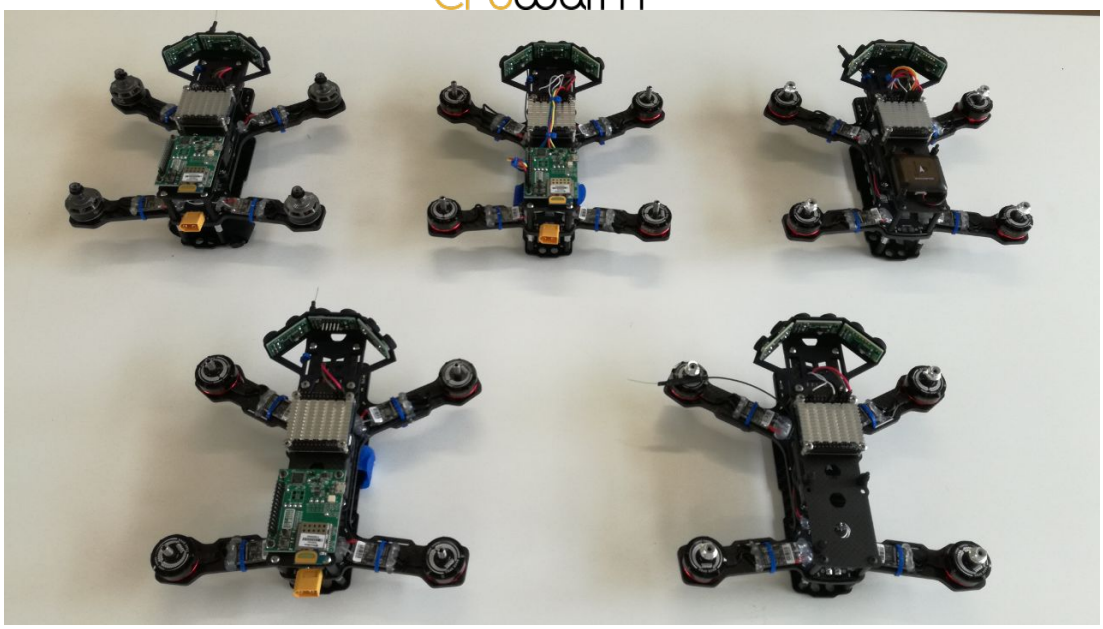

Figure 7 Drone platforms

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

- The SAR_Behavior component holding CPS behaviour.
- Three sonars named ps1, ps2 and ps3 providing a value representing the light intensity,
- One camera measuring the distance,
- One UWB giving the local position of the device,
- One communication interface exchanging with the outside world,
- Two locomotion component respectivel in charge of moving -forward or backward - and turning - left or right.

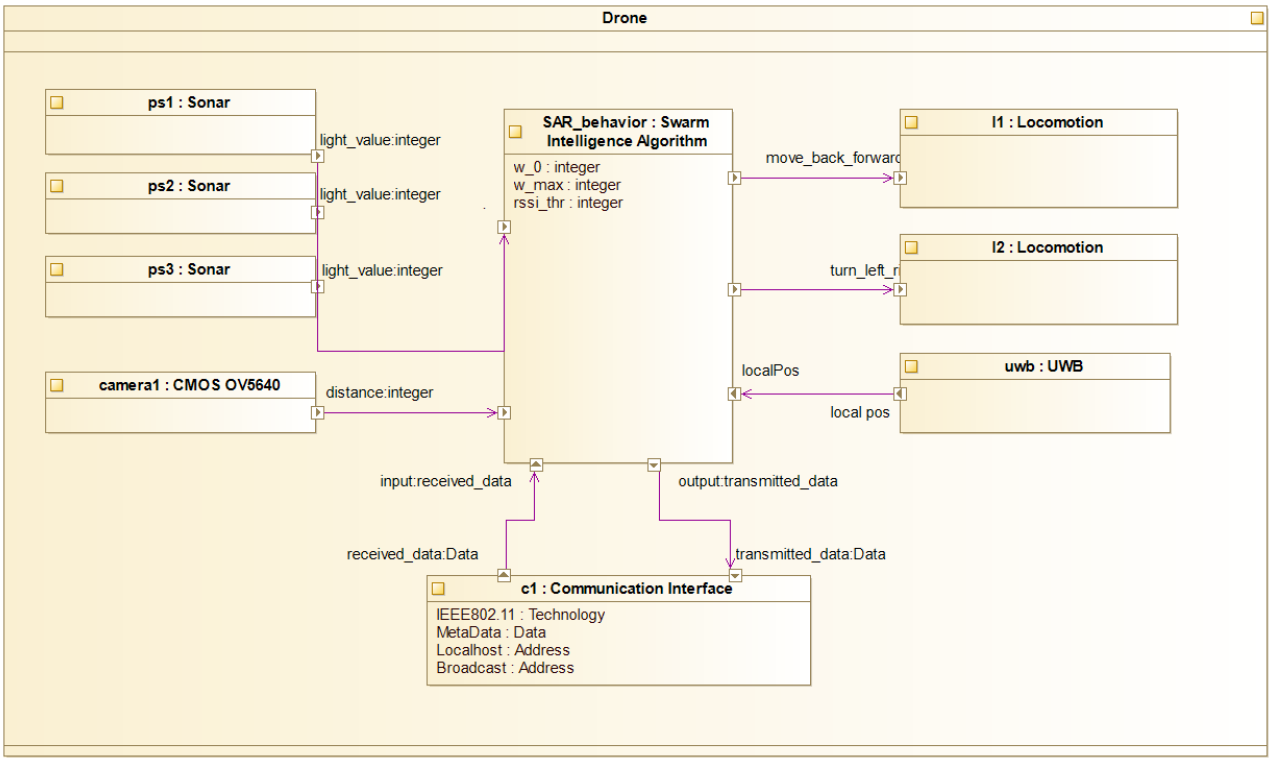

Figure 8 Drone hardware description

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 "Start up" 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.

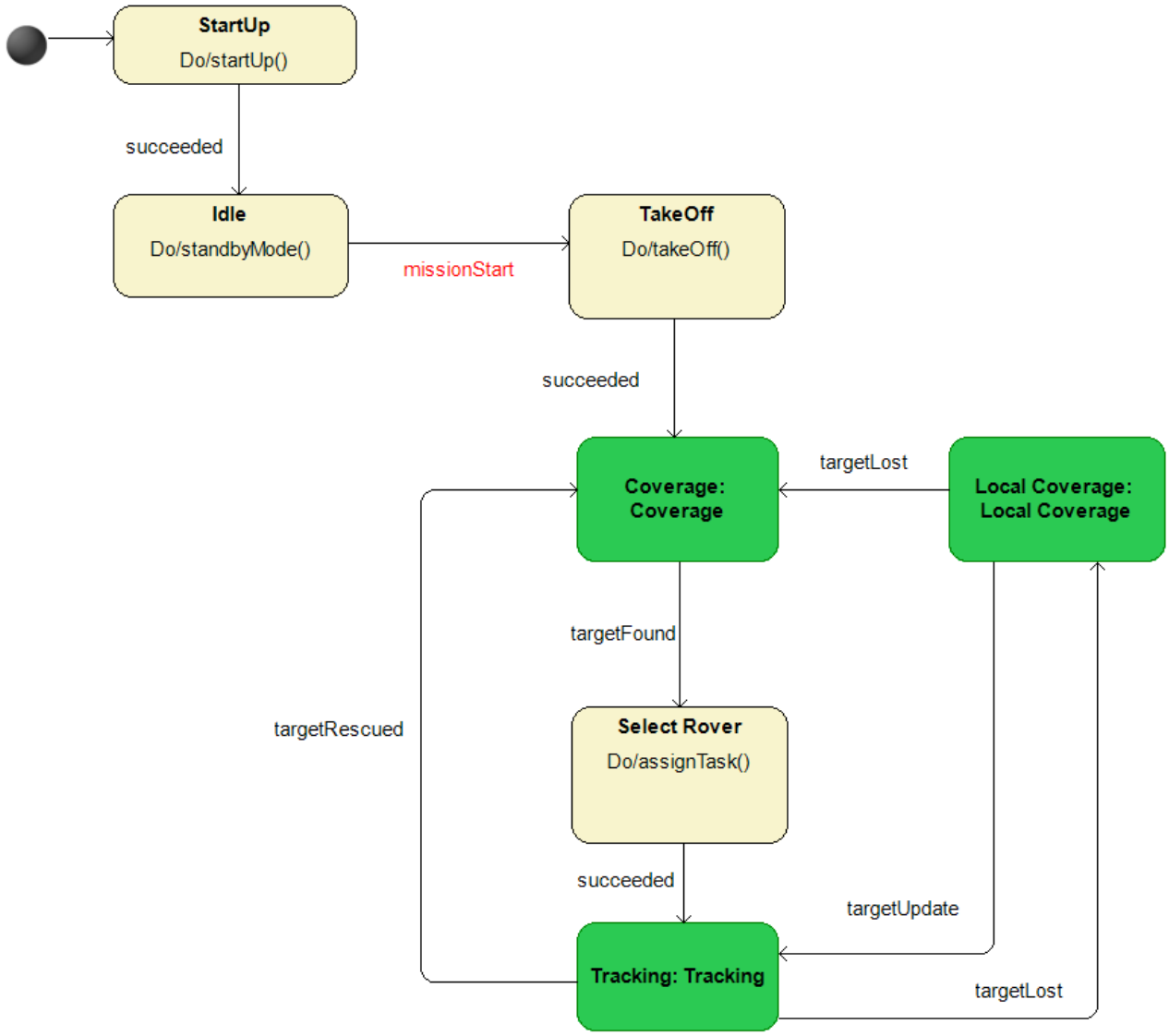

Figure 9 Drone behaviour

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

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 11 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.

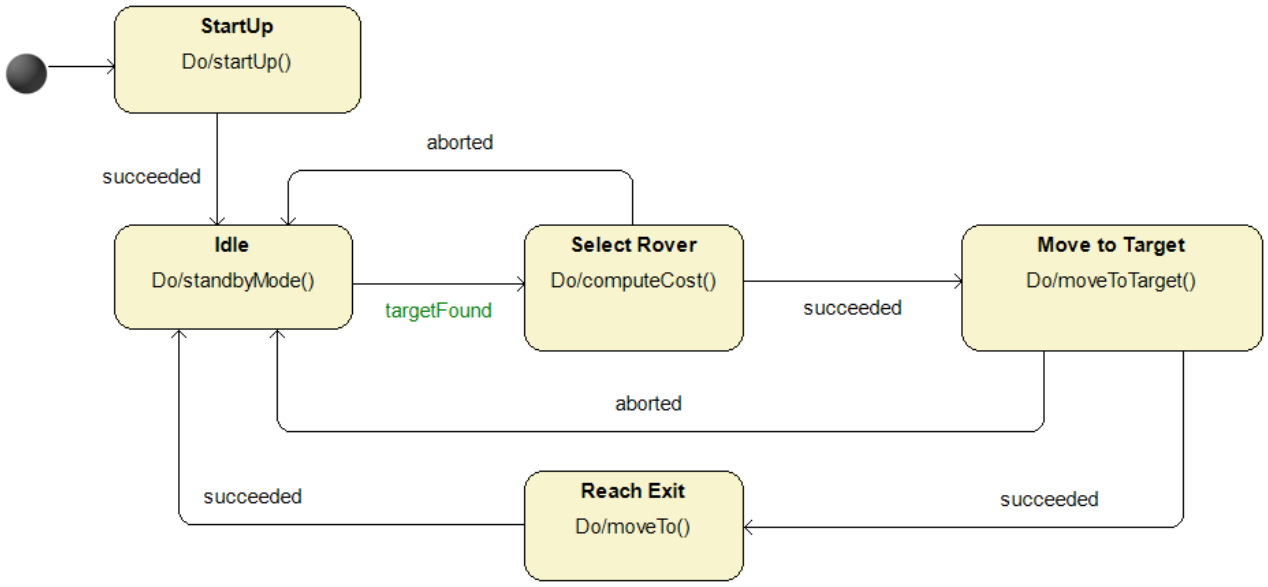

Figure 11 Rover behaviour

5 Communication among CPSs

A swarm of CPSs works as a consolidated swarm if communication between swarm members is available. Different kind of data needs to be exchanged among the swarm members, including current position, current speed, sensor data, etc. This data is also necessary to execute swarm algorithms, which interact dependent on the outputs of the involved swarm members.
Thus, communication is a central point in CPSs swarm design and therefore, needs to be embedded at the very early design stage – namely as a model in the modelling tool.

As described in D4.1 - Initial CPS Modeling Library, a class called Communication Interface models relevant variables to be exchanged as public attributes in a subclass. The „use" relationship indicates which members are going to use these variables (see as an example for drone-rover communication in Figure 12).

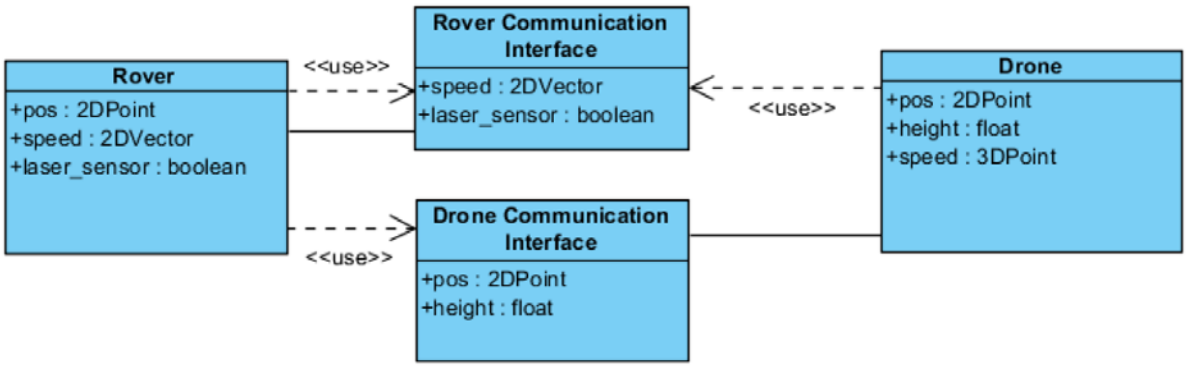

Figure 12 Example to model communication between rovers and drones in a swarm (see D4.1)

5.1 Communication Protocols

Before any high-level discussion of protocols can take place, it is important to agree on the basic infrastructure on which any solution later developed will have to operate. Back in November, the CPSwarm Consortium have agreed to use IP based networking – but the details and consequences of this decision haven't been explored further at that point. During the several meetings, the partners examined the networking stack below IP, and identified four concrete low-level stacks that CPSwarm aims to support:

- Standard, infrastructure mode wireless network (based on IEEE 802.11 a/b/g/n/ac)
  This is a realistic option for scenarios where swarm members are close together in a well-defined area, and an excellent testing ground for CPSwarm solution. This will be the starting point, since it should be easy to integrate due to its ease of use.
- Cellular network (based on 3G/LTE)
  When swarm members are spread out over a larger area, this might be the only option. Since it requires cellular service, deployment options are limited to areas with coverage. If the high-level protocol requires members to be on the same subnet, a GRE1 tunnel (Generic Routing Encapsulation) can be used to connect them to a virtual router. As this configuration is related to network topology, higher level protocols need not be aware of this implementation, and can operate in the same way they would if an infrastructure mode wireless network was in place. Tests might be conducted at later stages of the project using cellular networks, but no further effort should be placed on getting this to work, since no use case partner has indicated that this is their primary method of communication.
- Time triggered wireless network (property technology defined by TTTech).

The solution is similar to a standard wireless network, but it has a time-triggered implementation on the MAC layer. The network topology is fixed, however, to support its unique features, a way to pass information on the priority and criticality of the packet from the application layer to the network layer is needed. This assumes that the operating system on which the applications are running support this protocol natively, or that the protocol does not require such support. Till the end of the project further investigation about the standard will be done with the support of TTTech. *Further updates will be provided once a working prototype on the considered standard wireless network will be available, ready to be extended to support time-triggered networks.*

- Low-rate wireless personal area mesh network (based on IEEE 802.15.4, (Gutierrez, Callaway, & Barrett, 2003)

  The ultimate goal of the analysis of these protocols is to arrive at a solution that can work on mesh networks – for this, hardware support will be necessary. While 802.11s2 based mesh networks can work with common wireless network adapters, their range leaves a lot to be desired, and one of the requirements identified was to use longer range mesh networks in place of cellular networks, in areas where coverage is poor or nonexistent – a likely scenario for the SAR use case. This requirement is satisfied only by a subset of 802.15.4 based systems that can support IP networking: Zigbee IP and Google Thread – other 802.15.4 based protocols might support mesh networking, but lack support for IP. The XBee and XBee Pro hardware components have been investigated, but they do not support Zigbee IP, and XBee only claims that Thread will be supported in the future There are other solutions, however: the OpenThread library runs on a number of platforms including on Linux using raw 802.15.4 sockets For workbench components to work with this solution, dongles will be required since commercial PC hardware has no support for the physical layer used by these protocols. Mesh networking also has much lower bandwidth and works best with smaller packet sizes – something to keep in mind when the protocol will be defined. As such, if hardware platforms can be fitted with Linux compatible 802.15.4 radios, this option can be further investigated, but only at a later stage of the project and when the basics of the protocol are in place.

The essence of the features that are required include

- Discovery and enumeration – getting a list of swarm members
- Property setters and getters – to set parameters and goals
- Property subscriptions and notifications – to stream telemetry and monitor members
- Events (and commands) – so individual swarm members can react by changing behavior
- Large messages – to send software updates
- Prefer UDP over TCP
- Must support authentication, integrity protection and encryption
- The less bandwidth it uses, the better and the smaller the messages, the better it will work over mesh networks – to send commands related to operation.
- It should be tolerant to swarm members falling out and joining
- Must have a way to notify the central control in case of communication problems

After a research on the protocols discussed, and based on the information presented so far, Three solution have been considered suitable for the CPSwarm use cases.

- *The piece-of-cake protocol*
  This is a quick and dirty solution for all CPSwarm problems, but like all quick and dirty solutions, there is a price to pay. For this, XMPP will be used – all swarm members connect to an XMPP server through a secure TLS connection, and use XMPP primitives to realize the required features – In-Band Bytestreams9 for deploying software updates and Publish-Subscribe10 for monitoring and events. Enumerating swarm members and tracking their presence is a piece of cake, while there are plenty of authentication options available. Server software and client libraries need to be evaluated to find the ones with best support for the subset of features that will be used, since some of these are still

undergoing standardization. While this protocol can work with any underlying IP based network infrastructure, its heavy XML messages and the use of TCP based TLS sockets come at a stiff price in terms of bandwidth and latency

- *The do-it-yourself protocol*
  ZMQ acts as a glorified network abstraction layer – of all the available network protocols, epgm (which is basically PGM over UDP) is the best fit to handle swarm-wide event distribution (using a publish-subscribe pattern, while one-to-one communications required by the monitoring and configuration tool as well as the deployment tool can be handled by simple ZMTP sockets over TCP. In the first case, if integrity protection is sufficient, a simple signature scheme can be implemented, while ZMTP sockets can be secured using CURVE- with the tools acting as servers and the swarm members as clients for the purposes of the encryption, but not for purposes of initiating the connection. It is most likely desirable to combine ZMQ sockets with some kind of binary serialization solution – Apache Thrift or Google Protobuf, for example. It is likely that a custom discovery solution will also need to be rolled. *Overall, this will lead the protocol to be very lightweight, suited for mesh networks, but will require a little more legwork during development, since the library only provides relatively low-level primitives.*
- *The hybrid protocol*
  This is a combination of the two ideas – using the XMPP approach for monitoring, configuration and deployment, and using ZMQ for realizing the communications between swarm members. If a hybrid system is used with a mesh infrastructure, an interesting new option opens up: if an out-of-band connection like LTE is used for connecting to the central server, and the mesh network is used to provide connectivity between swarm members, all nodes with working cellular connections can act as edge routers for the mesh network to provide a bridge for other nodes located in areas with poor coverage. *The performance characteristics of a hybrid protocol are somewhere in-between – if the out-of-band connection scheme is used, the better the cellular coverage, the more it will converge to the pure decentralized approach.*

The current implementation of the communication library is a C++ library called swarmio. It is based on the ZMQ protocol using the Zyre library[1] which is an open-source framework for proximity-based peer-to-peer applications. swarmio itself provides a C++ API for swarm members and swarm management tools to implement a number of services: discovery, key-value store, event handling, pinging. More services are to come (telemetry, deployment, etc.). Low level communications are abstracted by Endpoints. While Zyre supports encryption, it is not enabled at the moment - a new type of endpoint most likely not based on Zyre will be coming later to support authentication, authorization and encryption. Additional endpoint types will also need to be developed to support the fixed network topology that will be used for the time-triggered wireless Ethernet.

5.2 Communication Modelling

The swarmio library is integrated into the CPSwarm workbench through a ROS package called swarmros. swarmros bridges the ROS internal communication over a network interface to other CPSs in the swarm. This is achieved by implementing a bridge that subscribes to ROS internal topics and reads ROS internal parameters. This information is send out over the network interface and received by other swarmio/swarmros instances. The receiving bridge then publishes this information again to ROS. swarmros allows to configure the communication endpoint as well as several services. The communication endpoint is modeled by

- Name: Possibly non-unique name for the local node.
- Type: Endpoint type, e.g., zyre.
- Parameters
  - ifname: Network interface to bind to.

1 https://github.com/zeromq/zyre/

- Port: Port to use for UDP beacons.

There are several services that can be set up using the swarmros bridge with the swarmio library:

- Telemetry topics: The bridge will subscribe to these topics and forward their latest value to telemetry subscribers.
  - Source: The ROS topic that provides the value for the property - preferably latched.
  - Message: Fully qualified name of the underlying ROS message type.
  - Name: The discoverable name of the topic.
  - Status: Whether to include the value of this property in the status broadcast.
- 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 which must have a header field of message type swarmros/EventHeader.
  - Source: ROS topic to forward events from.
- Incoming events: The bridge will listen to these events and republish them under ROS topics. Only one handler per event name can be added. If desired, handlers with the same message type can republish to the same topic.
  - Suffix: Name for the local topic which will be published under events/<suffix>.
  - Message: Fully qualified name of the underlying ROS message type which must have a header field of message type swarmros/EventHeader.
  - Name: Discoverable event name.
- 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. 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: Name for the local parameter which 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.
- Bridged parameters: The bridge will load the value of these parameters from the ROS Parameter Server, then publish them as remotely available key-value targets. 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.
  - Name: Discoverable key-value path.
  - Path: ROS parameter path.
  - Rw: Whether set requests are accepted.

Generally, the modelling of communication using the publisher/subscriber concept will have the form as shown in Figure 13.

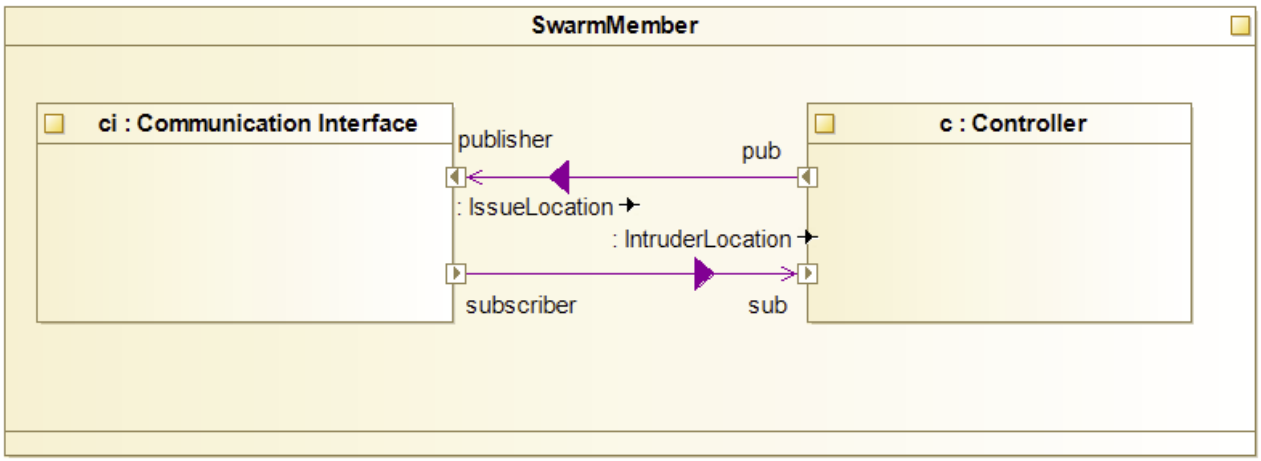

Figure 13 Communication modelling

6 Human in the loop

This section updates the human-in-the-loop concept as proposed in D4.1 - Initial CPS Modeling Library. In D4.1 the authors described the main roles of humans in the design of the CPSwarm workbench adapted from D2.1 – Initial Vision Scenarios and Use Case Definition. These roles included the Modeller, Software Developer, Engineer and Operator. As this distribution is not enough to characterize the human as part of a CPSwarm project, this document proposes a three-part approach as sketched in Figure 14 In the following subsections the three key elements will be detailed.

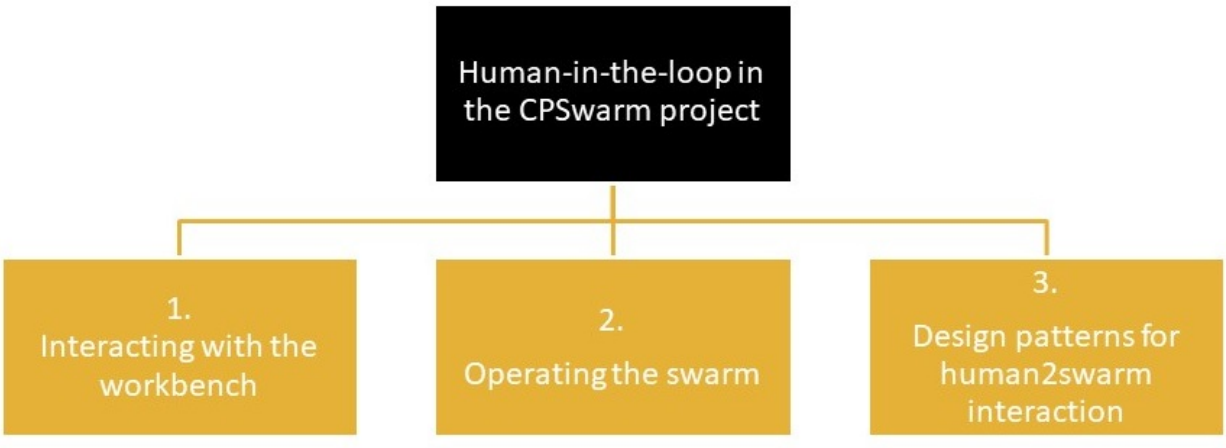

Figure 14 General concept for human-in-the-loop in the CPSwarm project

6.1 Interacting with workbench

Different activities in a development team of the CPSwarm Workbench are related to individual roles. Within D2.1 – Initial vision scenarios and use case definition these roles were identified (see Figure 15) and relevant ones are described in more detail in D4.1 - Initial CPS Modeling Library.

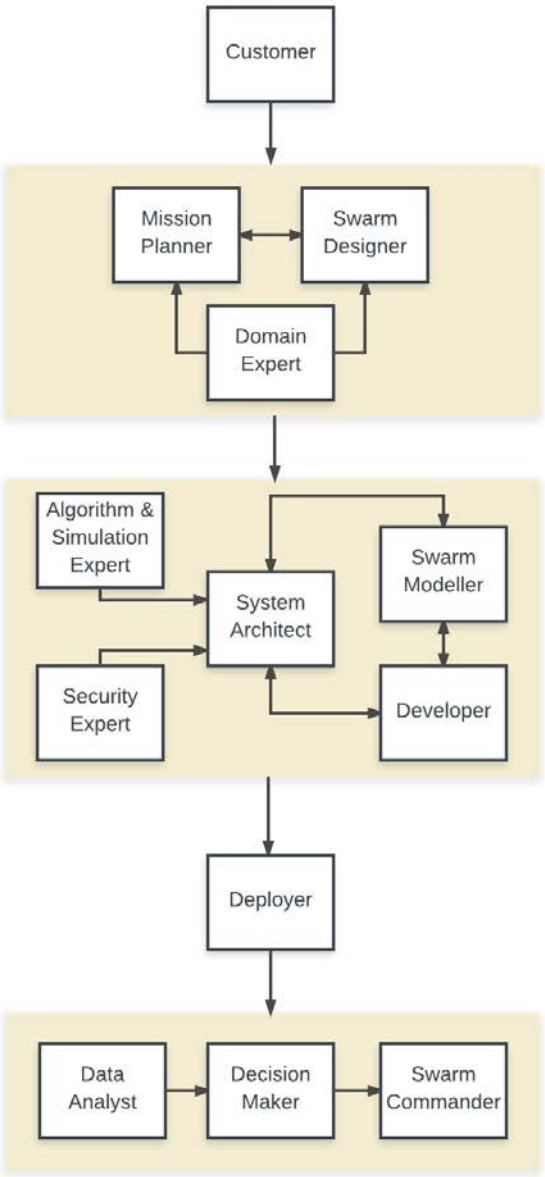

Figure 15 Roles in the CPSwarm Workbench and the communication flow among them.

6.2 Operating the swarm

The operator interacts with the swarm of CPSs through a communication interface as in Figure 16. The communication flow Human2Swarm can be in three modes: (one) human to one swarm member, (one) human to multiple (selected) swarm members, or (one) human to swarm. Independent of the mode, the operator is able to:

- get outputs from the swarm(s) via visualization
- send inputs to the swarm(s) via control inputs.

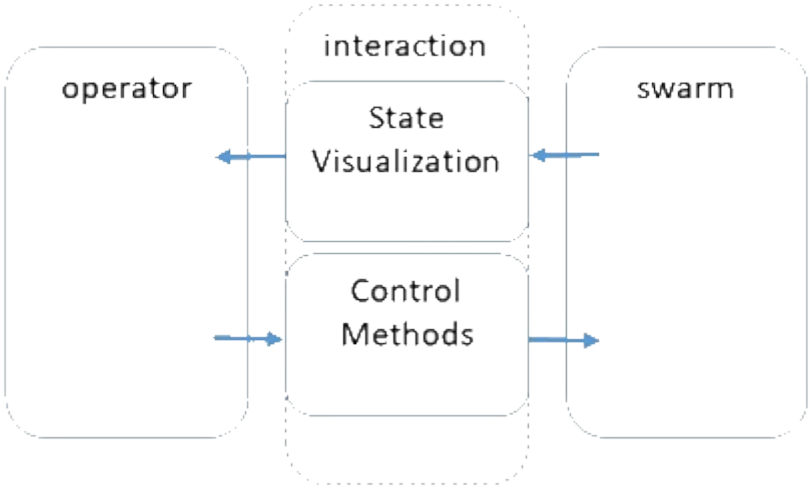

Figure 16 Human-Swarm Interaction Model (adapted from (A. Kolling, 2016))

Getting and setting inputs and outputs needs to be defined already in the modelling stage of the swarm. Therefore, the persons dealing with the modelling/design of the swarm of CPSs need to be aware of the information needed by the operators at the end of the information flow (see Figure 17).

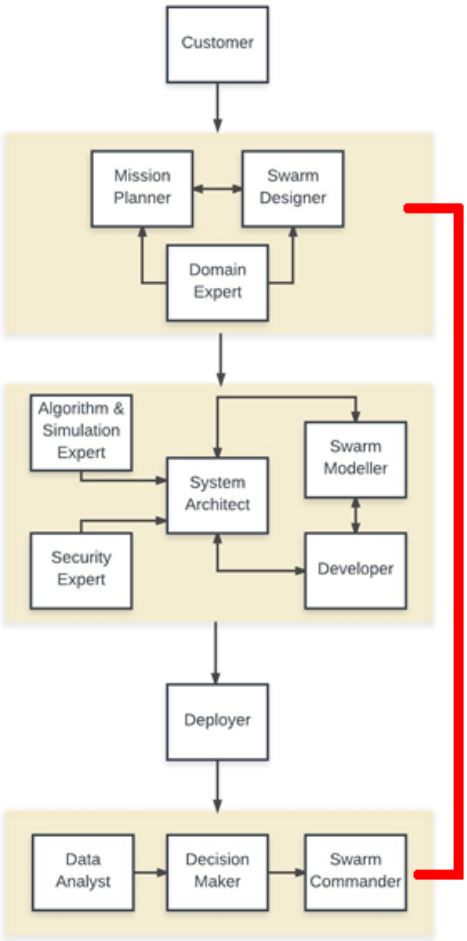

Figure 17 Combination of the designers with the operators.

6.2.1 Remote vs. Proximal Swarm Interaction

Interacting with a swarm can be further separated into remote and proximal interaction (A. Kolling, 2016). In **remote** interactions, the operator acts outside of the swarm. Typically, the operator uses a computer terminal locally separated from the swarm. Remote interaction is usually chosen, if the swarm needs to enter dangerous or inaccessible environments. However, the concept contradicts to the distributed nature of swarms with a human as central control.
Typical examples of existing concepts for remote control include, e.g., the SwarmCraft (James McLurkin, 2006) in Figure 18. The central GUI is inspired by the concepts of video games and graphical software debuggers. Basically, it is a monitoring tool that allows to receive data from the robots about the swarm state.

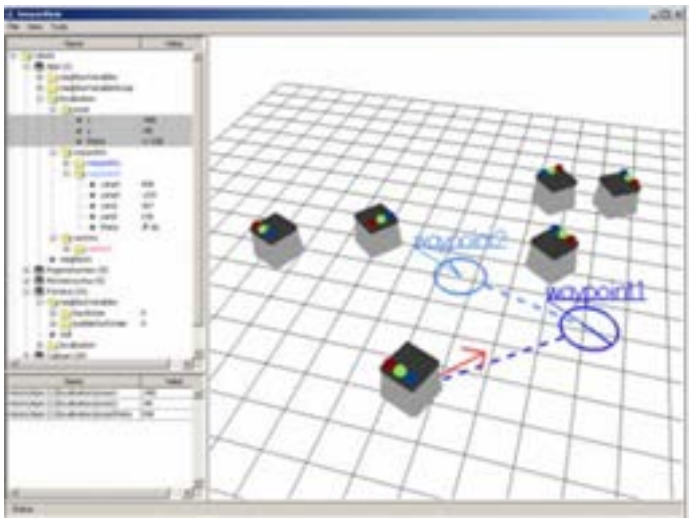

Figure 18 The SwarmCraft

A way to control single or multiple robots was presented in (Jun Kato, 2009). They propose a GUI as live video feed of multiple robots. Operators can control the paths of the robots by drawing a vector field on the screen (see YouTube Video).

In **proximal** interaction the operator shares the environment with the swarm. Thus, the operator does not have a specific location – he/she can walk and work among the swarm of CPS, or even work with the swarm as swarm member. For the interaction with the swarm, the operator can use mobile devices, mechanical or sensor-mediated interfaces, or even gestures to perform local interactions with the swarm. The only limitations come from the limited local capabilities – sensing and computational power - of the CPS itself.
Examples for gesture recognition are presented in (Alessandro Giusti, 2012) and (J. Nagi, 2014). They use gloves that are recognizable by different kinds of robots. Line of sight is required, as the robots perform a consensus on the recognized gestures to identify and follow multiple instructions, e.g., take-off, go to base or search. Therefore, all, some or individual robots can be selected (see YouTube Video). In (S. Pourmehr, 2013) they go even a step further: they create, modify and command groups of robots through speaking a number and looking at them, like "You three!".

6.2.2 Monitoring and Controlling

For a proper supervision, the operator is in need of **monitoring** the swarm. To guarantee a correct execution, the operator needs to observe multiple parameters of the swarm dependent on the application:

- Swarm state
  - Overall connectivity
- State of individual swarm members
  - Battery charge

    - Connectivity to other swarm members
- Motion of the swarm
    - Speed
    - Direction
- Location of the swarm (members)
    - Location of individual swarm members
    - Location of center of mass of the swarm
- Sensor Values
    - Video Streams
    - Temperature, Pressure, etc. values

The observation needs to be provided in an optimal way, so the operator is able to react accordingly fast and intuitive. Thus, visualization is of great importance, together with a facilitation of swarm dynamics and impacts of control inputs. Multiple studies already exist that analysed swarm visualizations, including latency in communication channels between swarm and human (P. Walker, 2012), multimodal feedback in terms of speed, strength, capability and dispersion (Ellen C. Haas, 2009). Furthermore, the operator should be able to predict future states/locations/etc. on the current set of received data (A. Kolling, 2016). Forecasting swarm behaviour is necessary to decide on future control inputs, if needed. In (Seyed Behzad Tabibian, 2014) they studied whether humans are able to predict the output behaviour of a simulated swarm related to their given input. Moreover, in (Sasanka Nagavalli, 2015) they did a similar study focusing on the timing of an operator's control input. Their task was to give commands to move the swarm from one to another location. The operators had to give the input at the time that would minimize the convergence time.
In several cases an intervention to swarm behaviour is necessary. This is done by the operator that can **control** the swarm. Controlling a swarm can be split in various tasks that need a varying representation. The authors identified following groups

- Simple commands (e.g. start, stop)
- Selection (e.g. x swarm members out of the swarm)
- Switching behaviour (e.g. from flocking to relaying): to switch the behaviour of the swarm (the running swarm algorithm), the operator needs to have the possibility to choose between available algorithms.
- Adjusting parameters (parameters from a swarm algorithm, e.g. coherence): typically, each swarm algorithm has parameters that can be manipulated. One parameter of a flocking algorithm is, e.g., coherence, another one is speed. Changing these parameters doesn't directly switch the behaviour type, but rather fine-tunes the emergent swarm behaviour. Changing parameters – nevertheless – is a critical action. For example, if you lower the value for coherence in flocking too much, the swarm of CPSs could have problems in exchanging information due to a large distance between individual swarm members.

6.2.3 Concept of Using Parameters to utilize Human-Swarm Interaction

To utilize the human swarm interaction, the authors establish a new concept of incorporating parameters that influence the swarm behaviour. Parameters like, e.g., coherence, speed or energy efficiency of a swarm, can be used by the operator to control the swarm and tune its behaviour during mission. These parameters need to be defined already during the design phase – in the CPSwarm Workbench it is through the modelling tool that the user can define such parameters. Finally, these parameters are used at the monitoring/control panel of the operator to manipulate the swarm behaviour in its current application. Therefore, two sets of parameters are distinguished:

1) Static Parameters

These parameters allow the user to monitor/manipulate the swarm behaviour on a pre-defined range. Here there are two types of parameters: the ones used to monitor and ones used to control the swarm. Typically, these are simple commands.

2) Evolvable Parameters

Evolvable parameters crucially characterize the swarm behaviour. They are part of the swarm algorithm and therefore, are evolved together with the swarm behaviour. Important characteristics are:

a) To achieve an effect, the parameters need to be part of the fitness function.
b) When the behaviour is evolved with the Optimization tool (i.e. FREVO), the scope of the parameters is evolved as well.
c) The resulting ANN uses those evolved parameters as input.
d) The operator's control panel presents the evolved parameter as pre-configured to the optimal value. Furthermore, the parameter can be still controlled by a GUI at the operator's side in a given range – defined by the evolutionary process.
e) The parameters serve as dynamic input to the swarm members.

These parameters can be mapped as an example to the search and rescue applications of drones in the following way:

1) Static parameters

Monitoring

- Select visualization of received images (single image from single drone, single images from multiple drones, stitched overview image)
- "Victim detected" visualization
- Navigational information from the safeguard to the victim
- Environmental conditions (wind speed, temperature, humidity)
-

Controlling

- Select a group/single drones
- Set a GPS coordinate that is used as a central point, where the swarm of drone flocks around
- Switch between flocking and relaying application
- Follow the safeguard
- "Go Home" button

2) Evolvable Parameters

- Coherence of the swarm of drones
- Altitude
- Speed
- Topology
- Flock Size

6.2.4 Performance Prediction

Evolving the swarm behaviour using evolutionary methods requires evaluation of the swarm algorithm through simulations. The number of required simulations is

$$n_{sim} = n_{gen} \cdot n_{pop} \cdot n_{eval}$$

where n_{gen} is the number of generations, n_{pop} is the number of candidates in one generation, and n_{eval} is the number of evaluations for each candidate. Assuming a typical optimization process with $n_{gen} = 200$, $n_{pop} = 50$, and $n_{eval} = 10$, every optimization requires

$$n_{sim} = 200 \cdot 50 \cdot 10 = 100{,}000$$

simulations.

When evolvable parameters are considered, the optimization needs to be performed for all values of each parameter. Hence, each parameter increases the number of optimizations by a factor of n_{values}, the number of different values the parameter can have. This yields the total number of simulation runs

$$n_{sim} = n_{gen} \cdot n_{pop} \cdot n_{eval} \cdot n_{values}^{n_{params}}$$

with n_{params} parameters.
Assuming a parameter is evolved for $n_{values} = 10$ values, every parameter increases the number of simulations ten-fold. E.g., $n_{params} = 3$ evolvable parameters lead to a total number of simulations of $n_{sim} = 200 \cdot 50 \cdot 10 \cdot 10^3 = 100{,}000{,}000$.
The simulations of one generation can be parallelized on $n_{threads}$ threads reducing the number of sequential simulations to

$$n_{sim} = n_{gen} \cdot \frac{n_{pop} \cdot n_{eval}}{n_{threads}} \cdot n_{values}^{n_{params}}$$

given that $n_{threads}$ is within the range $[1, n_{pop} \cdot n_{eval}]$.
If parallelization is used, in the best case with $n_{threads} = 500$ parallel simulations, the number of sequential simulations to be performed can be decreased to $n_{sim} = 200 \cdot \frac{50 \cdot 10}{500} \cdot 10^3 = 200{,}000$.

These preliminary calculations show us that we need to take care of the number of parameters to be evolved. We will elaborate approaches in evolution to reduce the simulation runs or limit the input-output relation to it. Nevertheless, we will investigate on this approach as it could enrich the CPSwarm workbench with an innovative human-in-the-loop concept.

6.3 Design patterns for human2swarm interactions

When designing a CPS there are rules, guidelines and safety regulations that must not be violated. Similar to Asimov's Three Laws of Robotics (Asimov, 2013), it is needed to define the elementary laws for swarms of CPSs. These rules act as a 2nd level system, no matter which task is executed. However, these aspects have another quality of information that is mostly formulated in a non-formal manner – usually in natural language. This kind of information must be "in the back of the engineer's head" when deriving and implementing concepts. Thus, it is not a technical asset that can automatically be integrated and checked. However, it can be made available as library knowledge accessible to users of the CPSwarm Workbench. This knowledge is either written in existing guidelines, laws and regulations, or often captures within existing design existing in real world implementations or within evaluated concept. In many cases, first ideas exist that need to be researched further. However, since these pieces of knowledge cannot always be formalized but furthermore represent inherent qualities. For this reason, it cannot be put in software-libraries for reuse.

Building on top of laws and rules, there is yet another, less demanding layer that covers knowledge about successful and accepted human-swarm system design. The questions go into the direction of finding good means of interaction between human and swarm or single swarm member, respectively. In addition, a swarm's behavior, i.e. supporting vs. guiding, for example, is to be defined in a proper and accepted way suiting the needs for a specific situation that are definitely different in a warehouse compared to a disaster site.

For well-proven implementations and design decisions, the notion of "design patterns" has been successfully instrumented over the last 40 years. The pattern concept is well known and capable of capturing working solutions to recurring problems that a community of experts has developed over time. Patterns originate from the architectural domain where Christopher Alexander presents them as readily formulated pieces of solutions for general design problems (Alexander, 1977). Gamma et al. transferred the idea to software design (Gamma, Helm, Johnson, & Vlissides, 1994). Other areas such as user interface design (Tidwell, 2011), human-computer interaction (Borchers, 2001) or website design (van Duyne, Landay, & Hong, 2007) use the pattern concept to share their knowledge. Several other non-technical domains adopted the concept and therewith formulate patterns for a variety of topics. Manns and Rising present patterns for the introduction of organizational process changes (Manns & Rising, 2005). Coplien and Harrison give organizational advice through patterns (Coplien & Harrison, 2005) and Bergin et al. present patterns dealing with teaching aspects (Bergin, et al., 2012).

Patterns are usually formulated in prose in order to avoid specific vocabulary and keeping the formulations understandable for all stakeholders that are involved in the design process. Recent research takes into account semi-formal and formal approaches in order to automate the structuring, retrieval and selection processes for patterns (Cornlis & Hedin, 2000), (Eden, Yehudai, & Gil, 1997) which, however, are not considered in the project.

The general structure of a design patterns covers at least the following fields and is adapted according to the needs of pattern authors or domains in which they are used, respectively:

- A *Name* that clearly expresses the central idea of the pattern.
- A brief description of the *Problem*.
- A *Solution* to the problem.

Often, the following fields are taken into account additionally, like in this project:

- The *Context* describes in which design situation the problem occurs ranging from high-level concepts up to details that occur late in the design or conceptual process.
- *Illustrations* are closely connected to the name, such that the reader receives a sensitizing example for the pattern's application and can quickly grasp the main idea of the pattern.
- Examples and Discussion intend to help the reader to inductively follow the solution that is suggested by the pattern suggests.
- References connect patterns to each other in case they are related by presenting additional information or alternatives to the proposed solution.

The CPSwarm project sees patterns not only as a way to capture and represent design knowledge, but also as means of communication and documentation within and beyond the current project knowledge and the dynamic requirements engineering process. This way, they are used to capture domain knowledge, processes and technical knowledge during all phases of the project lifecycle from many perspectives. Similar to Erickson (Erickson, 2000), patterns are regarded as a lingua franca between stakeholders.

First ideas for pattern formulation originate from work in the requirement engineering process and are based on a first collection illustrated by the mind map in Figure 19 and cover the following aspects:

- Identification of living beings
- Identification of specific human beings
- Policies for Human-Swarm interaction
- Sensing distance
- Collision avoidance
- Constraints
- State Dependency
- Tamper detection
- Escape
- Self-destruct behavior

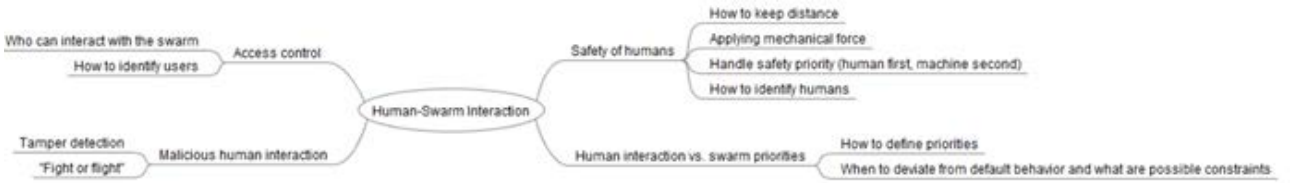

Figure 19 Aspects of Human-Swarm interactions to be considered.

In the next project phase, workshops will be held within the consortium to start formulating pattern on top of these aspects.

Evolving Pattern Formulation, Maturation and Organization

With a growing number of patterns, their organization plays a more important role. Different approaches keep patterns as loose collection from with readers select by skimming though it. Gamma et al. structure derived sets of patterns, e.g., by purpose, context or behavior, making it easier for readers to mentally order the patterns and focus on current needs. Pattern languages interconnect single patterns and therewith provide a context, in which a single pattern can be applied. References to related patterns reveal additional details as pattern combinations or alternatives. Thus, pattern languages can be represented as directed acyclic graph structures according to the formal definition for pattern languages as shown by Borchers (Borchers, 2001). In a first step, patterns will be categorized and published in an online design pattern library. First ideas for categories that are subject to change are:

- Laws and Ethics
- Safety
- Technology and System Design
- Human-Swarm Interaction Design
- Data Management
- Privacy and Security

Over time, connections between patterns may evolve and can be added later. Moving patterns into different categories or introducing hierarchies between them is possible from the concept that supports the idea of an evolving pattern library.

An important aim of the approach is the parallel documentation of findings and updating the collection of requirements while performing research and development. In order to integrate as many findings as early as possible in the pattern collection, the presented approach lowers the threshold for contributing. This allows contributors to provide early formulations that may range from considerations and incomplete pattern ideas up to acknowledged and proven solutions according to the original pattern concept. The patterns are created in parallel to the engineering process as described in (René, 2014). This approach will be adapted to this project and thus will take into account that early formulations for discovered problems evolve over time and reflect the current engineering progress. Figure 20 illustrates the pattern formulation and maturation process.

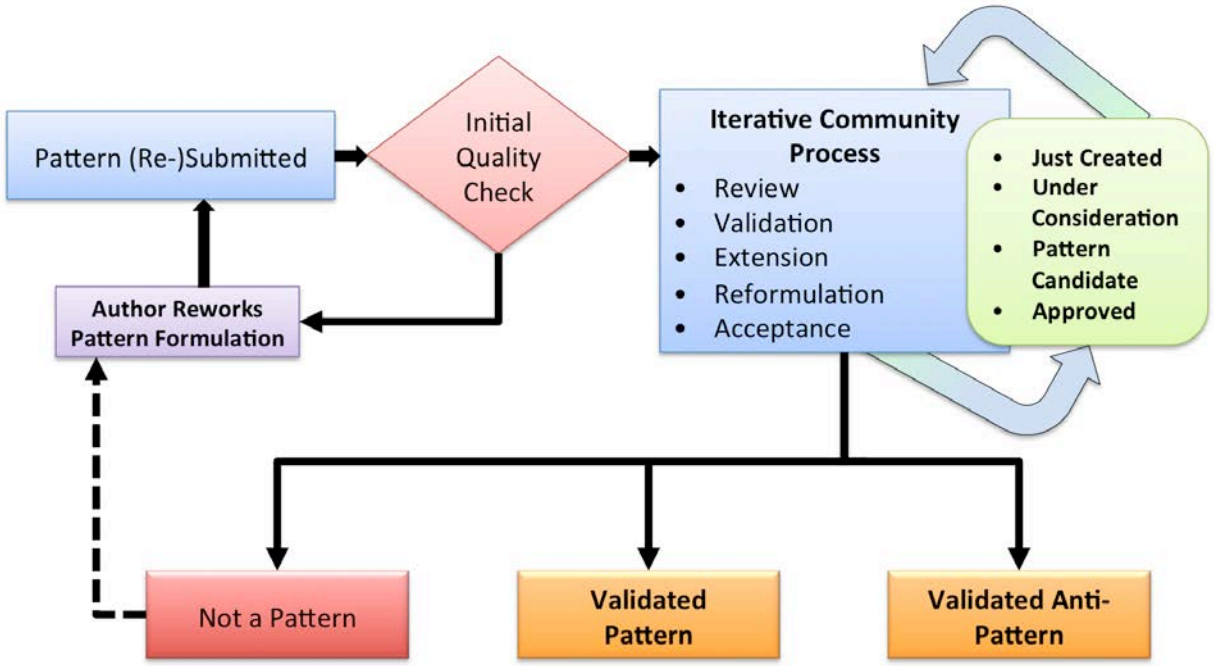

Figure 20 The CPswarm Design Pattern Formulation and Maturation Process.

Accordingly, a pattern first represents an open problem that is refined and updated over time until both - problem and solution(s) are found. Clarifications and additional explanations are provided by the feedback from the project members that rate on the factors readability, understandability of the solution and relevance to the project scope.

The entire derivation of the process, its rules and role model together with the abstract formulation of the different criteria for a pattern's maturity state is described in (René, 2014). Each newly submitted pattern undergoes a first semi-automated quality check process meaning that the system first ensures that all required pattern fields are actually filled.

The submission is then forwarded to a group of pattern reviewers that decide whether the pattern can be published in the library or if certain formulations need to be changed again by the author. This quality check avoids flooding of the pattern library with inappropriate content. After a submission has passed the quality check, it is published in the library and is therewith open for the community-based discussion within the community.

Meszaros and Doble (Meszaros & Doble, 1997) provide additional hints for a pattern's formulation in their ''Pattern Language for Writing Patterns''. In summary, they state that patterns should be:

- Readable such that quick parsing of the contents is possible
- Tailored to the target audience, i.e., in the context of this work, the project domain and work package tasks
- Understandable by the audience
- Structured such that the meaning of the different parts of a pattern is clear
- Self-explanatory by providing meaningful names and examples as well as using a common notation that is understood by every member of the audience

After its publication within the pattern library, every registered user of the system can provide feedback to the pattern, its formulation and support or refute the pattern statement by providing more evidence in favor of the pattern or against it. Evidence can originate from references from literature or realizations in products or applied processes that support or refute the pattern's validity.

On the right, Figure 20 shows different maturity states a pattern undergoes during the described process. From an initial pattern idea that is submitted as an ''open problem'', i.e. a requirement, it develops from a pattern ''under consideration'' to the more mature states ''pattern candidate'' and ''approved pattern''. These states reflect the usage and support of the pattern and tie the initially stated problem to the proposed solution.

In order to pass the state from an open problem to a pattern ''under consideration'', an eventually not yet validated solution must be added to the pattern. The approach assumes that proposed solutions are implemented within the project and validated. The results of the validation are fed back as evidence to the pattern. In addition, assuring the pattern formulation quality is the aim of the threshold to the next maturity state. In order to pass, the project stakeholders review the pattern formulation and provide feedback and rating with regard to different aspects. These can be its readability, understandability and its appropriateness for the current project. The needed amount of reviews, considered aspects and needed rating scores can be freely defined. After the formulation quality threshold is passed, the formulation becomes a ''pattern candidate'' that needs to be evaluated. Based on validation or research related to the stated problem, evidence that support or refute the pattern is collected. Again, the needed number of pieces of evidence as well as their individual weights can freely be defined.

After this last threshold is passed, the problem formulation and solution finding process ends. The pattern now encapsulates the history from an initial problem to a proven solution. In case that the pattern library is used as knowledge repository for several current or future projects, the correctness of a pattern needs to be checked periodically. Eventually, the maturity state needs to be revalidated and formulations within the patterns need to be updated.

Besides the found and proven solutions, surprisingly failing solution attempts are also part of the library and denoted as anti-patterns as described in (Reiners, Astrova, & Zimmermann, 2011). They also represent important knowledge that must be considered in future, similar problems such that repeating errors can be avoided and therefore save development time. Additionally, possible solutions should be documented to open the design space such that new ideas can be generated while taking up inspiration from existing ones.

Throughout the remaining project duration, the online pattern library will be implemented and used for collaborative pattern formulation. First pattern formulation workshops are planned for November/December 2018 and will be held consecutively in 2019. A first concept for the design pattern library is shown in Figure 21 and will be available online[2].

[2] https://patterns.fit.fraunhofer.de/cpswarm/index.php/browse-patterns

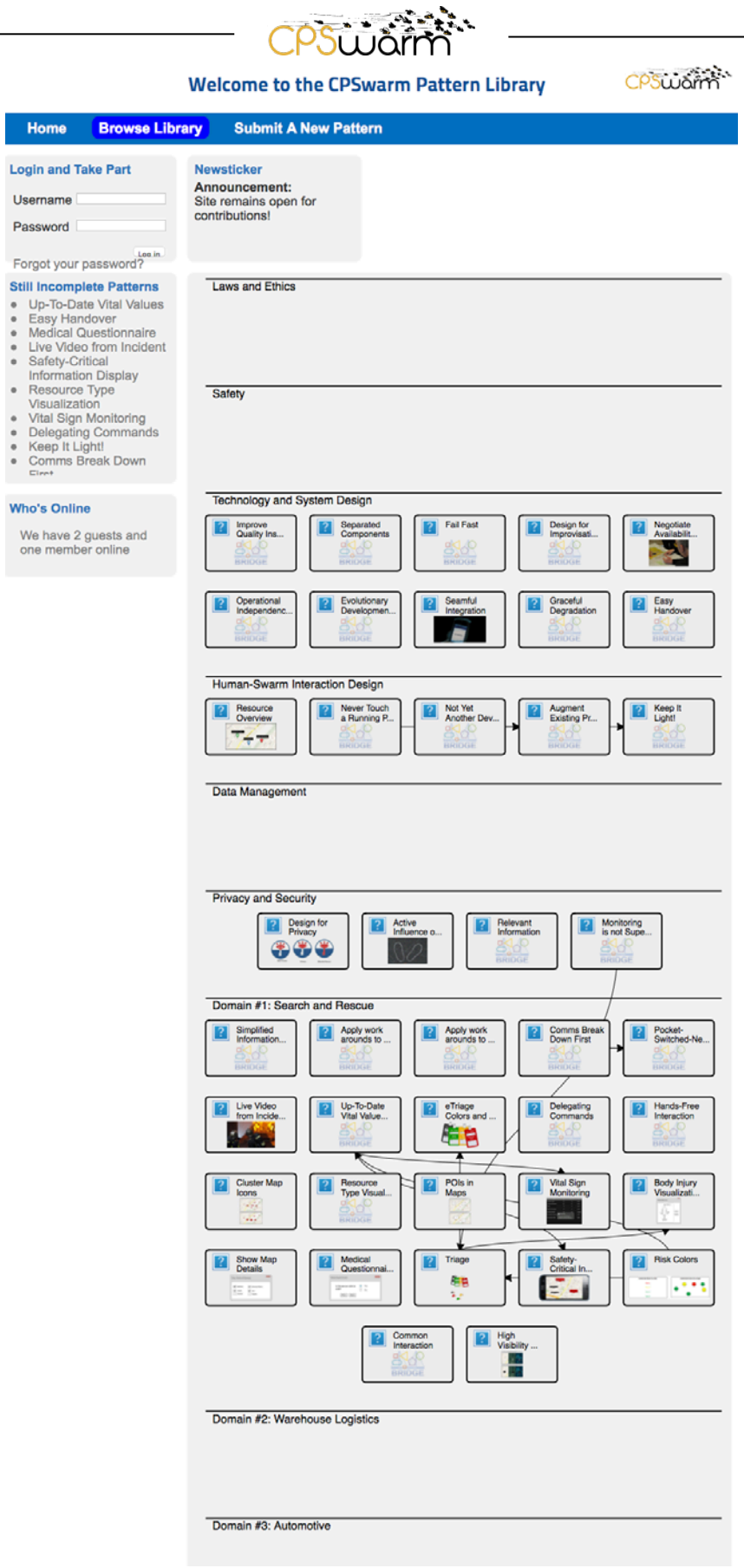

Figure 21 First concept for the online version of the CPSwarm Evolving Design Pattern Library.

7 Conclusions

This deliverable presents the status of the available CPS models at M20 of CPSwarm project. Even if the presented models (CPSwarm library, CPS hardware and behaviour aspects, communication, and human in the loop) have not yet been fully developed, they have been already used inside both research and industrial case studies at different level.

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

References

A. Kolling, P. W. (Feb. 2016). Human Interaction With Robot Swarms: A Survey. *IEEE Transactions on Human-Machine Systems, 46*(1), S. 9-36.

Alessandro Giusti, J. N. (2012). Human-swarm interaction through distributed cooperative gesture recognition. *7th ACM/IEEE International Conference on Human-Robot Interaction*, S. 401-402.

Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction.* New York, USA: Oxford University Press.

Asimov, I. (2013). *Runaround in I,Robot. arperVoyager.*

Bergin, J., Eckstein, J., Völter, M., Sipos, M., Wallingford, E., Marquardt, K., . . . Manns, M. L. (2012). *Pedagogical Patterns: Advice For Educators.* CreateSpace Independent Publishing Platform.

Borchers, J. (2001). *A Pattern Approach to Interaction Design.* West Sussex, England: John Wiley & Sons.

Coplien, J. O., & Harrison, N. B. (2005). *Organizational Patterns of Agile Software Development.* Hamilton, NY, USA: Pearson Prentice-Hall.

Cornlis, A., & Hedin, G. (2000). Tool Support for Design Patterns Based on Reference Attribute Grammars. *Proceedings of WAGA'00*.

Eden, A. H., Yehudai, A., & Gil, J. (1997). Precise Specification and Automatic Application of Design Patterns. *Proceedings of the 12th IEEE International Conference on Automated Software Engineering*.

Ellen C. Haas, M. F. (2009). *Extreme Scalability: Designing Interfaces and Algorithms for Soldier-Robotic Swarm Interaction.* DTIC Document.

Erickson, T. (2000). Lingua Francas for Design: Sacred Places and Pattern Languages. *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, S. 357-368.

Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994). *Design Patterns. Elements of Reusable Object-Oriented Software.* Amsterdam: Addison-Wesley Longman.

Gutierrez, J. A., Callaway, E. H., & Barrett, R. (2003). *IEEE 802.15.4 Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensor Networks.* IEEE Press.

J. Nagi, A. G. (2014). Human-swarm interaction using spatial gestures. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 3834-3841.

James McLurkin, J. S. (2006). Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots . *Proc. AAAI Spring Symposium*, S. 72-75.

Jun Kato, D. S. (2009). Multi-touch Interface for Controlling Multiple Mobile Robots. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, S. 3443–3448.

Manns, M. L., & Rising, L. (2005). *Fearless Change: Patterns for Introducing New Ideas: Introducing Patterns into Organizations.* Boston, MA, USA: Addison-Wesley.

Meszaros, G., & Doble, J. (1997). A Pattern Language for Pattern Writing. In R. C. Martin, D. Riehle, & F. Buschmann, *Pattern Languages of Program Design 3* (S. 529-574). Boston, MA, USA: Addison-Wesley Longman Publishing.

P. Walker, S. N. (2012). Neglect benevolence in human control of swarms in the presence of latency. *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, S. 3009-3014.

Reiners, R., Astrova, I., & Zimmermann, A. (2011). Introducing new Pattern Language Concepts and an Extended Pattern Structure for Ubiquitous Computing Application Design Support. *Proceedings of the 3rd International Conference on Pervasive Patterns and Applications*, S. 61-66.

René, R. (2014). *An Evolving Pattern Library for Collaborative Project Documentation.* Shaker.

S. Pourmehr, V. M. (2013). “You two! Take off!”: Creating, modifying and commanding groups of robots using face engagement and indirect speech in voice commands. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 137-142.

Sasanka Nagavalli, S.-Y. C. (2015). Bounds of Neglect Benevolence in Input Timing for Human Interaction with Robotic Swarms. *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, S. 197-204.

Seyed Behzad Tabibian, M. L. (2014). Towards a Cognitively-Based Analytic Model of Human Control of Swarms. *AAAI Spring Symposium Series*.

Tidwell, J. (2011). *Designing Interfaces.* Sebastopol, CA, USA: O’Reilly Media.

van Duyne, D. K., Landay, J. A., & Hong, J. (2007). *The Design of Sites: Patterns for Creating Winning Websites.* Prentice Hall.