D4.1 INITIAL CPS MODELING LIBRARY

Deliverable ID	D4.1
Deliverable Title	**Initial CPS modeling library**
Work Package	**WP4 – Models and algorithms for CPS Swarms**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**06-10-2017**
Status	**FINAL**
Lead Editor	**Alessandra Bagnato (SOFTEAM)**
Main Contributors	**Melanie Schranz (LAKE), René Reiners (FRAUNHOFER), Etienne Brosse (SOFTEAM), Wilfried Elmenreich (UNI-KLU), Omar Morando (DGSKY), Enrico Ferrera (ISMB), Claudio Pastrone (ISMB), Bálint József Jánvári (SLAB), Regina Krisztina Bíró (SLAB)**

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	2017-05-16	Alessandra Bagnato (SOFT)	First Draft
0.2	2017-06-07	Melanie Schranz (LAKE)	Extending the TOC with Chapter 4, 5 and 6
0.3	2017-06-07	Alessandra Bagnato and Etienne Brosse(SOFT)	Adding foreseen content for Sections 4.1, 4.2, 4.3 (examples of models)
0.4	2017-08-06	Melanie Schranz (LAKE)	Adding content for 6.2.2 Human-Swarm Interaction
0.5	2017-08-27	Alessandra Bagnato (SOFT)	Integration of various contributions
0.6	2017-09-04	Alessandra Bagnato (SOFT)	Integration of SLAB & FIT contributions
0.7	2017-09-08	Alessandra Bagnato (SOFT)	Integration of LAKE contributions, added a paragraph to the conclusion to show the relation to D4.4
0.71	2017-09-11	Etienne Brosse (SOFT)	Comment response
0.72	2017-09-12	Alessandra Bagnato (SOFT)	Integration of ISMB contributions on IoT Entity Modeling background
0.73	2017-09-12	Alessandra Bagnato (SOFT)	Integration of LAKE contributions on 7.2 and conclusions
0.8	2017-09-29	Alessandra Bagnato (SOFT)	Addressing LAKE and ISMB review comments
0.81	2017-09-29	Alessandra Bagnato (SOFT)	Including FIT feedback and ISMB revision to IoT modelling background
1.0	2017-10-06	Alessandra Bagnato (SOFT)	Final Integrated Version

Table of Contents

1 Executive summary

This deliverable, namely "D4.1 Initial CPS modeling library", introduces the whole new CPSwarm initial modelling library and its initial Catalogue for CPS Models.
The deliverable starts introducing the background state-of-the-art of our CPSwarm first modelling library. We present the existing approaches that were used to be adapted/extended for swarm integration and where new CPS models must be developed. More specifically, in the following we describe the available open source approaches and examples we considered while discussing Embedded and Cyber-Physical Systems (CPS) modelling, Security Modeling and IoT Entity Modeling. We already used part of the background described in this deliverable as a basis for the whole new CPSwarm initial modelling library that is described in Section 4 and we will keep using it for the future versions of the library.
The deliverable then presents the structure of the Emergency Exit example of the Initial CPS modeling library, the initial reasoning on the communication among the Models and the Human-in-the-Loop of the models that will be further extended in the future versions of this deliverable.

This deliverable, D4.1 Initial CPS modeling library, is a software deliverable that include the CPSwarm developed models till M9 within WP4, the Emergency Exit example models are available on the Modelio forge at http://forge.modelio.org/projects/cpswarm.

2 Introduction

Cyber-Physical Systems (CPS) find applications in many large-scale, safety-critical domains e.g. transportation, smart cities, etc. As a matter of fact, the increasing interactions amongst different CPS are starting to generate unpredictable behaviours and emerging properties, often leading to unforeseen and/or undesired results. Rather than being an unwanted by product, these interactions could, however, become an advantage if they were explicitly managed, and accounted, since the early design stages. The CPSwarm project,
aims at tackling these kinds of challenges by easing development and integration of complex herds of heterogeneous CPS. Thanks to CPSwarm, systems designed through a combination of existing and emerging tools, will collaborate on the basis of local policies and exhibit a collective behaviour capable of solving complex, real-world, problems [32].
D4.1 – "Initial CPS modeling library" is a public document defining the CPSWarm publicly available CPS models defined till CPSwarm at M9 of the project.
The models are developed by the project team and hosted on the CPSwarm Modelio forge at http://forge.modelio.org/projects/cpswarm
The deliverable introduces the models and the modelling approaches that allowed the team to create the initial CPSwarm modeling library. More specifically, in the following we describe the available open source approaches and examples we considered for discussion as the Embedded and Cyber-Physical Systems (CPS) modelling, Security Modeling and IoT Entity Modeling. We already used part of the background described in this deliverable as a basis for the whole new CPSwarm initial modelling library that is described in Section 4 and we will keep using it for the future versions of the library.
The deliverable then presents the structure of the Emergency Exit example of the Initial CPS modeling library, the initial reasoning on the communication among the Models and the Human-in-the-Loop of the models that will be further extended in the future versions of this deliverable.

Softeam, as deliverable leader, initially drafted the document, which has subsequently been enriched by all partners' contributions with existing background CPSs models' examples and the agreed CPSwarm models structure for our initial library.

2.1 Scope

This deliverable provides an analysis of the background, the agreed CPSwarm models structure and all reasoning behind the current CPS models available so far within the CPSwarm Initial Model Library.

2.2 Document organization

The remainder of this deliverable is organized as follows:

Section 3 describes the background state-of-the-art model examples of our CPSwarm first modelling library, firstly describing background of the project in Embedded and Cyber-Physical Systems (CPS) modelling thenIoT Entity Modeling. and in Security Modeling. Section 4 describes the project evolutions with respect to those modelling techniques within the whole new CPSwarm initial modelling library. Sections 5 details the EmergencyExit model example available on the CPSwarm forge. Section 6 describes Initial Communication among the Models while Section 7 details Human-in-the-Loop. Eventually Section 8 draws conclusions and provides a first feedback.

2.3 Related documents

ID	Title	Reference	Version	Date
[D9.2]	Initial Project Website and Advertising Materials	D9.2	1.0	31/03/2017
[D3.1]	Initial System Architecture Analysis & Design Specification	D3.1	1.0	30/06/2017
[D2.1]	Initial Vision Scenarios and Use Case Definition	D2.1	1.0	30/04/2017
[D5.2]	Initial CPSwarm Modelling Tool	D5.2	1.0	06/10/2017

3 Background State-of-the-Art of the CPSwarm Initial modelling library

This paragraph describes the background, state-of-the-art knowledge we took advantage of to build our CPSwarm Initial modelling library and the background we will also use for our Final modelling library.
We present the existing approaches that were studied to be adapted/extended for swarm integration and where new CPS models must be developed.
In particular, the following three sections analyse and describe the available open source approaches and examples we considered while discussing Embedded and Cyber-Physical Systems (CPS) modelling, Security Modeling and IoT Entity Modeling.
We already used part of the background described in the following Sections as a basis for the whole new CPSwarm initial modelling library that is described in Section 4 and we will keep using it for the future versions of the library. A resulting example included in the library is presented in Section 5.

3.1 Embedded and CPS modeling

The main source of background for embedded and CPS modeling for CPSwarm is the INTO-CPS project[29]. The INTO-CPS H2020 Project started in January 2015 and will last till December 2017 and aims to develop a model based tool chain for CPS development. The project integrates both Hardware-in-the-Loop (HiL) and Software-in-the-loop (SiL).

The INTO-CPS project makes use of UML and SysML [6] profiles to develop an INTO-CPS profile that is used to map to FMU concepts [1]. *Model-based systems engineering* (*MBSE*) is used for test automation, high level simulation using the FMU model exchange mechanism with simulation engine and related tools; while code generation and test automation is also in the scope of the project via the usage of tools such as RT-Tester Model-Checker, Overture resulting in platform-independent C source code.
The project aims to develop an integrated tool chain that focuses around a core Co-Simulation Orchestration Engine and involves tools such as Modelio [3], Overture, 20-Sim [2], OpenModelica and RT-Tester. Test Automation, Design Space Exploration and Code generation is also in the scope of the INTO-CPS project as showed in Figure 1 INTO-CPS Project ToolChain.

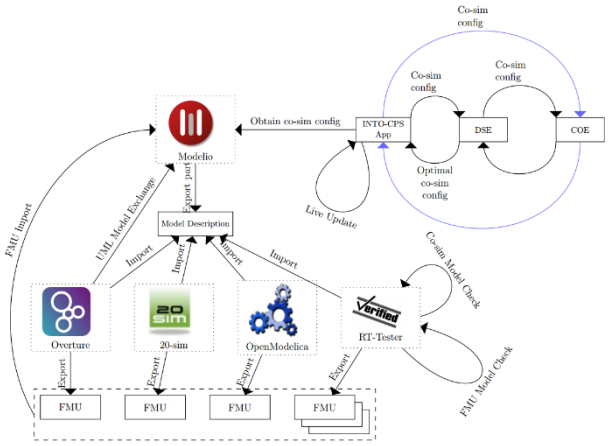

Figure 1. INTO-CPS Project ToolChain

INTO-CPS is built around the FMI (Functional Mockup Interface) [1] standard that allows FMI co-simulation and model exchange. Model-exchange takes place when the FMU is imported into a simulation tool and handled as a black box; whereas FMI's co-simulation is based on a master-slave architecture.
The SysML profile is used in the project to develop Architecture Structure Diagrams as well as Connection diagrams, resulting in the INTO-CPS UML profile, which use a subset of SysML block and internal block diagrams respectively and define specific semantics for their usage.
The INTO-CPS profile is used to develop CPS models in Modelio UML environment [7]. SysML is also mapped to FMI concepts via the INTO-CPS profile, while allows to export the high-level models in form of

automatically generated FMI which are then taken as input by the Orchestration Engine for simulation and *Design Space Exploration* (*DSE*) purposes.
The INTO-CPS project GitHub page [4][5] provides some useful modeling examples, all models are designed with Modelio open source INTO-CPS module [8]. For each example the Architecture and Connection diagrams available in the INTO-CPS profile are analysed.

3.1.1 LineFollowRobot Example

Two versions are present for an application where the intent is to control the robot's body by means of a controller and have sensors for the movement control (an optional 3D component is present for visualization in other tools).

3.1.1.1 First LineFollowRobot example

The application intent is to control the robot's body by means of a controller and have sensors for the movement control (an optional 3D component is present for visualization during FMU co-simulation):
The example models one CPS and uses an Architecture and two Connection diagrams available in the INTO-CPS profile. The example comes from Robotics domain.
The Architecture Diagram in Figure 2 indicates a system consisting of a Robot that contains a controller, body and a sensor (all subsystems). There is also a 3D visualization cyber component that is present.

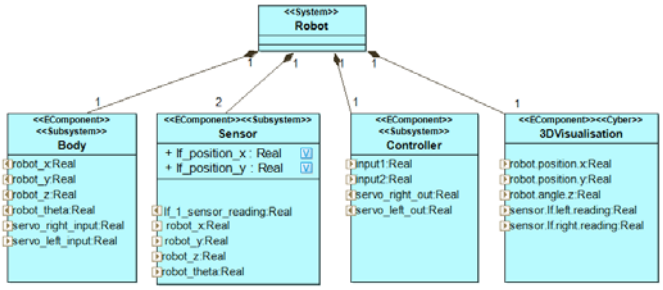

Figure 2. LineFollowRobot architecture diagram

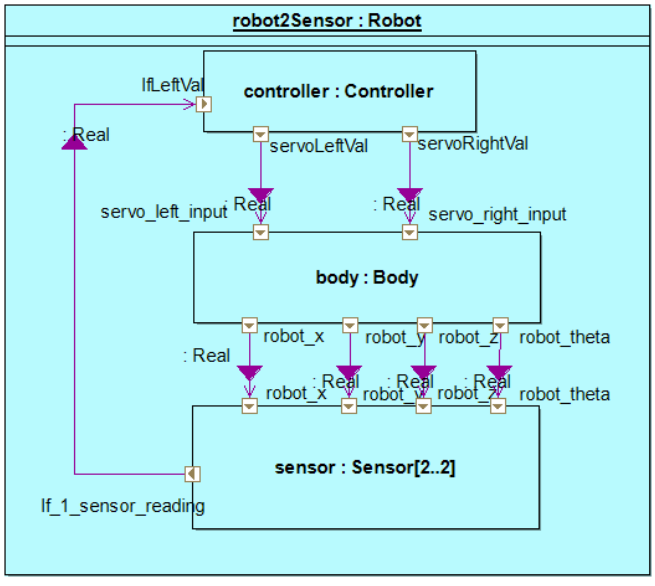

Figure 3. LineFollowRobot First Connection Diagram

The first connection diagram in Figure 3shows an instance of the robot, where the controller is connected to the body which in turn is connected to two sensor instances by connectors. The input from the sensor instances go to the controller.

The second connection diagram in Figure 4 shows a second instance of the robot, where the controller is connected to the body which in turn is connected to the two instances of the sensor, and a 3D visualization component by connectors. The input from the sensors go to the controller and the 3D visualization component.

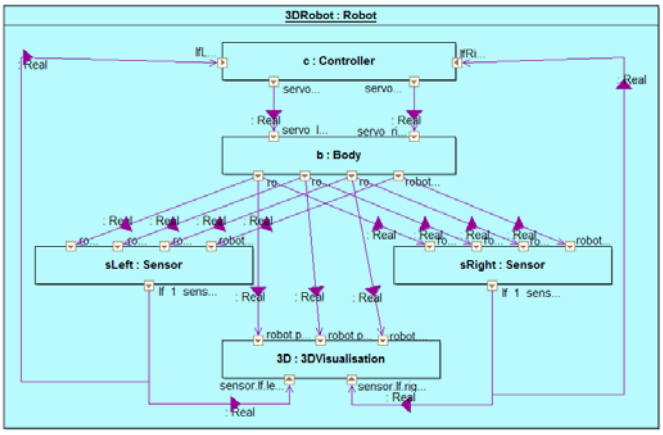

Figure 4. LineFollowRobot Second Connection Diagram

3.1.2 Single Tank Example

The Single Tank example models one CPS and uses an Architecture and a Connection diagram available in the INTO-CPS profile. The example comes from the water tank application to control the water level of tank by means of a controller.

The architecture diagram in Figure 5 indicates a system consisting of a water tank (physical component) and a Controller (cyber component) of the CPS. The connection diagram in Figure 6 indicates the flow between the water tank and the controller component.

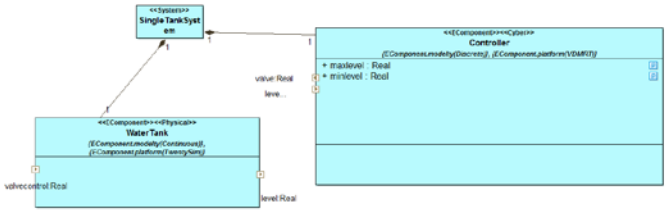

Figure 5. Single Tank Example Architecture Diagram

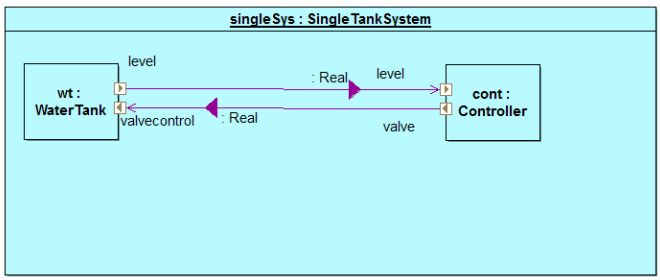

Figure 6. Single Tank Example Connection Diagram

3.1.3 Three Tank Example

The Three Tank example models one CPS and uses an Architecture and a Connection diagram available in the INTO-CPS profile. The example comes from the water tank application to control the water level of the tanks by means of a controller.

As compared to the single tank example, this example uses three tanks. The architecture diagram in Figure 7 indicates a system consisting of two water tanks (subsystem components) and a Controller (cyber component) of the CPS.
Each water tank is then composed of Tanks (physical components) and other physical components such as Pipe, Drain etc. So, there is an additional hierarchical layer as compared to the single tank example. Additional data types are also specified such as Flowrate and water level along with a user defined enumeration.

The connection diagram in Figure 8indicates the flow between the water tank 1 to water tank 2 and then from water tank 2 to the controller component. component) of the CPS. The connection diagram indicates the flow between the water tank and the controller component.

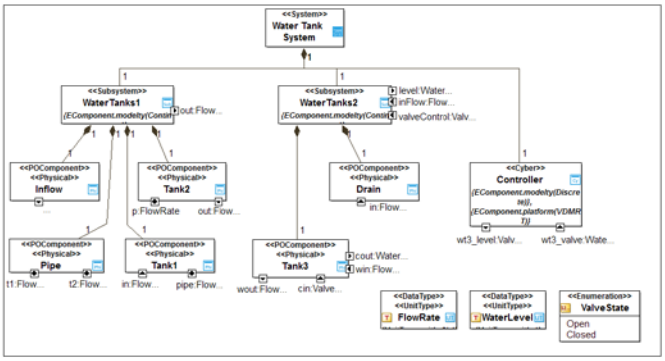

Figure 7. Three Tank Example Architecture Diagram

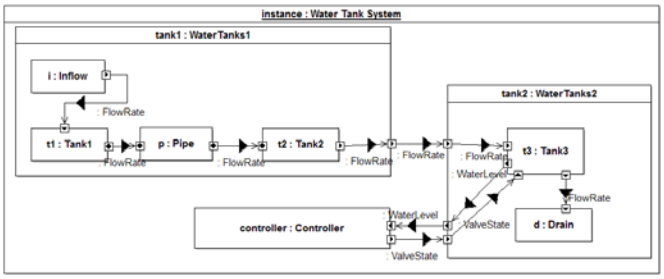

Figure 8. Three Tanks Example Connection Diagram

3.2 Security and safety modeling

Security modeling aims to find and systematically represent the ways how a system and its components can be exploited by an attacker. The following sections contain methodologies used for security modeling in CPS, with a few examples of relevant modeling tools.

3.2.1 Threat modeling in general

Usually one of the first steps of conducting a security analysis, threat modeling takes a look at the system from the attackers' perspective. Its goal is to identify high value assets, as well as potential vulnerabilities and threats. Over time, several methodologies have been developed to establish a systematic way to conduct security analysis, the most famous of which is STRIDE[1] / DREAD[2], the words themselves being mnemonics for their respective categories. Using these methodologies, threats can be identified, risks can be assessed and decisions can be made on the development of countermeasures.

3.2.2 Failure mode and effects analysis (FMEA)

As a structured and systematic technique for failure analysis, FMEA has been in use for over 50 years to assess the reliability and safety of critical systems. Its main goal is to identify failure and eliminate (or at least minimize) the number of catastrophic failure conditions. It is an inductive process, as it works with a single failure at a time and examines its effect on the system as a whole. The analysis aims to identify and eliminate all single points of failure for the system, and should be conducted in parallel with the design process, to minimize the cost of developing countermeasures. Figure 9 shows the FMEA flowchart.

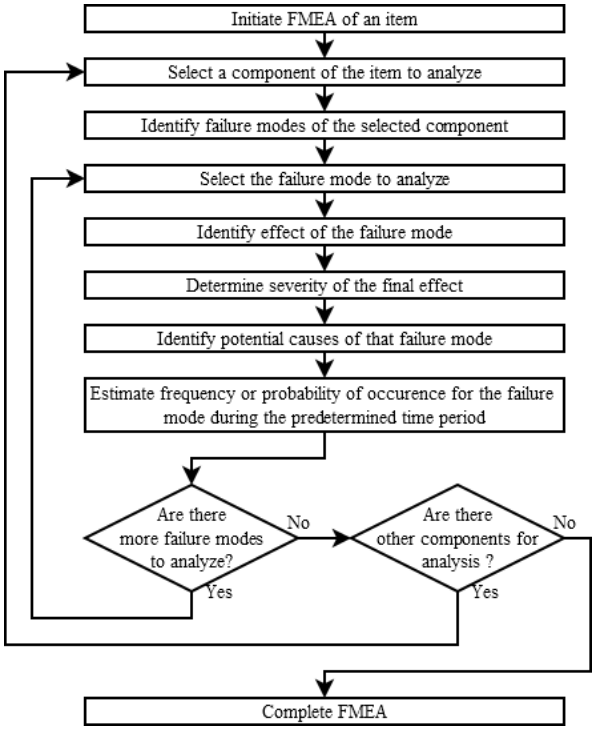

Figure 9. FMEA flowchart [15]

3.2.3 Fault tree analysis (FTA)

Using Boolean logic to combine fault indications from system components, FTA builds an event tree for system failures. Deductive reasoning is used to determine how different events contribute to a single system failure condition – starting from the top-down, from the system state that needs to be avoided, and working backwards, trying to establish when that condition can occur. When events on

[1] Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
[2] Damage, Reproducibility, Exploitability, Affected users, Discoverability

which such a tree is built are combined with their probabilities, the tree itself can be used to ascertain the probability of system failure and to identify areas in need of additional countermeasures. A welcome side-effect of the analysis is that resulting graph can also be used as a service manual for identifying the root cause of system problems. Figure 10 shows the fault tree analysis on a vehicle headlamp.

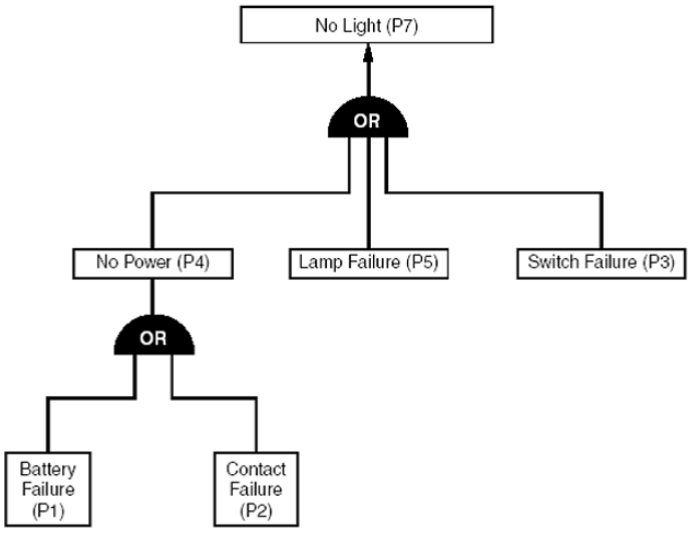

Figure 10. Fault Tree Analysis on a vehicle headlamp [16]

3.2.4 Attack impact analysis

Attack impact analysis is a similar approach to the FMEA method from the perspective of safety. The use of the attack impact analysis tool presented in [14] is to model graphically how a vulnerability can affect a given component. A simple example is presented to illustrate the method: a temperature controller with two sensors (Sensor 1 and Sensor 2) connected to it (see Figure 11). Based on the sensor inputs, the controller activates the heater or cooler components.

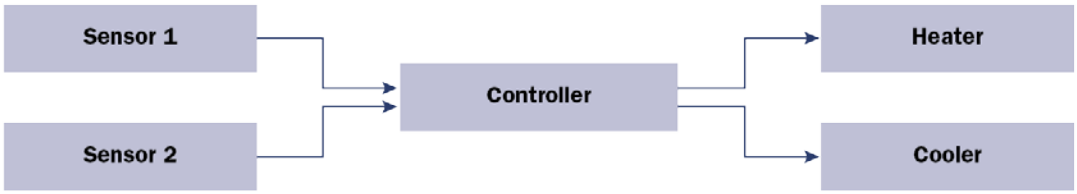

Figure 11. A temperature controller with two sensors [14]

Since it is assumed that the sensors are physically exposed to any attacker, it is possible to inject compromised data to the controller to change the outputs to the heater or cooler. The vulnerability identified here is physical exposure depicted on Figure 12 with the data flow shown between the sensors and the cooler and heater actuators.

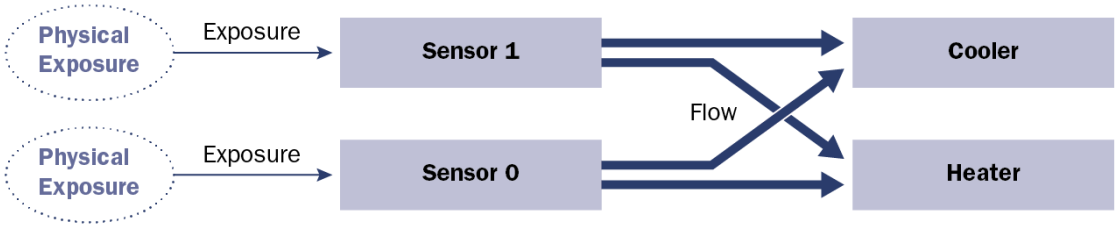

Figure 12. Physical exposure vulnerability [14]

3.2.5 Attack trees

The attack tree approach identifies possible threats with conceptual diagrams called attack trees consisting of one root node, internal nodes, and leaf nodes. From the bottom up, nodes are conditions which must be satisfied in order to make the direct parent node true. There are two types of parent nodes: *Type AND* and *Type OR as in* Figure 13. In the first case, every child node must be satisfied; in the second case, even one is enough. When the root node of an attack tree is satisfied, the attack is complete.

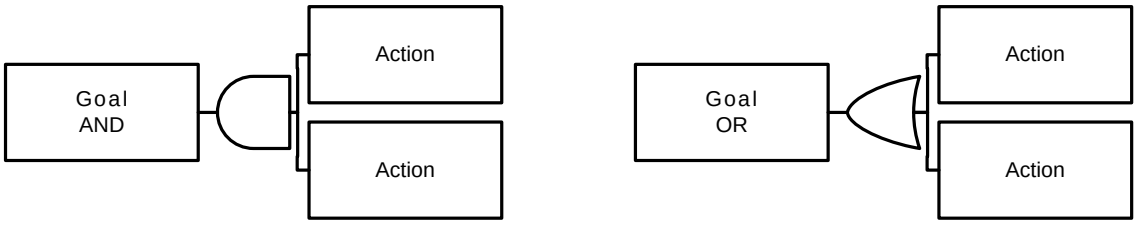

Figure 13. AND and OR example [17]

An example for attack trees can be given from the H2020 COSSIM project [17] aiming to provide an integrated CPS simulation framework. Figure 6 depicts an attack tree describing an attacker's objective to obtain confidential data sent between the COSSIM Framework elements. Figure 14 shows how Obtain user data attack tree from the COSSIM project.

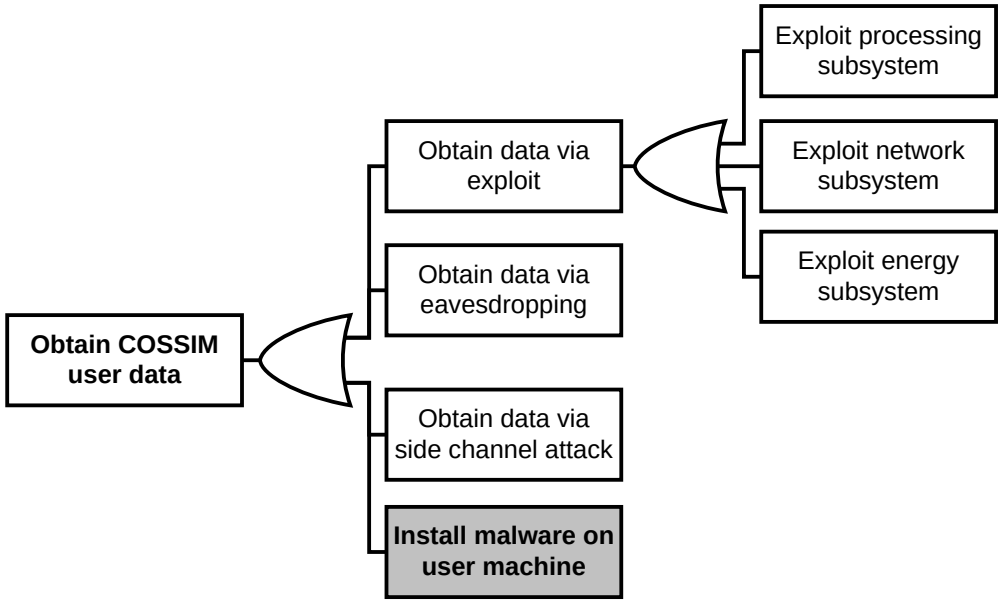

Figure 14. Obtain user data attack tree from the COSSIM project [17]

The internal and child nodes represent sufficient attacks to reach the root node representing the goal of the adversary. It suggests that an attacker may be able to:

1. obtain user data by exploiting a vulnerability in the COSSIM processing subsystem's implementation, and reading the data via direct memory access;
2. perform eavesdropping on the communication between different components, and obtain data in that way as well;
3. perform a side channel attack against the processing subsystem or energy subsystem;
4. plant malware on the COSSIM user's computer, and steal the confidential data from the user directly.

A slightly different approach following the same logic as described above is an attack tree analysis tool presented in [14] follows a top-down approach by showing all the attacks that might compromise a component. Similarly, to the attack impact analysis, it is used to represent vulnerabilities concerning safety. The same example scenario with the temperature controller is used in Figure 15 to visually present the ways of compromising the cooler, as an example.

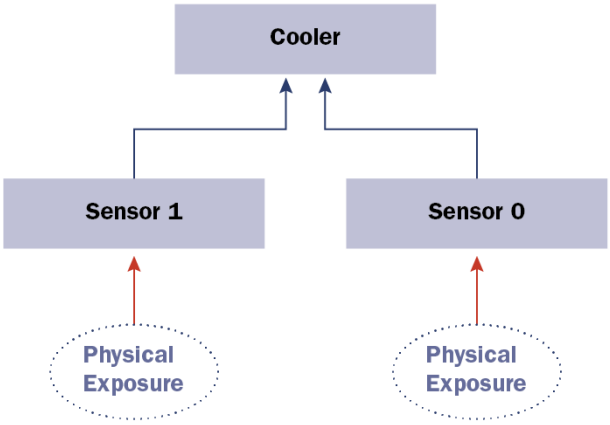

Figure 15. Ways of compromising the cooler [14]

This analysis tool includes both a standalone editor to create attack trees from scratch and an attack tree generator which can be used on a given component's attack impact analysis.
The two latter modeling tools for security analysis (attack impact, attack tree) are open source and are extensions of the Architecture Analysis & Design Language (AADL), and can be downloaded from [18].

3.2.6 Security-aware Functional modeling

This approach was proposed in [19] to model and simulate system-level attacks for the concept design of cyber-physical systems. On contrary to traditional functional models representing system functionality and data flow, security aware functional modeling expands this concept and includes cybersecurity functions, thus providing means to both analyze the effect of cyber and physical attacks on the system and to refine the design of CPS and integrate cybersecurity countermeasures (see [19] for an example). Software tools are available to model the system and its security functions and environmental conditions[3].

3.3 IoT Entity Modeling

Within CPSwarm, we consider also CPSs able to communicate with external IoT infrastructures. Telemetry data, as well as data generated by CPS payloads (e.g., thermal imaging, air quality parameters, etc.) can be monitored through the CPSwarm Monitoring and Configuration Tool. In this kind of scenario, CPS itself can be considered as an IoT sensing device.

[3] https://www.plm.automation.siemens.com/en/products/lms/imagine-lab/amesim/

In IoT domain it is important to model sensing devices, and consequently generated data, in order to guarantee syntactic and semantic interoperability with the infrastructure and the applications. The central concept around the modelled IoT sensors is that they are mainly data sources performing Observations.
One of the most used modelling approaches based on the Observations concept is the OGC Sensor Things API [26] standard. The OGC SensorThings modelling approach (see Figure 16) follows the ITU-T definition, i.e., regarding the Internet of Things, where a Thing is defined as an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks [23]. The Location entity locates the Thing or the Things it is associated with. A Thing's Location entity is defined as the last known location of the Thing. A Thing's HistoricalLocation entity set provides the current (i.e. last known) and previous locations of the Thing with their time. A Datastream groups a collection of Observations and the Observations in a Datastream measure the same ObservedProperty, and are produced by the same Sensor. A Sensor is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property. An ObservedProperty specifies the phenomenon of an Observation. An Observation is an act of measuring or otherwise determining the value of a property. An Observation results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the FeatureOfInterest of the Observation. In the context of the Internet of Things, many Observations' FeatureOfInterest can be the Location of the Thing. For example, the FeatureOfInterest of a wifi-connected thermostat can be the Location of the thermostat (i.e. the living room where the thermostat is located in). In the case of remote sensing, the FeatureOfInterest can be the geographical area or volume that is being sensed.

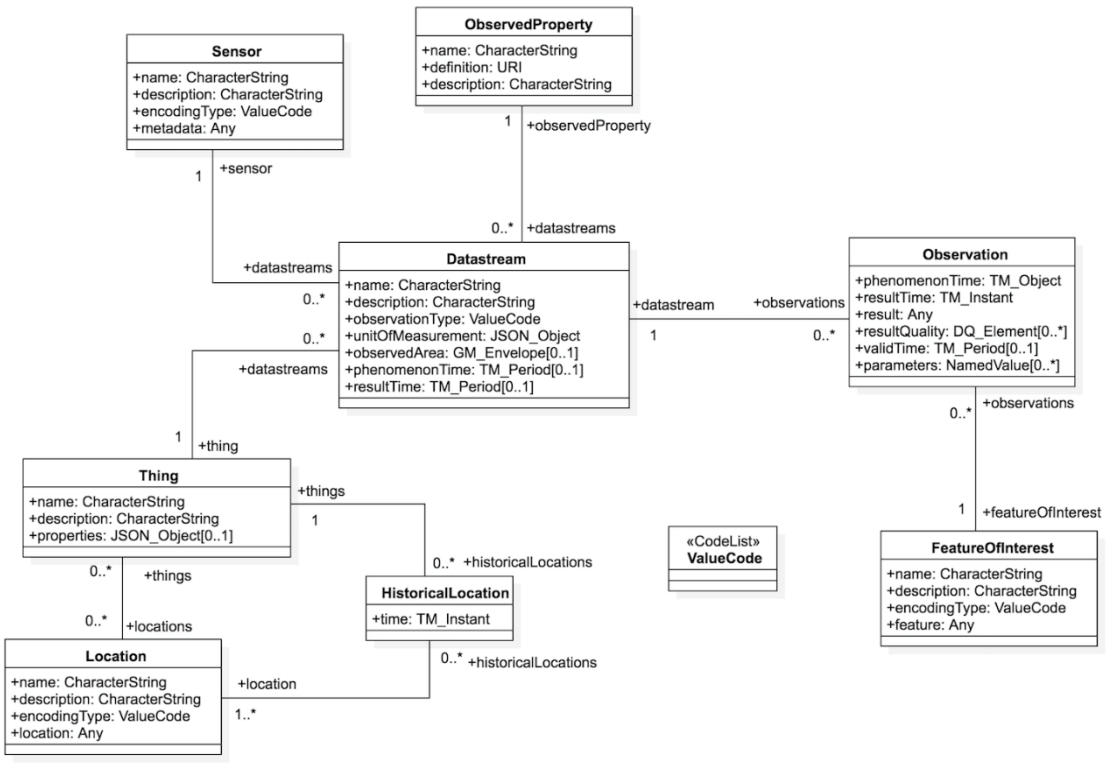

Figure 16. The OGC Sensor Things model

Several EU-funded projects, such as Almanac [20], Amazon IoT [22], FOODIE [21], etc., adopt this modelling approach.
Other than a Sensing Profile, OGC Sensor Things model defines also a Tasking profile, where the concept of Actuator is introduced, modelled according to its capabilities.

In Figure 17 some example of IoT device data modelling from [24] are shown, where different observed properties are sensed (e.g. temperature, water vapor in the air, atmospheric pressure, etc.) by relevant sensors (e.g. temperature sensor, humidity sensor, barometer, etc.).

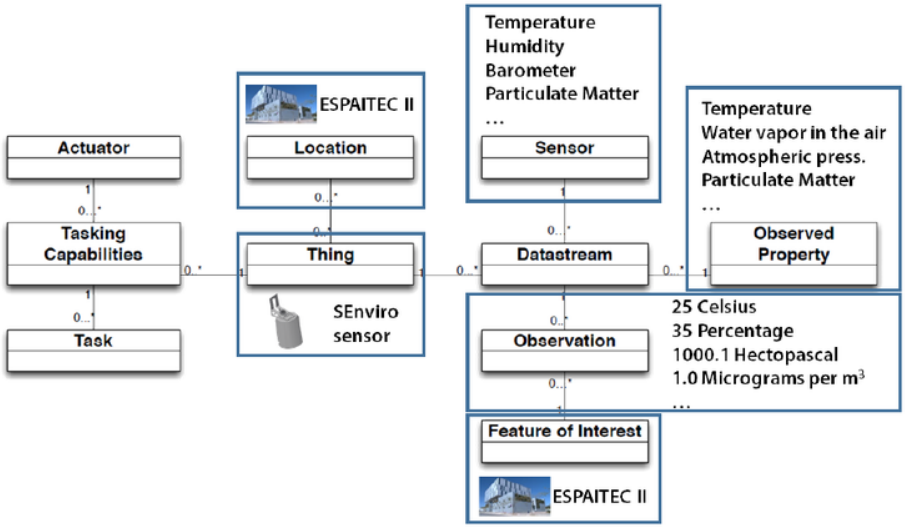

Figure 17. OGC Modelling Example

Each sensor generates a stream of data (datastream) composed by the corresponding observations (e.g. 25 celsius, 35 %, 1000.1 hPa, etc.). Sensors are embedded in a physical object, or Thing (e.g. SEnviro sensor), which could be physically deployed in a specific Location.

Other modelling approaches are based on a similar concept of Observation. Linksmart.net [25] provides a straightforward model for representing objects in the physical world and their properties, and map them to software resources that allow control and reading of data about the physical objects. The three main concepts are IoTEntity, IoTResource and IoTWorld. An IoTEntity represents a physical entity, e.g. a house, a car or a door. IoTEntity corresponds to the IoT-A [26] concept of Virtual Entity. An IoTEntity has properties: a house may have indoor temperature and energy consumption; a door may be locked or unlocked, opened or closed. IoTStateObservations represent the states of these properties at different points in time – typically generated by an IoTResource connected to a physical sensor. IoTResources are software objects that provide IoT Services for applications and end-users for retrieving and analysing data about the physical world as well as invoking actions like switching of a light. Some examples of IoTResources are software objects representing actuators, sensors, data streams, databases, etc. It could represent physical, functional or organizational units in some application domains. For instance, you can model your house as one IoTWorld, or an office, a complete building or even a whole city, it depends on the need of your application. The IoTWorld is accessed through a Linksmart.net software defined gateway.

4 Initial Libraries for CPS Models

In this Section, we make a first definition of the CPSwarm library for CPS models. The overall idea is to have a

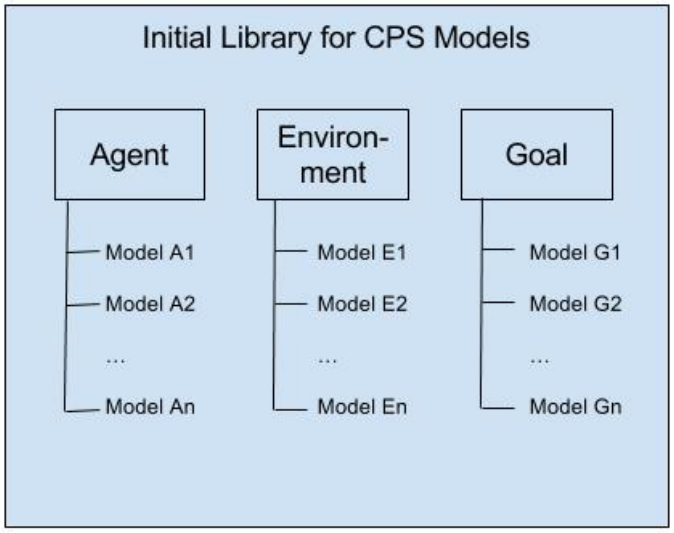

Figure 18. Structure of the Intial Library of CPS Models

library for the modeller that contains certain predefined models. These models can be reused, changed or added by the modeller. Such a library structure supports the process of modelling swarms of CPSs.

4.1 CPSWarm Libraries

The models in the libraries are separated into three groups of libraries: swarm members, environments, and goals (see Figure 18), whereby Model A1 is an exemplary model for library swarm member, Model E1 is an exemplary model for library environment and so on. In addition, sub-libraries will be possible. In the following sections, we define a generic library of CPS models. Then we provide a concrete example based on the EmergencyExit example, which serves as a prototype for the CPSwarm workbench (see Section 5).

Each model in the library (swarm member, environment, and goal) has the following mandatory parts:

1) Unique name:
   - each model needs to be distinguishable from other models by name;
   - each model's name needs to be given in a way that is associated with the model's functionality.
2) Description:
   - a detailed description is necessary for *i)* documentation and *ii)* the programming tasks by the software developer;
   - the description is created as parameter of the model with Property: string [256] (see point 3 *Parameters* for further details).
3) Parameters:
   - each model contains a set of parameters that have the following form:
     - Property: name, *type* [*range*]: Defines a constant parameter of the model;
     - Input: name, *type* [*range*]: Defines an input parameter to the model;
     - Output: name, *type* [*range*]: Defines an output parameter of the model.

4.1.1 Swarm member

Description of the library: The swarm member library describes an individual CPS used in swarm applications. Following sub-libraries are available:

1. local memory
2. behaviour
3. physical aspects
4. security (optional)
5. human interaction (optional)

All sub-libraries include several models. Local memory, behaviour and physical aspects need to be modelled mandatory. Depending on the application, also security and human interaction can be modelled. The application also defines the level of detail in which the chosen models are described.

The final swarm is modelled using multiple swarm members.

Structure of the library: All five sub-libraries (local memory, behaviour, physical aspects, security, and human interaction) have several associated models. The modeller can use these models to model an individual swarm member. Each of those models contains the property string for a textual description of the model. The modeller is able to change the a-priori defined input/output/property parameter of each model. Moreover, the modeller is able to add parameters to each model.

The swarm member's local memory expresses one (or more) local states of the swarm member. An example is the x/y position of a swarm member or the status of its locally available resources, like memory, energy, and so on. A local status is necessary for, e.g., further calculations in other models.

The swarm member behaviour describes the behaviour of the swarm member in the environment. The behaviour usually refers to collecting data from local sensors, performing calculations or sending data to the actuators. The behaviour is designed in two different levels:
High-level representation
The high-level representation shows the entire behaviour in one single model.
Low-level representation
In the low-level representation, the modeller has the possibility to customize the behaviour of a swarm member by assembling mini-behaviours taken out of a sub-library.

The physical aspects of swarm member describe its sensors and actuators that are required for functional operation. All sensors are defined with an output, and all actuators with an input parameter, each of a certain type within a certain range.

Optional security aspects in the modelling library include fault detection for physical aspects, a selection of contingency plans as part of the swarm member's behaviour and modelling remote control as a way for a human operator to take control of single swarm member or the whole swarm in case of emergency.

Human interaction models describe ways of humans interacting with the swarm of CPS. There are different roles for humans (see Section 6). Each role is modelled as an actor with a certain role description.

4.1.2 Environment

Description of the library: The environment library describes the environment in which the swarm of CPSs is acting. Several models express an environment, whereby the following ones are indispensable (further ones can be added if necessary):

- 2D/3D Map of the environment

    - occupancy grid map, i.e. free space and obstacles
    - expressed as a bitmap file
- Size of the environment
    - width and height
    - expressed in number of grid cells
- Resolution
    - expressed in number of grid cells per meter

Structure of the library: Three mandatory models describe the environment: map, size and resolution. Each of those models contains the property string for a textual description of the parameter as well as a property to define the associated values. Other parameters can be added to an environment model as necessary.

4.1.3 Goal

Description of the library: The goal library describes the goal that the swarm of CPSs wants to reach. The goal is expressed by a fitness value and a calculation specification. The calculation is done by incorporating parameters from other models. If the application asks for it, multiple fitness values can be modelled, possibly related to each other.

Structure of the library: The goal is described by a fixed model: fitness. The model contains the property string for a textual description of the model. Further, it gives an output as a single float value as result of the calculation.

5 Use Case of the Initial Catalog of CPS Models - the EmergencyExit Example

The EmergencyExit example is used as use case for the initial library of CPS models. It was defined in two steps: First, the main ideas were collected in a joint brainstorming session between LAKE and UNI-KLU. Second, the results were send to SOFTEAM to get feedback on the feasibility, especially in Modelio. The results were presented to the consortium and discussed jointly. The CPSwarm initial library on the EmergencyExit is stored on the Modelio Forge at https://forge.modelio.org/projects/cpswarm-modelio36/files as shown in Figure 19. Installation instructions for the modules to run the example are included in D5.2 Deliverable due at M9.

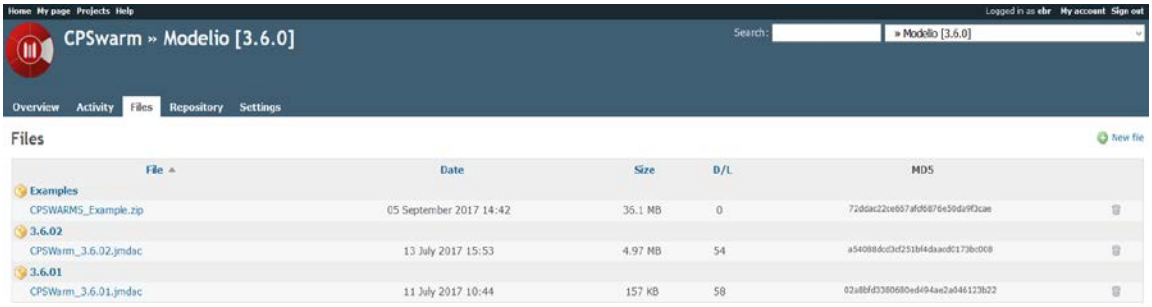

Figure 19. CPSwarm Initial Library on the Modelio Forge

Description EmergencyExit example

In the EmergencyExit example, multiple swarm members move in a 2D, discrete environment and try to find one of two emergency exits. In each discrete time step, a swarm member senses the neighbouring cells and moves to a free cell. When a swarm member reaches an emergency exit, it is removed from the environment. The goal is that all swarm members exit the environment.

The swarm member, the environment and the goal are modelled in the following subsections. The swarm member is modelled as an individual and then as a swarm member with the swarm modelling structure as represented in Figure 20.

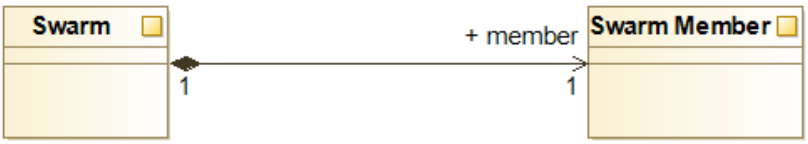

Figure 20. Swarm Architecture

5.1 Swarm member

Description of the library: The swarm member in the EmergencyExit example is a ground rover. Each rover has a local state (sub-library local memory). Further, each rover needs sensors and actuators (sub-library physical aspects) and a behaviour (sub-library behaviour).

Structure of the library: The swarm member needs a local state to express its current x/y position, two sensors and one actuator. All models are defined with an output or input parameter, respectively, of a certain type within a certain range. Furthermore, the swarm member needs a behaviour. The behaviour is modelled as high-level behaviour by using a single model describing the emergency behaviour as in Figure 21.

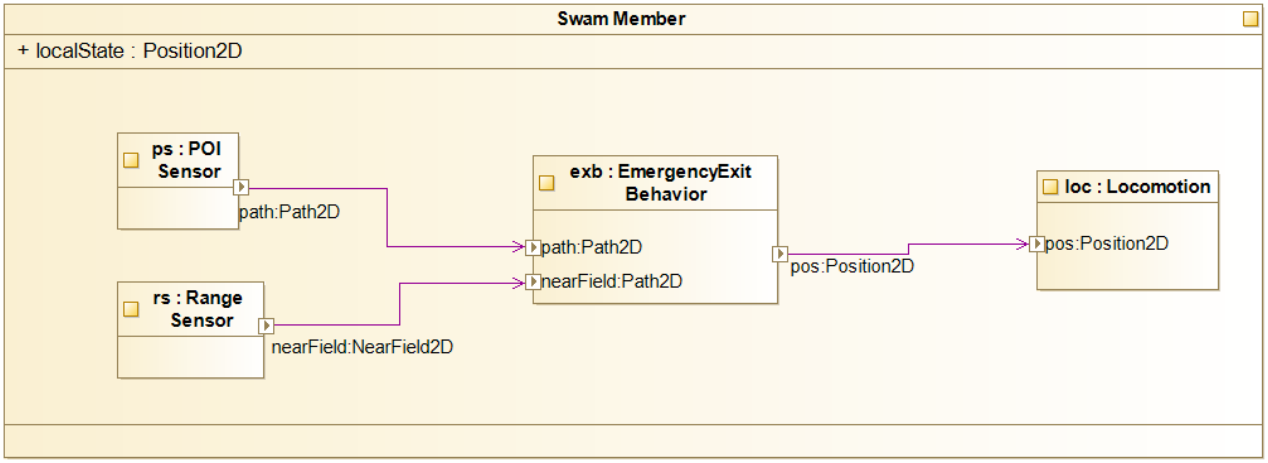

Figure 21. Model to represent a swarm member

Models of the library:

- "localState"
  - Description:" The local state describes the current x/y position of the swarm member"
  - Property name: "localState.x", type: int
  - Property name: "localState.y", type: int

- "ps: POI Sensor"
  - Description:" The POI sensor returns a vector, describing the path to the next point of interest (POI)"
  - Output: Path, type: int[2]

- "rs: Range Sensor"
  - Description:" The range sensor returns the map around the swarm member."
  - Output: NearField, type: int [8]

- "loc: Locomotion"
  - Description:" The locomotion actuator is a motor that moves the swarm member to a given x/y coordinate.""
  - Input: Position, type: int [2]

- "exb: EmergencyExit behavior"
  - Description:" In the EmergencyExit example, multiple swarm members move in a 2D, discrete environment and try to find one of two emergency exits. In each discrete time step, a swarm member senses the neighbouring cells and moves to a free cell. When a swarm member reaches an emergency exit, it is removed from the environment. The goal is that all swarm member exits the environment."
  - Input1: Path, type: int [2]
  - Input2: NearField, type: int [8]
  - Output: Position, type: float [2]

5.2 Environment

Description of the library: The library describes the environment where the swarm of CPSs is moving in (2D map, size, and resolution). A list of points of interest is added.

Structure of the library: The environment of the EmergencyExit example needs to be specified. This includes an image of the map, the resolution and the size of the map. Furthermore, we have a list of two points of interest (the location of the emergency exits), described as x/y coordinates on the map.
All models have a property to express a textual description.

Models of the library:

- "map: String"
  - Description:" The map defines a file path to the bitmap file of the environment."
  - Property: path, type: string "C:\temp\CPSwarm\environment\EmergencyExit.png"

- "res: Resolution2D"
  - Description:" The resolution defines the resolution of the map (number of grid cells per meter)."
  - Property: x, type: int
  - Property: y, type: int

- "size: Size2D"
  - Description: «The size defines the size of the map (total number of grid cells in height and width)."
  - Property: x, type: int
  - Property: y, type: int

- "poi: Position2D"
  - Description: «The POI defines two points of interest (POI) in the map with their x/y coordinates."
  - Property: poi.x., type: int
  - Property: poi.y., type: int
  -

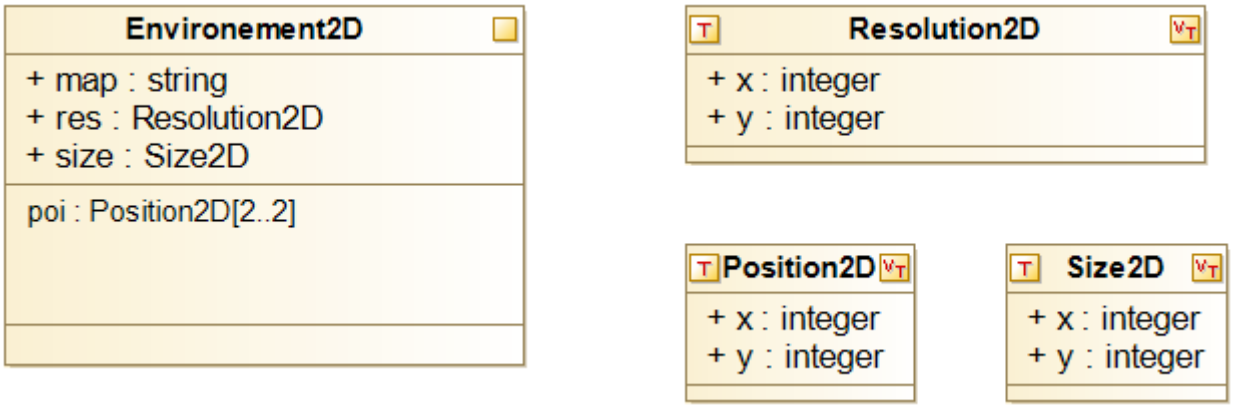

Figure 22. Model to represent the environment

5.3 Goal

Description of the library: The goal of the swarm of CPSs is to find the two points of interest - the emergency exits.
Achieving this goal is expressed by the fitness value, computed with: $-\sqrt{(POI.x - LocalState.x)^2 + (POI.y - LocalState.y)^2}$

Structure of the library: The structure of the goal in the EmergencyExit example is rather simple. A single model, the fitness, expresses it. It gives an output as a single float value.

Models of the library:

- "Fitness"
  - Description:" The fitness value is computed with $-\sqrt{(POI.x - LocalState.x)^2 + (POI.y - LocalState.y)^2}$ "
  - Input: poi, type: int []
  - Input: localState.localx, type: int
  - Input: localState.localy, type: int

Figure 23 shows a model to represent a fitness function.

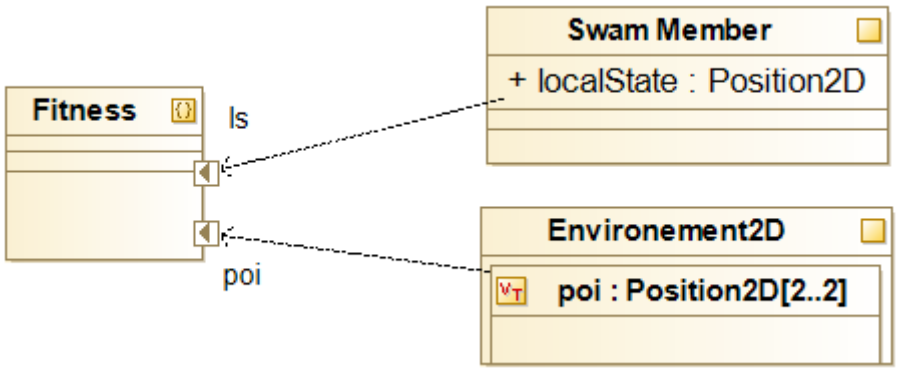

Figure 23. Model to represent a fitness function

5.4 Future Work

The security models will be elaborated in the updated versions of this deliverable. In addition, further work on threat and attack models will be included as part of the modelling library to have applicable contingency plans assigned to each of the threats and attacks addressed.

In the current implementation, the behaviour is modelled from a high-level point of view. This means that the entire behaviour is represented in a single model (Figure 21). Future work will be to model on a low-level. Therefore, we would like to use so-called mini behaviours that allow the modeller to customize the behaviour. Examples for mini behaviours of the emergency exit example could be following ones (visualized in Figure 24):

- "rsd: call read sensor data"
  - Description: string "This behaviour reads the sensor data from A2: POI Sensor and A3: Range Sensor."
  - Input1: Path, type: int [2]
  - Input2: NearField, type: int [8]

- "sc: call select cell"
    - Description: string "This behaviour uses the ANN and the data from A5 to select the next grid cell, the agent should move on."
- "m: call Move"
    - Description: string "This behaviour advises A4: Locomotion to move to a certain cell by providing x/y coordinates."
    - Output: Pos, type: float [2]

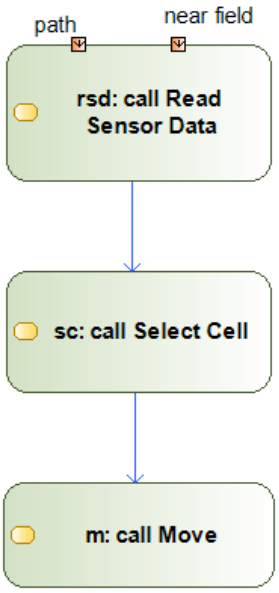

Figure 24.Customized behavior out of mini behaviors on the EmergencyExit example

6 Initial Communication among the Models

The sections above described the modelling of a single swarm member as well as the swarm problem definition. However, for a group of CPS to function as a consolidated swarm, communication between members is also necessary. Information such as a swarm member's current position, current speed or current reading of obstacle sensors could be useful to other members and it should be shared among the swarm.

This section is meant to give a first example of the modelling of a single swarm member; more details will be included in D5.1 Modeling Language deliverable due at M12.

The communication between a single swarm member and other swarm members is bi-directional. On one hand, a swarm member may need other swarm members' information to carry out its swarm algorithm. On the other hand, it also outputs necessary information to others. A swarm member may have access to many information, such as its current position, its current speed, etc., which are modelled as local memory of the swarm member (see previous section). However, not all this information is needed for communication. Therefore, one responsibility of a swarm modeller is to model the swarm members so that they expose the correct information to each other. For example, in map exploring swarm algorithm, the positions of detected obstacles are valuable, while in other cases, such information may be irrelevant.

We provide the possibility to model swarm communication in the modelling tool. For this purpose, a methodology was developed based on conventional class diagram with our own custom extension and interpretation. We use a special class *Communication Interface* to model the exposed variables of a type of swarm members. Exposed variables are modelled as public attributes in a subclass of the *Communication Interface*. By associating this subclass with a swarm member class, it indicates which type of swarm members is going to expose the specified variables. By using the "use" relation in class diagram, it indicates which members are going to use these variables.

Figure 25 shows an example of modelling the communication between rovers and drones in a swarm. The classes *Rover Communication Interface* and *Drone Communication Interface* are extension of class *Communication Interface* and they are used to describe the exposed information of rovers and drones respectively.

In this example, rovers have access to three pieces of information: their own current position (*pos*), current speed (*speed*) and the reading of the laser sensor (*laser_sensor*). Drones have access to their current position (*pos*), current height (*height*) and current speed (*speed*).

On one hand, two variables from rovers (speed and *laser_sensor*) are chosen to be exposed to other members. This information will be used not only by drones, but also by other rovers, to carry out their swarm algorithm, just as indicated by the two dashed line arrows. On the other hand, two variables from drones (*pos* and *height*) are also exposed and they will be utilized by rovers.

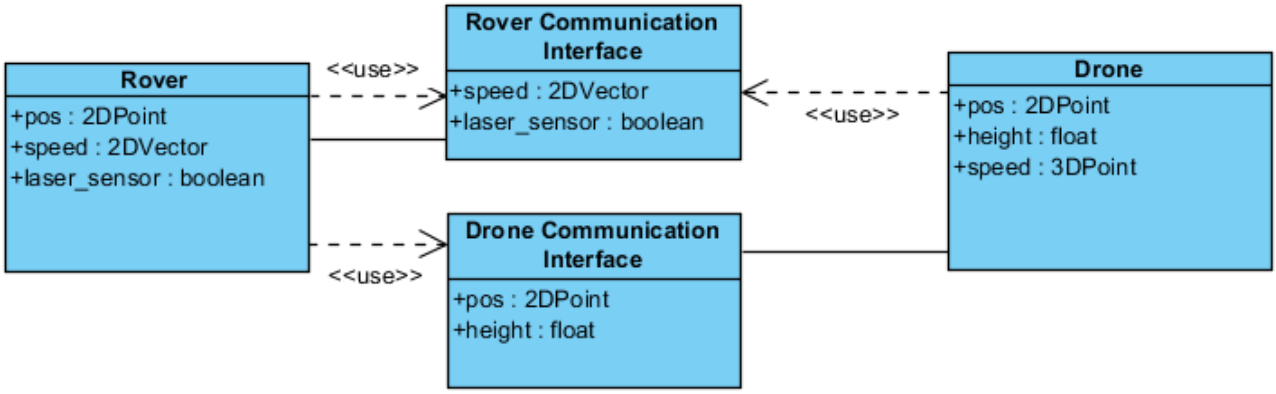

Figure 25. Example diagram for swarm communication modelling

By using this method, swarm modellers/designers have an intuitive and yet highly flexible tool to model communication among the swarm. If a type of swarm member need to exposed different variables to different kinds of swarm members, the user just simply needs to build a separate Communication Interface for each group of swarm members (see Figure 26, *Rover* is providing different interfaces to *Drone* and *Rover 2*). Besides, this method could also be applied to configure what variables to monitor in the monitoring tool by simply connecting the *Monitoring Tool* class to interesting different interfaces (see Figure 26).

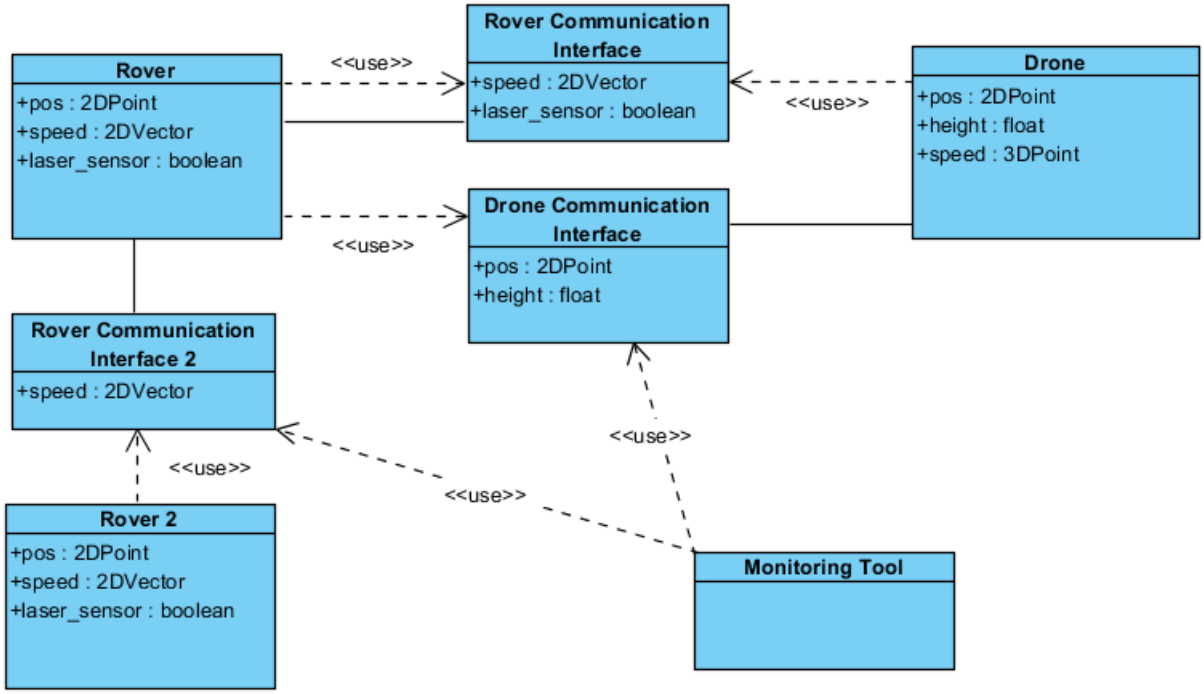

Figure 26. Extended example diagram for swarm communication modelling

7 Human-in-the-Loop

7.1 Roles of Humans in the Design of CPS Swarms in the CPSwarm Workbench

This section outlines human roles in the CPSwarm workbench. The main structure of roles and actors is adapted from D2.1- Initial vision scenarios and use case definition (see Figure 27) and the publication of Schranz et. al in [29] (see Figure 28).

The summarized roles of the approaches from D2.1 and [29], lead to following construction, that will be used as an initial concept to model the human in the CPSwarm workbench: This concept comes with four roles namely the modeller, the software developer, the engineer and the operator, which abstracts several sub-roles from D2.1 (see Figure 28).The customer is skipped in this consideration, as it plays a minor role in working with the CPSwarm workbench. Each role has pre-defined tasks that are further explained in the following subsections.

CPSwarm workbench roles – modeller, software developer, engineer, and operator – are modelled as actors (see Figure 28). Feedback loops between roles are always possible and desirable. Security issues like identifying malicious sensory data, malicious swarm agents, swarm behaviour in case of malicious swarm member, security breach potentials and contingency plans will be considered in the deliverables D4.7 and D4.8 that are related to security threat and attack models.

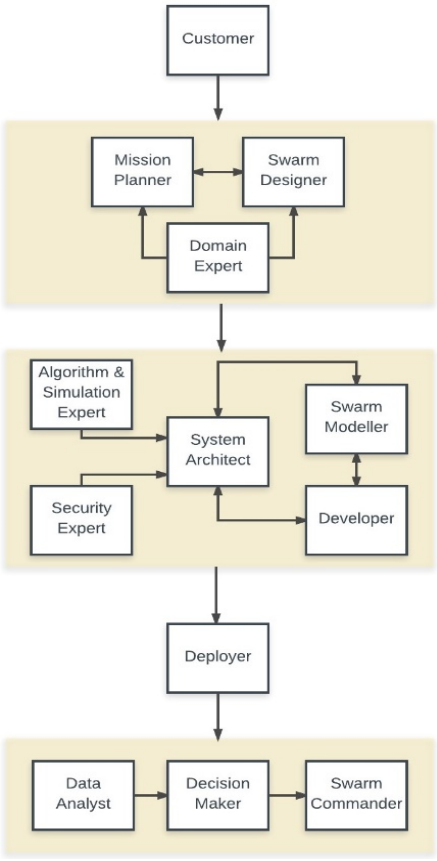

Figure 27. Communication flow between stakeholders (D2.1)

7.1.1 Modeller

The modeller is responsible for analysing and modelling a problem definition and all its associated

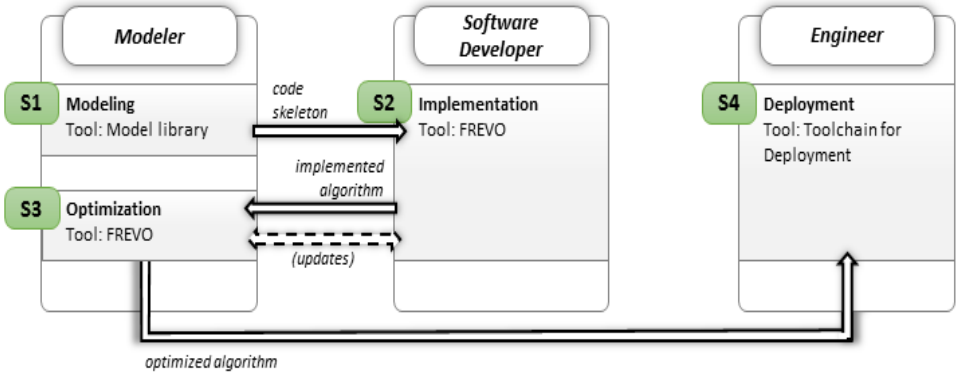

Figure 28. Three roles with their individual tasks in the CPS design process [29]

parameters with a computer modelling tool. She/he has a rapid perception for complex problems and a basic understanding of programming.

The tasks of a modeller comprise:

- Define a problem
- Formulate an overall approach to solve the problem
- Provide description and model of the libraries swarm member, the environment, and the goal (see Section 4)
- Define and model swarm structure and behaviour
- Formulate fitness function

Modelling swarm members, environment, and goal already includes the awareness of the modeller to the physical available resources (type, quality, capabilities, limitations, and security issues of the CPS). Furthermore, the modeller has basic knowledge about swarms in order to be able to specify a swarm structure and behaviour.

7.1.2 Software Developer

The software developer implements functionalities of the models provided by the modeller. Therefore, she/he writes code in a high-level programming language. Her/his characteristic is an analytic and structured operation.
The tasks of a software developer comprise:

- Implement functionalities as modelled by the modeller
- Create new models (out of the implemented functionalities) for the CPSwarm library
- Implement the fitness function (if not yet done by the modeller)

7.1.3 Engineer

The engineer has the task of deploying the final algorithms on hardware. She/he understands specific and detailed hardware characteristics as well as the deployment tool chain to bring code to hardware. Her/his characteristic is a structured working ability.
The tasks of an engineer comprise:

- Make use of the deployment tool chain in order to put the code onto the target systems
- Adjust parameters for deployment, including possible setup of communication channels (defining addresses and network names), calibrating sensors and testing the target system.

7.1.4 Operator

The operator is responsible for operating the swarm as deployed by the engineer. She/he is responsible for successfully controlling the swarm, managing communication and interactions between humans and the swarm, and retrieving information from the swarm. She/he interprets the data acquired by the swarm to make decisions and manipulate the swarm.
The tasks of an operator comprise:

- Monitor the swarm in real time
- Acquire, interrupt and analyse data about the swarm
- Interacting with the swarm by setting initial mission goals and adjusting the swarm behaviour during operation
- Assessing the status of the swarm and observing the environment based on information from the swarm members.

7.2 Future Work on Human-Swarm Interaction

In future work on human-swarm interaction models in the CPSwarm modelling library we will first focus on the perspective of the operator (see Section 6.1). The operator interacts with the swarm of CPS through a communication interface as in Figure 29 to

- get outputs from the swarm via state estimation and visualization
- send inputs to the swarm via control methods.

Interacting with a swarm is related communication to a single entity (the entire swarm) rather than to multiple individual robots. Topics we are going to deal with cover following points of interactions with a swarm:

- Monitoring
- Controlling
    - Setting mission goals
    - Adjusting behaviour
    - Interrupt behaviour
- Analysing data
    - Assess information provided by the swarm
    - Observing the environment based on the information provided by the swarm.

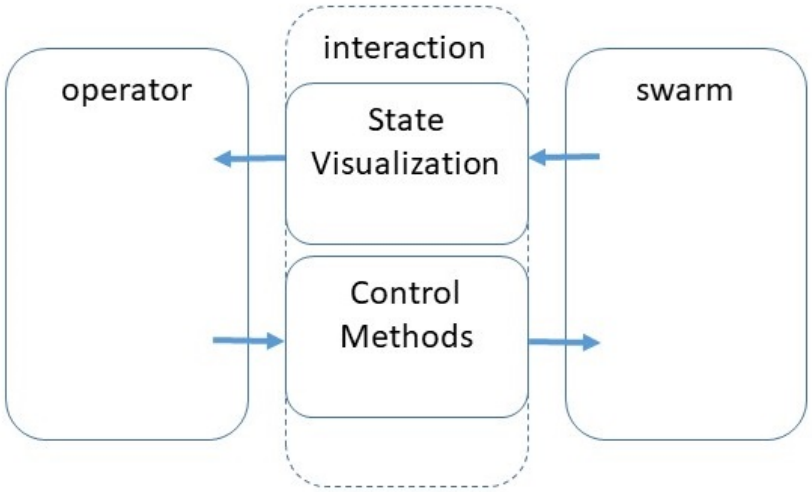

Figure 29. Human-Swarm Interaction Model (adapted from [9])

Moreover, the CPSwarm workbench is going to offer a library with further instructions and guidelines to specify a swarm's behaviour regarding human encounter. This relation becomes a topic in every vision scenario as defined in D2.1 for safe, smooth and effective collaboration between humans and swarms whereas this project regards a swarm in a supporting role for the human. The guidelines should cover safety aspects and task-related activities that need to be considered when implementing a swarm mission.
Furthermore, at the current state of concept development, WP2 and WP4 envision an extensible set of "behavioural parameters" that adjust characteristics of a swarm's behaviour. These aspects could be, e.g., defensive behavioural vs. demanding (like guiding and insisting on certain reactions of human in a certain context. Another example could be the configuration of the degree of autonomous behaviour vs. step-wise human control. This way of interaction depends on the mission to fulfil and the involvement of humans within a setting; a swarm assisting in certain tasks should preferably wait for commands and react efficiently.

On the other hand, in search and rescue missions, swarm members should organize themselves and search the environment in an efficient and effective way.
Further parameters need to be found and ranges need to be set in the future as well as mutual influences of parameter values.
The current idea is that parameters are given and adjusted by the modeller in the swarm model. The adjustment is performed on a pre-defined scale of float numbers between 0 and 1 that enables the modeller to decide the amount of influence in the final human-swarm interacting procedure. These parameters include, e.g., cognitive complexity, interaction, autonomy, environmental influence, and safety.

More details on the main ideas are part of the internal work-in-progress paper. Further details together with the implementation will be part in D4.2 – Updated CPS modelling library (M21) after updating requirements and further detailing the scenarios derived within WP2.

8 Conclusions

The deliverable D4.1 – Initial CPS modelling library provides a good overview on existing models in CPS design. Furthermore, it states the initial communication among the models. The most important part for the future work of the CPS modelling library is given in Section 4 with the initial catalogue for CPS models. Therein, formalizations, definitions and implementations were done for the three main model libraries: swarm members, environment and goal. The implementation focuses on the Emergency Exit example that offered a quite good understanding of final modelling issues in modelling swarms of CPS.
Finally, Section 7 gives a first idea for human-swarm interaction. The future concept for human-swarm interaction as part of the CPSwarm modelling library was already mentioned in Section 7.2.
In the current state of concept derivation, the swarm operator is focused to control the swarm and its behaviour. Future ideas extend this thought to specifying parameters regarding a swarm's behavioural characteristics when operating besides humans as assistive technology (cf. SWARM Logistics Assistant Scenario in D2.1), search and rescues technology (cf. Swarm Drones Scenario D2.1). Eventually, humans just need safe and unobtrusive swarms like in car convoy where safety, effectiveness and efficiency are the main goals for which an optimal problem solving strategy is needed (cf. Automotive CPS in D2.1). More details about the concept itself and its implementation will be addressed by the upcoming D4.2 – Updated CPS modelling library, scheduled for M21.

The initial catalogue for CPS models in Section 4 and the corresponding use case implementation in Section 5 are of main interest to deliverable D4.4 due M10. Therein, the initial swarm intelligence algorithm library will be created strictly keeping on these definitions and formalizations in this deliverable. The models for these algorithms will follow the same concept as the models for the behaviour described in Sub-Section 4.1.1: first, a high-level view is considered, where a model defines a swarm algorithm. Second, the swarm algorithms will be divided into mini-behaviours. This allows the modeller to customize the behaviour to create its own swarm algorithm.

Acronyms

Acronym	Explanation
DoA	Description of Action
DSE	*Design Space Exploration*
KPI	Key Performance Indicators
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
MBSE	Model-based systems engineering
POI	Points of interest
CPS	Cyber-Physical Systems
UAV	Unmanned Aerial Vehicles

List of figures

9 References

[1] FMI Standard. FMI 2.0. 2016. Available at: https://www.fmi-standard.org/

[2] 20-Sim. 2016. Available at: http://www.20sim.com/

[3] Softeam. Modelio Open-Source Modeling Environment. 2016. Available at: http://www.modelio.org/.

[4] The project deliverables are available at: http://into-cps.au.dk/publications/

[5] INTO-CPS GitHub page, the project examples are available at: https://github.com/into-cps

[6] Object Management Group (OMG). System modeling language specification v1.5. 2017. Available: http://www.omg.org/spec/SysML/1.5/

[7] System Modelling with Modelio http://into-cps.github.io/systemmodelling/

[8] INTO-CPS Project in project in Modelio.org forge http://forge.modelio.org/projects/intocps

[9] A. Kolling, P. Walker, N. Chakraborty, K. Sycara and M. Lewis, "Human Interaction With Robot Swarms: A Survey," in IEEE Transactions on Human-Machine Systems, vol. 46, no. 1, pp. 9-26, Feb. 2016.

[10] E. Haas, M. Fields, S. Hill, and C. Stachowiak, "Extremee scalability: Designing interfaces and algorithms for soldier-robotic swarm interaction," DTIC Document, Tech. Rep., 2009.

[11] Seyed Behzad Tabibian, Michael Lewis, Christian Lebiere, Nilanjan Chakraborty, Katia Sycara, Stefano Bennati, Meeko Oishi, "Towards a Cognitively-Based Analytic Model of Human Control of Swarms", in Proc. AAAAI Spring Symp. Series, 2014.

[12] Sasanka Nagavalli, Shih-Yi Chien, Michael Lewis, Nilanjan Chakraborty, and Katia Sycara. 2015. Bounds of Neglect Benevolence in Input Timing for Human Interaction with Robotic Swarms. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*(HRI '15). ACM, New York, NY, USA, 197-204. DOI=http://dx.doi.org/10.1145/2696454.2696470

[13] Lewis, M., Wang, J., & Scerri, P. (2006). Teamwork coordination for realistically complex multi robot systems. In NATO Symposium on Human Factors of Uninhabited Military Vehicles as Force Multipliers (pp. 1-12)

[14] Security Modeling Tools https://insights.sei.cmu.edu/sei_blog/2016/10/security-modeling-tools.html

[15] Security Application of Failure Mode and Effect Analysis, Christoph Schmittner et al.

[16] An Introduction to Fault Tree Analysis, Dr. Jane Marshall

[17] The COSSIM H2020 project http://www.cossim.org

[18] A set of modeling tools for security analysis (attack tree, attack impact) and a code generator to produce code for the seL4 platform from AADL models https://github.com/cmu-sei/AASPE

[19] Wan, Jiang, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. "Security-aware functional modeling of cyber-physical systems." Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on. IEEE, 2015.

[20] The Almanac project, http://www.almanac-project.eu

[21] The FOODIE project, http://www.foodie-project.eu/
[22] Amazon AWS IoT, https://aws.amazon.com/
[23] ITU-T-Y.2060, Overview of the Internet of Things, 2012.
[24] Trilles Oliver, Sergi, et al. "SEnviro: A Sensorized Platform Proposal Using Open Hardware and Open Standards." (2015).
[25] LinkSmart.net – IoTWorldServices, http://www.iotworldservices.com/
[26] Introduction to the Architectural Reference Model for the Internet of Things, http://iotforum.org/wp-content/uploads/2014/09/120613-IoT-A-ARM-Book-Introduction-v7.pdf

[27] J. Nagi, A. Giusti, L. M. Gambardella and G. A. Di Caro, "Human-swarm interaction using spatial gestures," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, 2014, pp. 3834-3841.

[28] B. Gromov, L. M. Gambardella and G. A. Di Caro, "Wearable multi-modal interface for human multi-robot interaction," 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, 2016, pp. 240-245.

[29] INTO-CPS Project http://projects.au.dk/into-cps/

[30] M. Schranz, W. Elmenreich and M. Rappaport, „Designing Cyber-Physical Systems with Evolutionary Algorithms", book chapter, Springer, to appear in 2018.
[31] http://www.opengeospatial.org/standards/sensorthings
[32] Alessandra Bagnato, Regina Krisztina Bíró, Dario Bonino, Claudio Pastrone, Wilfried Elmenreich, René Reiners, Melanie Schranz, Edin Arnautovic: Designing Swarms of Cyber-Physical Systems: the H2020 CPSwarm Project: Invited Paper. Conf. Computing Frontiers 2017: 305-312