D5.2 - CPSWARM MODELLING TOOL

Deliverable ID	**D5.2**
Deliverable Title	**CPSwarm Modelling Tool**
Work Package	**WP5 – CPSwarm Design Workbench**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**09-10-2017**
Status	**Final**
Lead Editor	**Alessandra Bagnato (SOFTEAM)**
Main Contributors	**Melanie Schranz (LAKE), Etienne Brosse (SOFTEAM)**

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	2017-07-11	Alessandra Bagnato (SOFTEAM)	First Draft
0.2	2017-07-19	Alessandra Bagnato (SOFTEAM)	Second Draft
0.3	2017-8-18	Melanie Schranz (LAKE)	Content for two chapters: OPTIMIZATION TOOL integration, External Simulator integration
0.4	2017-9-22	Etienne Brosse (SOFTEAM)	Content for chapters CPS Modelling and CPS Modelling Tool Architecture
0.5	2017-10-03	Etienne Brosse (SOFTEAM)	Integration of internal review comments
0.6	2017-10-09	Alessandra Bagnato (SOFTEAM)	Updates
1.0	2017-10-09	Alessandra Bagnato (SOFTEAM)	Ready to be submitted to EC

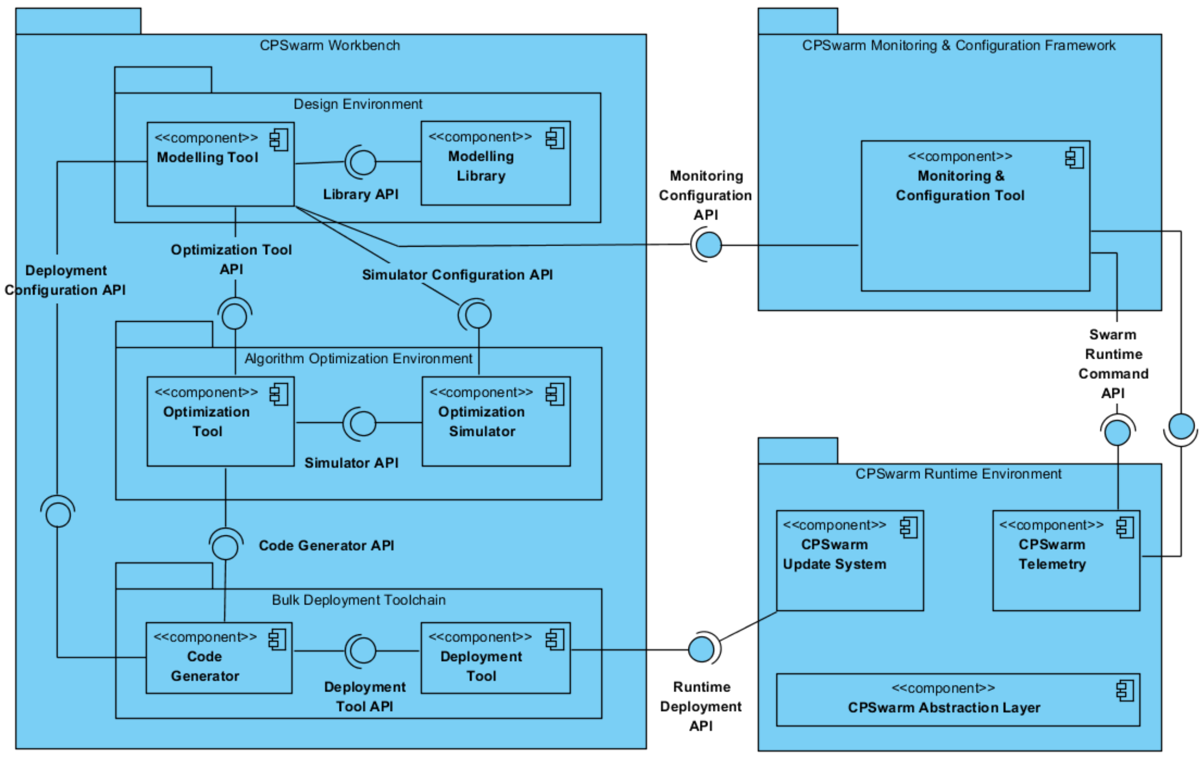
Contents

1 Executive summary

This deliverable, namely "**D5.2 - CPSwarm Modelling Tool**", introduces the implementation of the CPSwarm workbench. This includes the CPSwarm design environment – that supports the Design Environment Language which will be defined in the D5.1 - the Optimization algorithm environment – that optimizes the CPSwarm design – and their interfaces. These environments provide dedicated interfaces for end-users based respectively on SOFTEAM's Modelio and UNI-KLU's Optimization Tool tools. By exploiting the model libraries defined in WP4, the environments enable the design of both single CPS, focus of this task 5.2, and populations of CPS.

2 Introduction

As described in CPSwarm deliverable D3.1 - Initial System Architecture & Design Specification, delivered at M6 - the CPSwarm architecture adopts a component-based definition where each part or modules, named "components", of the system provided a set of well-defined functionalities as shown in Figure 1.



CPSwarm Workbench
Design Environment
<<component>> Modelling Tool
<<component>> Modelling Library
Library API
Optimization Tool API
Simulator Configuration API
Deployment Configuration API
Algorithm Optimization Environment
<<component>> Optimization Tool
<<component>> Optimization Simulator
Simulator API
Code Generator API
Bulk Deployment Toolchain
<<component>> Code Generator
<<component>> Deployment Tool
Deployment Tool API
Monitoring Configuration API
CPSwarm Monitoring & Configuration Framework
<<component>> Monitoring & Configuration Tool
Swarm Runtime Command API
Swarm Real-time Data API
Runtime Deployment API
CPSwarm Runtime Environment
<<component>> CPSwarm Update System
<<component>> CPSwarm Telemetry
<<component>> CPSwarm Abstraction Layer

Figure 1 - Overview of the Software Components in CPSwarm (see D3.1 for more information)

This "**D5.2 - CPSwarm Modelling Tool**" is a public deliverable focused on the Modelling Tool implementation on CPSwarm M9. It details the M9 status of Modelling Tool component and its implemented interfaces with related components.
Softeam, as deliverable leader, initially drafted the document, which has subsequently been enriched by all partners' contributions describing their developments.

2.1 Scope

This deliverable describes the M9 implementation of CPSwarm modelling tool and its connections to other CPSwarm component. For each component, a quick description is made, but the deliverable focuses more on their installation, graphical interface and their usages.
Softeam led the deliverable and contributed for the Design environment part including the Modelling Tool. UNI-KLU contributed by integrating the description of the Optimization Tool part.

2.2 Document organization

The remainder of this deliverable is organized as follows:
Section 3 describes the Modelling Tool by giving more details of its architecture and its Graphical User Interface (GUI). Section 4 describes the Optimization API exiting between the Modelling Tool and the Optimization Tool (this latter is more detailed in Deliverable D6.1). Finally, Section 5 concludes on the first implementation of CPSwarm Modelling Tool.

2.3 Related documents

ID	Title	Reference	Version	Date
[D3.1]	Initial System Architecture & Design Specification	D3.1	1.0	30/06/2017
[D4.1]	Initial CPS Modelling Library	D4.1	1.0	30/09/2017
[D6.1]	Initial Simulation Environment	D6.1	1.0	30/09/2017

3 Modelling Tool

3.1 Overview

Inside the CPSwarm architecture, a Design Environment has been specified. This environment is responsible of providing the CPS swarm architecture modelling functionality and is composed of two sub components - respectively named Modelling Tool and Modelling Library - strongly linked. The Modelling Tool also uses the Optimization Tool API to communicate with the Optimization Tool - part of the Algorithm Optimisation Environment – as shown in Figure 2.

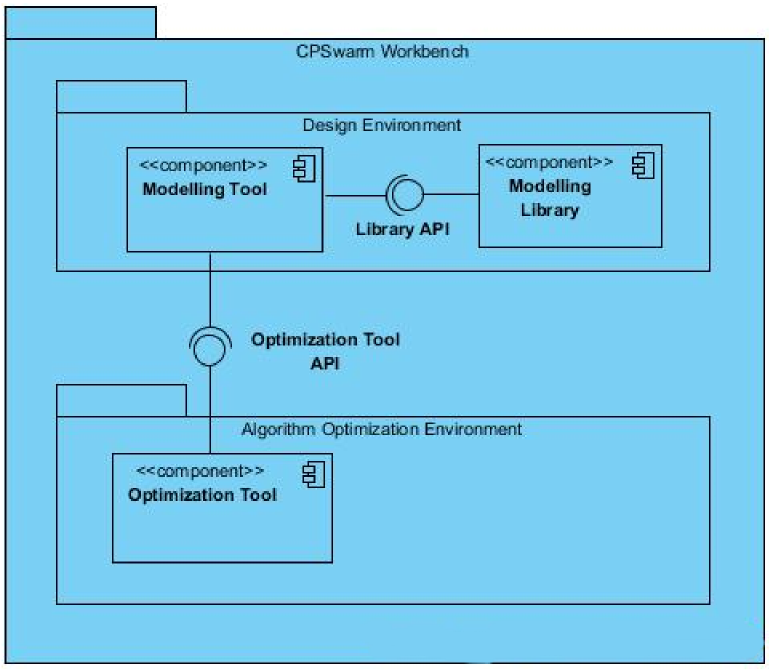


Figure 2 – Design Environment Architecture

The following section depicts the Architecture and Graphical User Interface of the Modelling Tool.

3.2 Architecture

3.2.1 Overview

The CPSwarm Modelling Tool is built on top of the open source modelling environment (UML2, BPMN2, MARTE and SysML among others) named Modelio. Modelio Architecture is built around a central repository, around which a set of extension are defined cf. Figure 3.

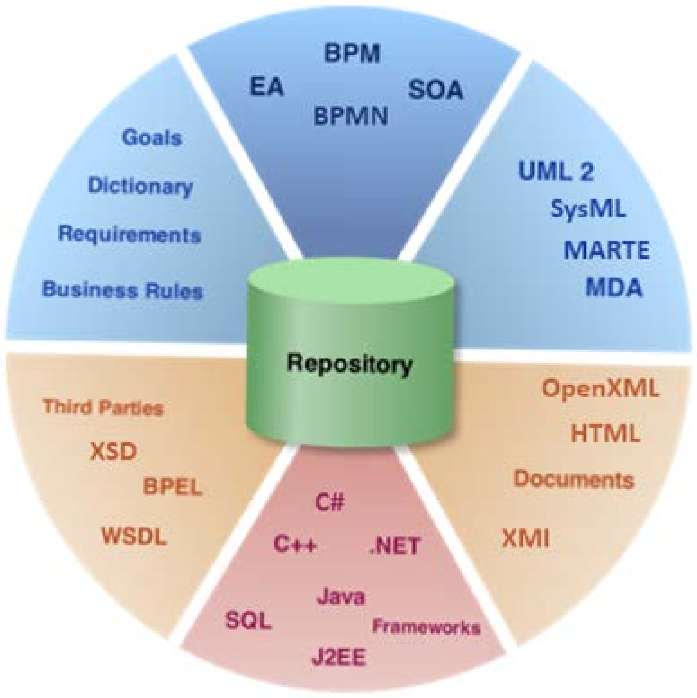


Figure 3 – Modelio Architecture

Each extension provides some specific facilities, which can be classified in the following categories:

- Scoping: this category is composed of Goals, Dictionary & Business Rules and Requirement Analyst which allow specifying high-level business models for any IT system;
- Modelling: for example, SysML, MARTE and BPMN are included in this category. The extensions belonging to this category are used to model different specific aspects of a system such as Business Process, component architectures, SOA or embedded systems;
- Code generators: such as C++ or JAVA. These extensions allow users to generate and to reverse the code to/from different programming languages;
- Utilities: modules allowing transversal utility facilities like teamwork or Import/export functionalities.

This architecture allows Modelio modelling environment to be flexible and configurable simply by adding the desired extension and related functionalities.

As depicted in Figure 4, the CPSWarm Modelling tool is composed of Modelio itself, a dedicated CPSwarm extension to provide the functionalities related to CPS swarm design, and also a set of pre-existing extensions to reuses their relevant functionalities in the CPSwarm context. At M9 of CPSwarm, only the SysML extension has been pick for its functionalities related to System modelling.

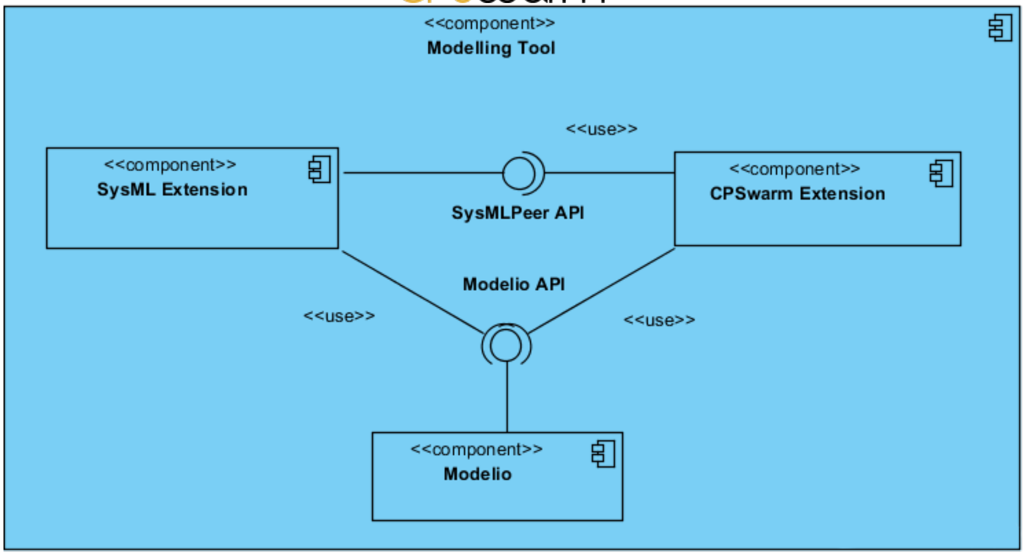


Figure 4 – Modelling Tool Architecture

Extension has mean to extend the underneath Modelling environment. This extension can be done in several ways including:

- the metamodel by specifying stereotypes,
- the menus by adding commands (or buttons),
- the behaviour by listening specific events.

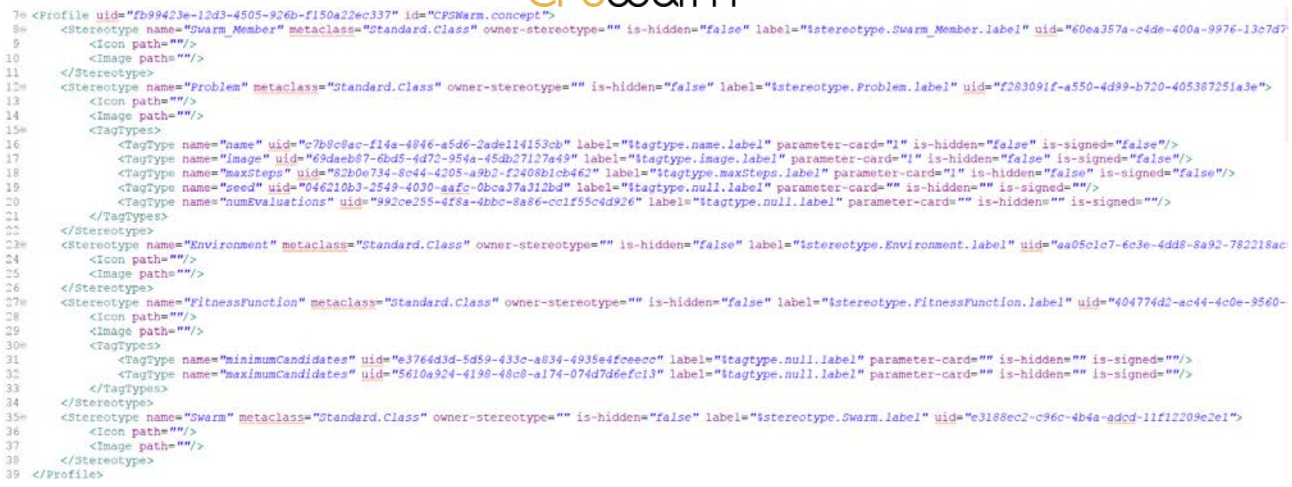
All the extension points are currently used by the CPS swarm extension and are detailed in the following sections.

3.2.2 CPSwarm Stereotypes

CPSwarm Modelling tool provides modelling environment oriented for CPS swarm designing. For this purpose, it reuses a sub set of existing concepts defined in other language as UML – in Modelio - or SysML - in the SysML extension - and also defines a set of dedicated annotations which are extensions of the Modelio metamodel.
These extensions – named UML Stereotypes - are defined and structured in UML profiles. These stereotypes are declared in module.xml file embedded inside the CPSwarm extension. To define them, a stereotype must have the following attributes:

- **uid** value is a unique identifier. It is very important that the value shouldn't be copied from another object (don't copy the id from the example).
- **name** value is the name of the stereotype.
- **label** value is the name of the stereotype that will appear in GUI and diagrams.
- **is-hidden** value indicates whether or not the stereotype should be displayed in the GUI when manually setting/unsetting stereotypes.

The Figure 5 depicts the list of CPSwam stereotypes.



```xml
<Profile uid="fb99423e-12d3-4505-926b-f150a22ec337" id="CPSWarm.concept">
    <Stereotype name="Swarm_Member" metaclass="Standard.Class" owner-stereotype="" is-hidden="false" label="$stereotype.Swarm_Member.label" uid="60ea357a-c4de-400a-9976-13c7d7
        <Icon path=""/>
        <Image path=""/>
    </Stereotype>
    <Stereotype name="Problem" metaclass="Standard.Class" owner-stereotype="" is-hidden="false" label="$stereotype.Problem.label" uid="f283091f-a550-4d99-b720-405387251a3e">
        <Icon path=""/>
        <Image path=""/>
        <TagTypes>
            <TagType name="name" uid="c7b8c8ac-f14a-4846-a5d6-2ade114153cb" label="$tagtype.name.label" parameter-card="1" is-hidden="false" is-signed="false"/>
            <TagType name="image" uid="69daeb87-6bd5-4d72-954a-45db27127a49" label="$tagtype.image.label" parameter-card="1" is-hidden="false" is-signed="false"/>
            <TagType name="maxSteps" uid="82b0e734-8c44-4205-a9b2-f2408b1cb462" label="$tagtype.maxSteps.label" parameter-card="1" is-hidden="false" is-signed="false"/>
            <TagType name="seed" uid="046210b3-2549-4030-aafc-0bca37a312bd" label="$tagtype.null.label" parameter-card="" is-hidden="" is-signed=""/>
            <TagType name="numEvaluations" uid="992ce255-4f8a-4bbc-8a86-cc1f55c4d926" label="$tagtype.null.label" parameter-card="" is-hidden="" is-signed=""/>
        </TagTypes>
    </Stereotype>
    <Stereotype name="Environment" metaclass="Standard.Class" owner-stereotype="" is-hidden="false" label="$stereotype.Environment.label" uid="aa05c1c7-6c3e-4dd8-8a92-782218ac
        <Icon path=""/>
        <Image path=""/>
    </Stereotype>
    <Stereotype name="FitnessFunction" metaclass="Standard.Class" owner-stereotype="" is-hidden="false" label="$stereotype.FitnessFunction.label" uid="404774d2-ac44-4c0e-9560-
        <Icon path=""/>
        <Image path=""/>
        <TagTypes>
            <TagType name="minimumCandidates" uid="e3764d3d-5d59-433c-a834-4935e4fceecc" label="$tagtype.null.label" parameter-card="" is-hidden="" is-signed=""/>
            <TagType name="maximumCandidates" uid="5610a924-4198-48c8-a174-074d7d6efc13" label="$tagtype.null.label" parameter-card="" is-hidden="" is-signed=""/>
        </TagTypes>
    </Stereotype>
    <Stereotype name="Swarm" metaclass="Standard.Class" owner-stereotype="" is-hidden="false" label="$stereotype.Swarm.label" uid="e3188ec2-c96c-4b4a-adcd-11f12209e2e1">
        <Icon path=""/>
        <Image path=""/>
    </Stereotype>
</Profile>
```

Figure 5 – CPSwarm Stereotypes declaration

All details of the CPSwarm stereotypes and their meaning will be part of D5.1 CPSwarm Deliverable - CPSwarm Modelling Language Specification - due by M12 of the project.

3.2.3 CPSwarm Commands

The CPS swarm Modeller - as defined in Section 7 of D4.1 - can interact with the design environment through commands that can be specific to the CPS warm extension.
There are several different kinds of commands:

- Menu commands,
- Toolbar button commands,
- Diagram toolbar button commands,
- Diagram palette button commands,
- Contextual menu commands,
- Property view toolbar button commands.

The Figure 6 illustrates the different kinds of command possible on the design environment.

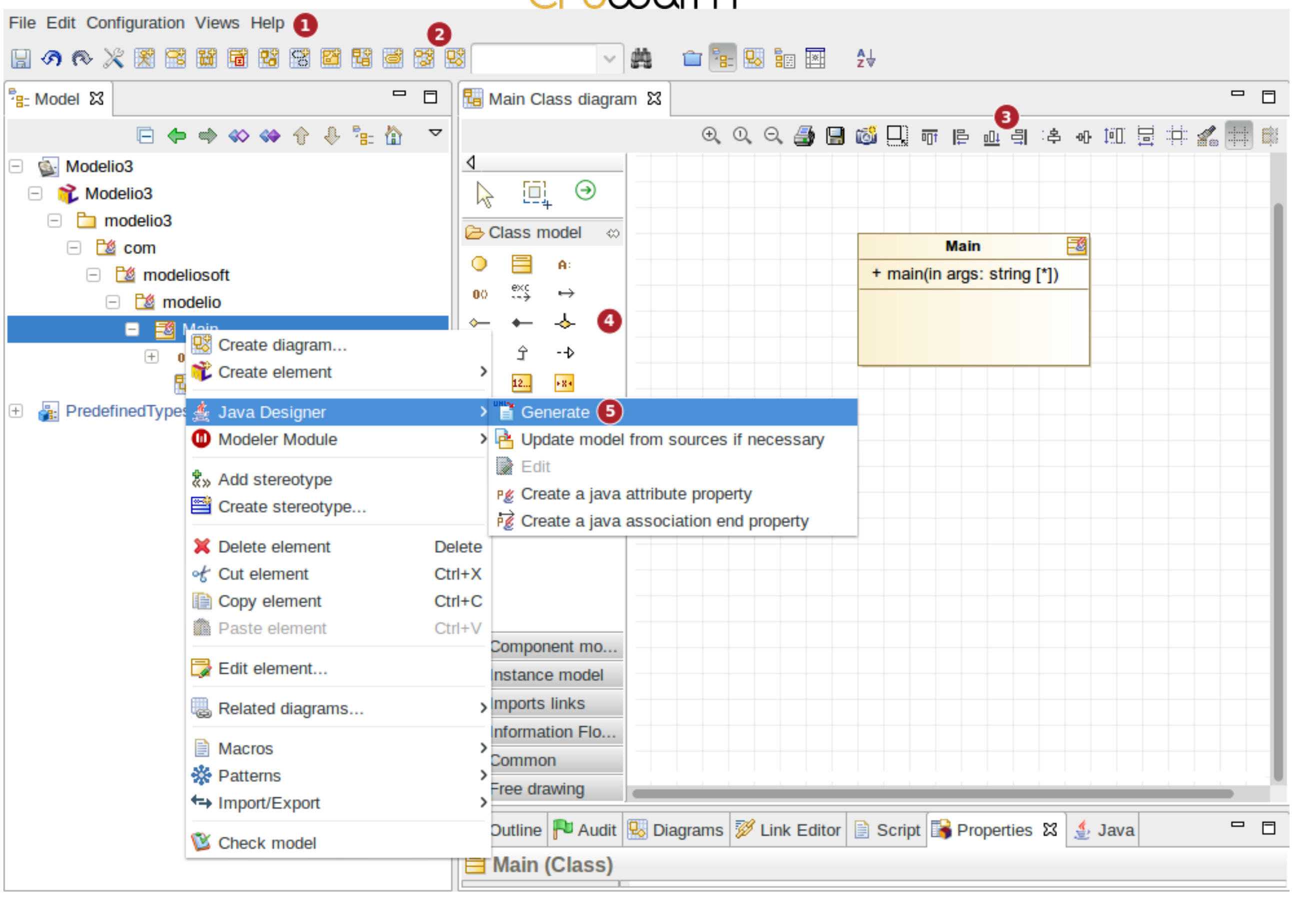


Figure 6 – Different kind of commands

Menu and toolbar commands (1), (2) and (3) are provided by Modelio and will be always displayed.
Diagram palette commands (4) are filtered according to both the type of the currently selected model element and the type of diagram which is active. For example, some commands will appear only for classes if the currently active diagram is a static diagram, but will be hidden when a class is selected in a component diagram.
Contextual menu commands (5) appear on user request (right click) in a dedicated sub-menu whose name is the name of the module ("Java Designer" in our screenshot example). They are filtered according to the type of the currently selected model element. For example, some commands will appear only for classes (when a class is selected), while others will be available only when an attribute is selected.
Two contextual menu commands are currently implemented inside CPSWarm extension. Both of them are detailed in the following sections. Figure 7 shows how one of them (the Swarm Template generation) is defined in the module.xml file.

```
<Command id="TemplateCreation" label="%command.TemplateCreation.label" tooltip="%command.TemplateCreation.tooltip" image="" modify-model="true">
    <Scope metaclass="Standard.Package" stereotype=""/>
    <Handler class="org.modelio.module.cpswarm.command.explorer.TemplateCreation"/>
</Command>
```

Figure 7 – CPSwarm command declaration

3.2.3.1 Swarm Template Generation

Modelling is always a difficult task want it start from scratch. In this case, guidance is helpful. The main goal of this swarm template generation command is to help the Modeler by creating a simple CPs swarm model with all minimum concept. The CPS swarm generation can be done by right clicking on any package, then selecting CPSwarm > CPS swarm creation entry as depicted in the following figure (Figure 8).

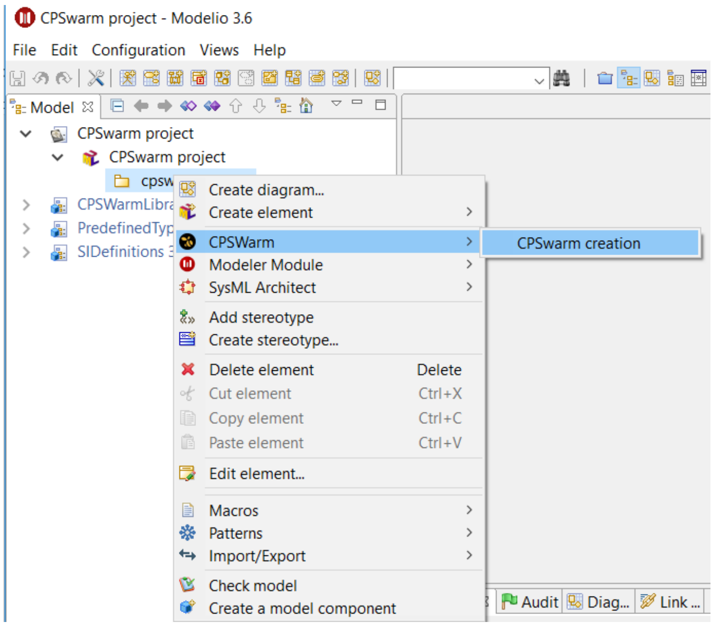


Figure 8 – Creating a new swarm modelling

Figure 9 show the result of the CPS swarm template generation.

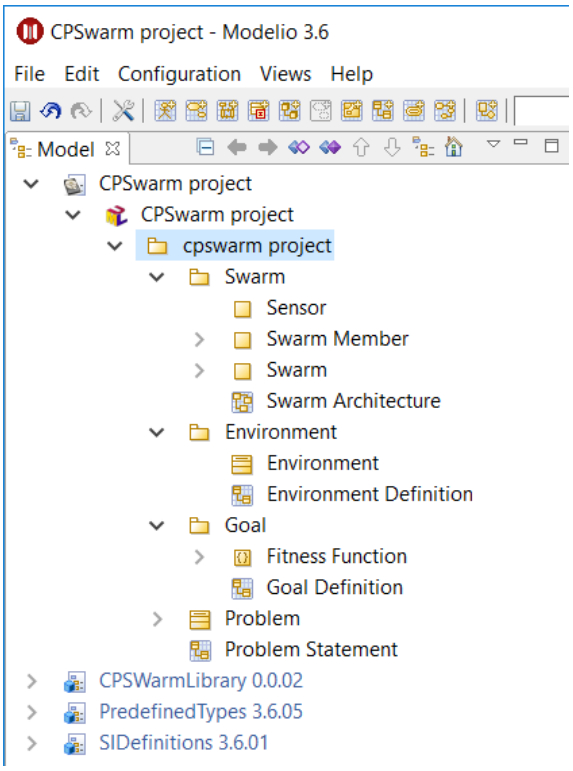


Figure 9 – New swarm result

The swarm template generator produces a set of initial diagrams (as shown in Figure 10) that have been identified as necessary to completely model a CPS swarm.

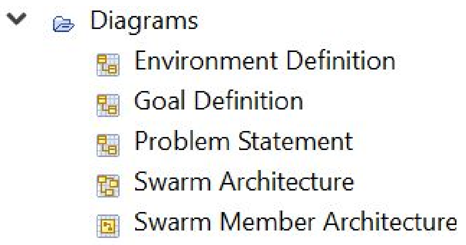
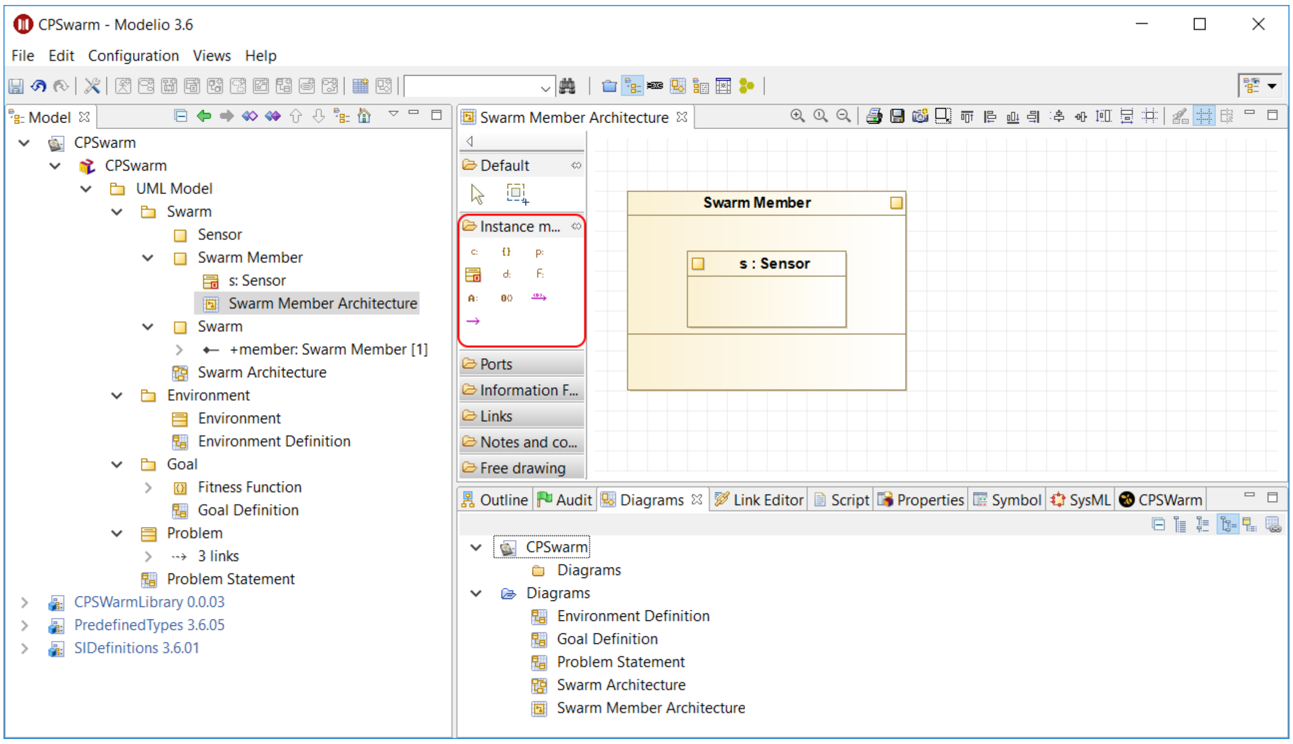

Diagrams
- Environment Definition
- Goal Definition
- Problem Statement
- Swarm Architecture
- Swarm Member Architecture

Figure 10 – CPSwarm predefined diagrams

The CPSwarm modeler can modify his/her initial content by following the needs of the specific case study he/she is modelling. The CPSwarm modeler will find on the right part of each of the diagrams (namely Environment Definition, Goal Definition, Problem Statement, Swarm Architecture, Swarm member architecture) the predefined selection of the modelling elements he/she can specifically use for that specific diagram context. Figure 11 and Figure 12 are showing the specific modelling environment for respectively the Swarm Architecture Diagram and for the Swarm Member Architecture Diagram. The two diagrams are meant to show the high-level structure of the swarm architecture and of the swarm member.


CPSwarm - Modelio 3.6
File Edit Configuration Views Help
Model
Swarm Member Architecture
Default
Instance m...
Ports
Information F...
Links
Notes and co...
Free drawing
Swarm Member
s : Sensor
Outline Audit Diagrams Link Editor Script Properties Symbol SysML CPSwarm

```
CPSwarm
  CPSwarm
    UML Model
      Swarm
        Sensor
        Swarm Member
          s: Sensor
          Swarm Member Architecture
        Swarm
          +member: Swarm Member [1]
        Swarm Architecture
      Environment
        Environment
        Environment Definition
      Goal
        Fitness Function
        Goal Definition
      Problem
        3 links
        Problem Statement
CPSWarmLibrary 0.0.03
PredefinedTypes 3.6.05
SIDefinitions 3.6.01
```

```
CPSwarm
  Diagrams
Diagrams
  Environment Definition
  Goal Definition
  Problem Statement
  Swarm Architecture
  Swarm Member Architecture
```

Figure 11 - Swarm Member Architecture Modelling Elements

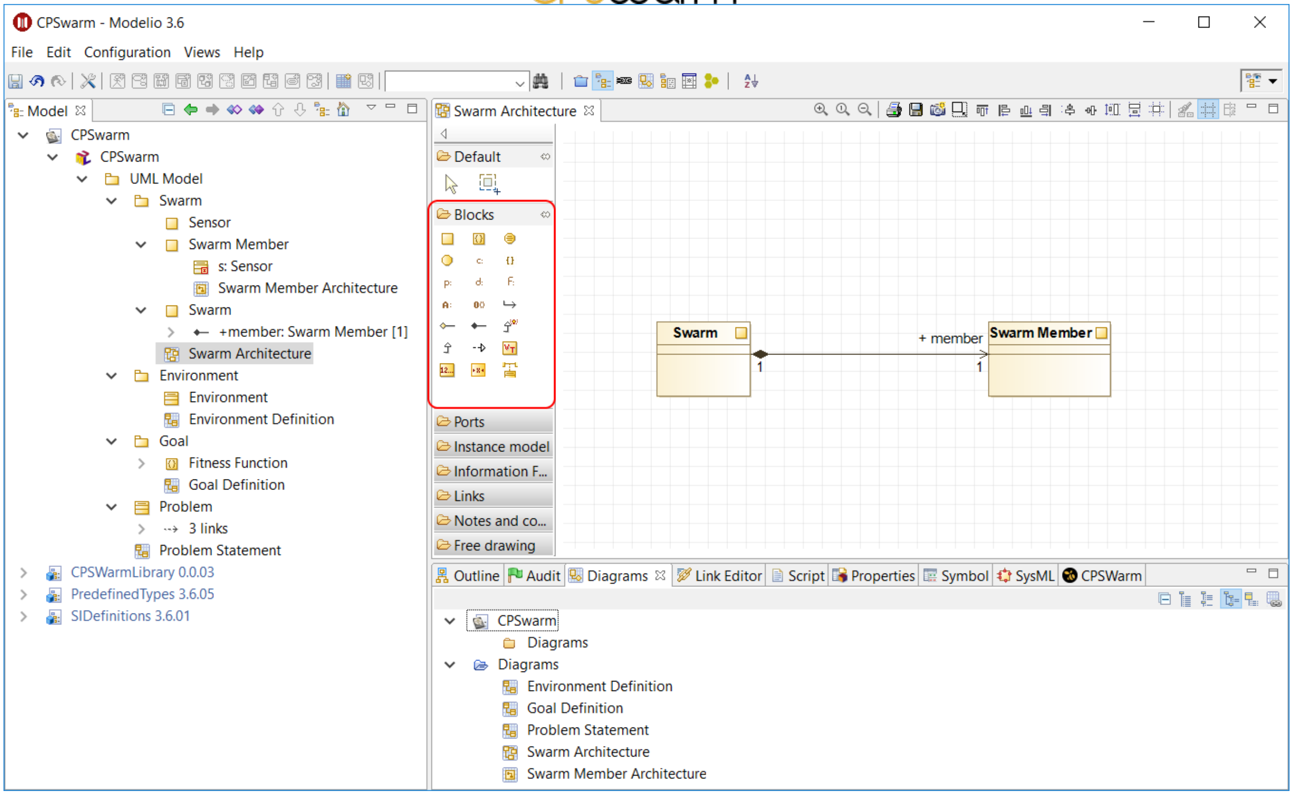


Figure 12 - Swarm Architecture Modelling Elements

A more detailed explanation of the diagrams and of the CPSwarm customized diagrams dedicated to the methodology will be provided in D5.1.

3.2.3.2 Optimization Project Generation

Once the CPS swarm is modelled, the next steps – in CPSwarm process as defined in D2.1- is to optimize it by using the Algorithm Optimization Environment. To do so Information must be passed from the Modelling Tool to the Optimization Tool through the Optimization API. This can be done by exporting using the Optimization Project generation. The Export can be done by right clicking on any CPS Problem, then selecting CPSwarm > Optimization Project generation entry as depicted in Figure 13.

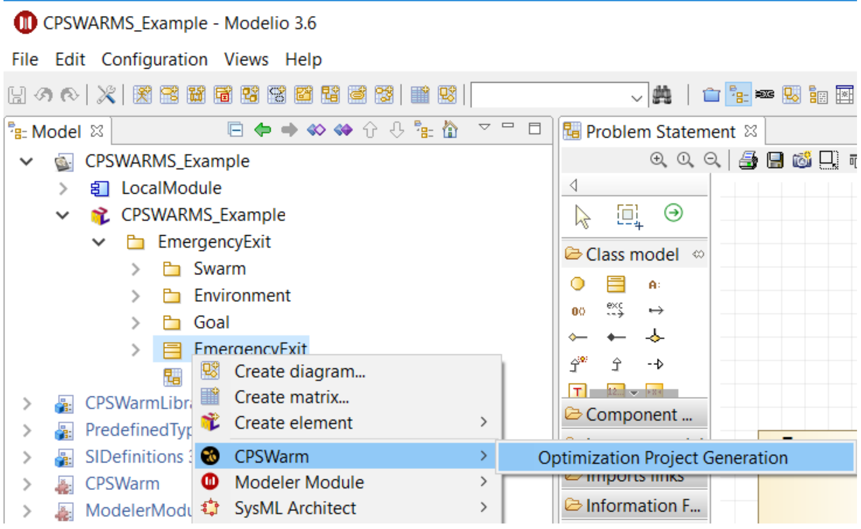


Figure 13 – Exporting the files for the optimization tool from the design environment.

3.2.4 CPSWarm Event Listener

From its deployment into user project to its operational use, the CPSwarm extension runs through several states that are called "runtime phases". Runtime phases are very important for the developer who must provide the Java code to be executed when certain events occur.
The runtime events at runtime are:

- **Install**: when the extension is installed (unzipped)
- **Selection**: when the extension is deployed in a project
- **Start**: when a project in which the extension has been deployed is opened
- **Stop**: when a project in which the extension has been deployed is closed
- **Upgrade**: when a new version of an extension is selected in the current project

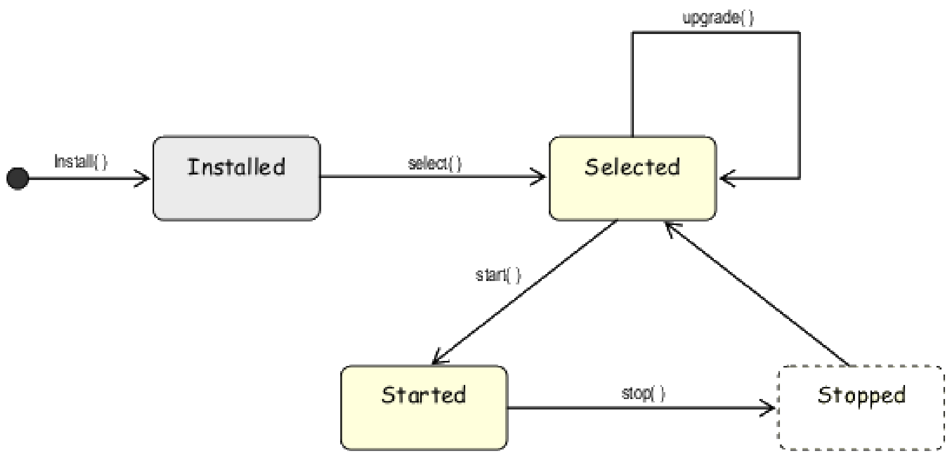


Figure 14 – Extension runtime states and events

At the "Start", the CPSwarm extension deploys the CPSwarm modelling library to make it available for current modelling. The CPSwarm extension starts by checking the current version of deployed Modelling library and checks if a newest version of this library is available. If a more recent version is available, the CPSwarm extension call the Library API to deploy the latest one inside the project. As result, the Modeller always design CPS with the most up to date version of the CPSwarm model library.
The Figure 15 shows CPSwarm Library component deployed inside the Modelling Tool.

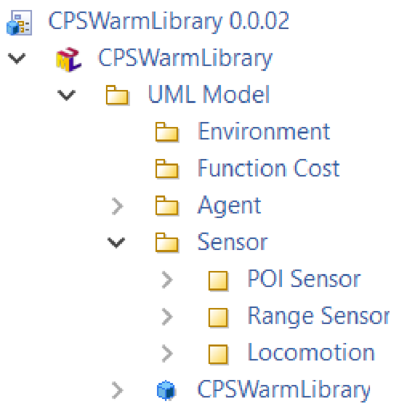


Figure 15 – CPSwarm Library deployed in the Modelling Tool

3.3 Graphical User Interface

A CPS swarm model contains several elements which can be sorted into three domains. That's the reason why three packages - respectively named "Swarm", "Environment", and "Goal" - are created when using CPS swarm template generation. This sorting has been defined to guide the user and separate the different concepts involved in the CPS swarm definition. The following sections described more in details each of these concepts by using the EmergencyExit example already provide as part of CPS model catalog (Cf. D4.1).

3.3.1 Swarm Modelling

In CPSwarm project, CPS swarms are described by the fact that a Swarm is composed by a set of Swarm Members but also the fact that Swarm Member(s) is (are) composed of different sub-component as for example sensors and actuators connected with each other. These compositions are made by using SysML language and more precisely a Block Definition Diagram (BDD) for the Swarm Architecture as depicted in Figure 16 and an Internal Block Definition Diagram (IBD) for the Swarm Member Architecture cf. Figure 17.

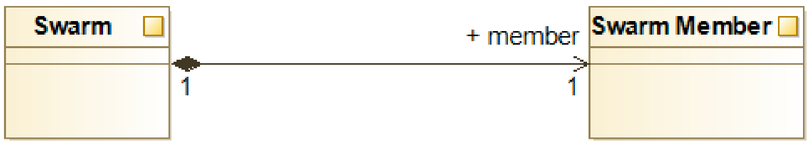


Figure 16 – Swarm Architecture

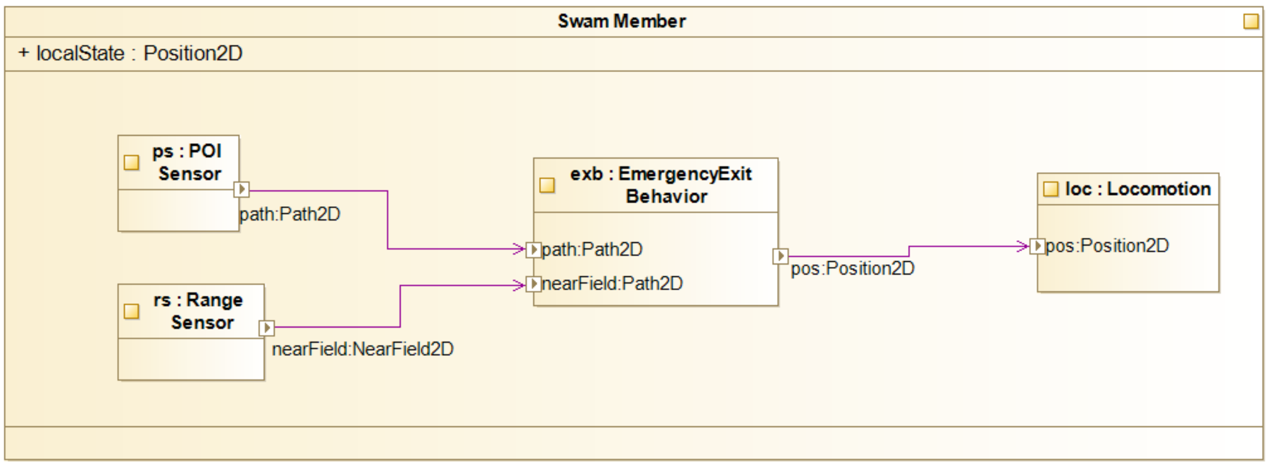


Figure 17 – Swarm Member Architecture

3.3.2 Environment Modelling

The environment modelling is related to the CPS swarm deployment. An abstract representation of the environment is made and will used by the simulator for CPS Deployment. The Figure 18 depicts a 2D environment used in the EmergencyExit example with a specific height and width. This 2D field also has 5 Exits and 20 Obstacles with their 2D position (xpos, ypos).

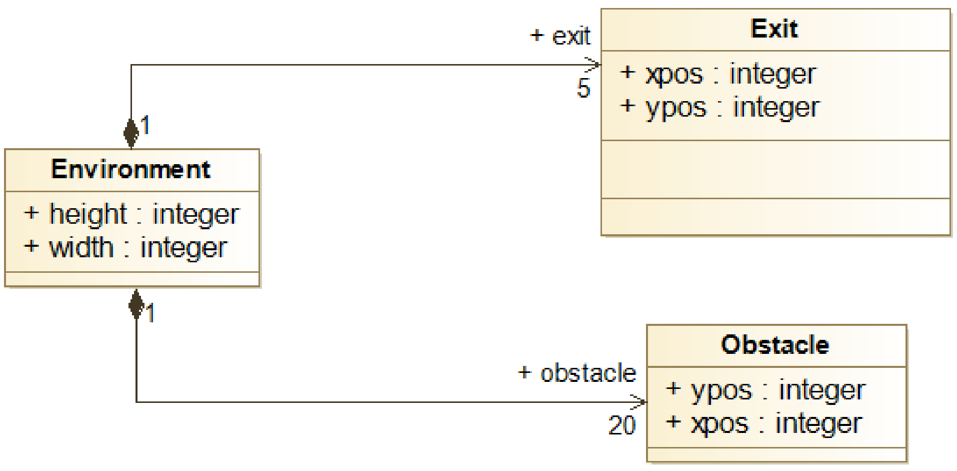


Figure 18 – Environment Example

3.3.3 Fitness Function Modelling

The Fitness Function represents the *goal* of the specified CPS swarm. By maximizing this fitness function, the Algorithm Optimization Environment (cf. Deliverable D6.1) can provide the best CPS swarm configuration. Figure 19 corresponds to the goal of the EmergencyExit problem where the swarm must exit to a given field as soon as they can. The result of the fitness function is the average speed of the swarm given a specific distance and the field (nearField) around it.

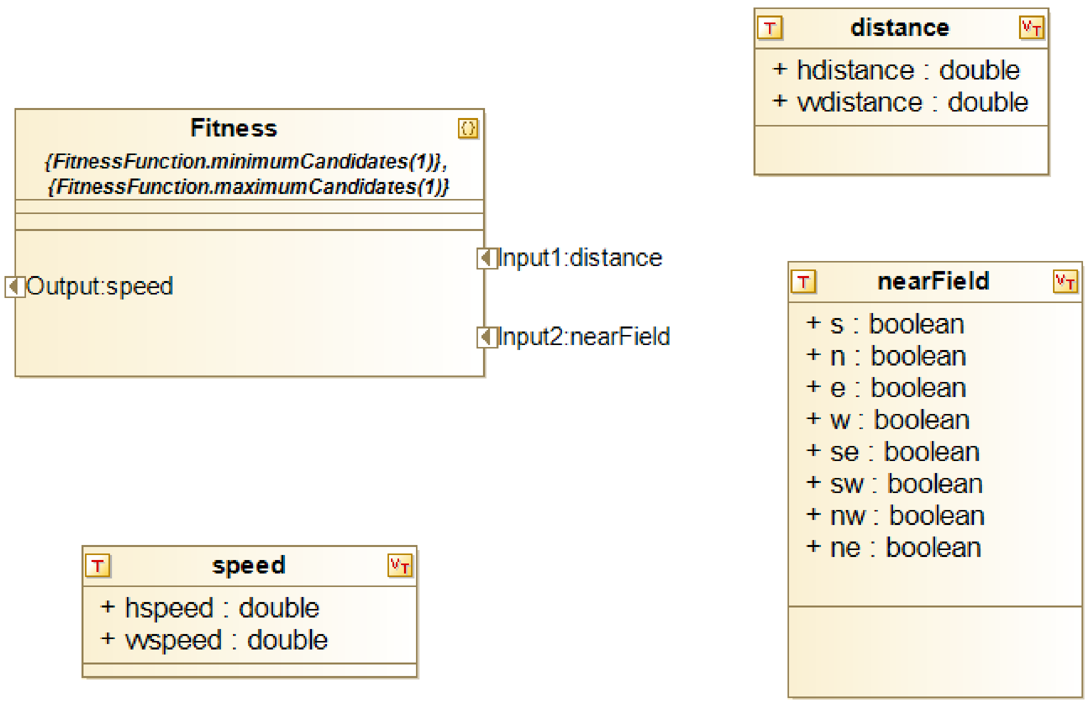


Figure 19 – Fitness Function definition example

Finally, the Problem is defined by selecting a Swarm, deployed in a given Environment with a given Goal (Fitness Function) as shown in Figure 20.

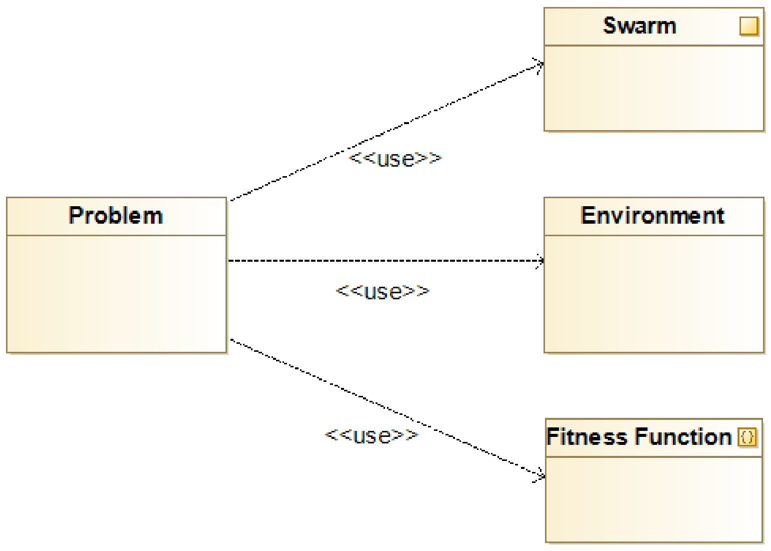


Figure 20 – Problem Statement

4 Optimization Tool API

4.1 Description

The optimization tool API provides all modeling details to the algorithm optimization environment. They are received by the optimization tool and further interpreted. The models that are required for simulation are forwarded to the optimization simulator through the simulator API.

Since both the modeling tool and the optimization tool run on the same machine, no network communication is required for this API. Instead, the models are transferred from Modelling Tool to Optimization Tool by means of files. There are two files provided, a problem description file and a parameters file. The problem description file is a Java class that implements the problem to be optimized. The parameters file complements the problem description as it specifies all model parameters as an XML file. They are defined during the modeling process in the Modelling Tool and are filled with a default value. During the optimization phase, it is possible to change these values for evaluating the optimized controller under variable conditions. Once the modeling is completed, these files are exported into the Modelio workspace. Then they need to be copied manually into the Optimization Tool workspace. The Java class implements the following methods.

- *evaluateCandidate(AbstractRepresentation candidate)*:
  This function starts the simulation process using a specific candidate controller. The abstract representation is used to translate the incoming sensor readings to outgoing actuator commands. Once the simulation is over, this function returns the fitness score of the candidate as a double value.
- *replayWithVisualization(AbstractRepresentation candidate)*:
  This function starts the simulation process with a specific candidate controller showing a GUI to visualize the simulation. It is used for introspection and analysis of the optimized solution.
- *getMaximumFitness()*:
  This function returns the highest possible fitness score. If a candidate controller reaches this value, Optimization Tool stops the optimization process.

The XML file is used by the problem in Optimization Tool to display the parameters in the GUI. They all have a default value preset but can be changed from the GUI. When a simulation starts they are read and transmitted to the optimization simulator as part of the parameters message. The XML file contains the parameters listed in Section 4.2.2 as part of the properties. Besides the properties, also configuration parameters and requirements are listed in the XML file. The configuration parameters describe the problem with name and textual description. The requirements specify the number of inputs (sensors) and outputs (actuators) the agents have as well as bounds on the number of agents that can solve the problem. It is shown in Section 4.2.1.

4.2 File Formats

The Modelling Tool passes configuration files to the Optimization Tool that uses a format consistent with the optimization tool API. This configuration consists of two files: the problem description and the optimization parameters. Both parts are exported as depicted in Figure 13.

Once these files are passed to the Optimization Tool, they can be used to start simulations and the optimization Process - both files are explained in the following sections.

4.2.1 Problem Description

The problem description is a generated .java file. It contains everything that is necessary to test and evaluate a possible solution. This functionality is used by the optimization tool to perform an automated search for a viable solution. The problem description implements the following tasks:

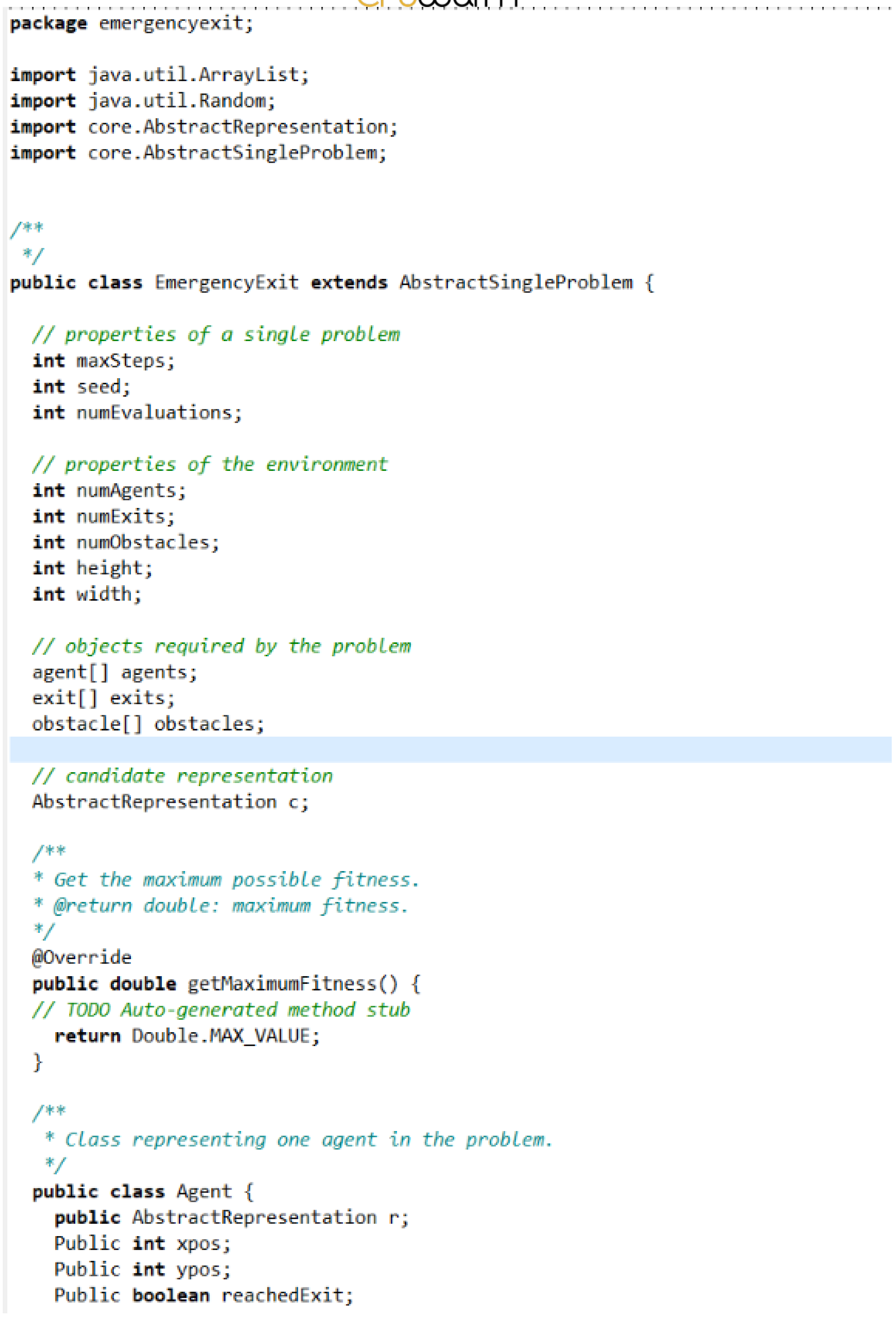
- reading the optimization parameters
- providing the optimization parameters to the optimization simulator
- setting up the optimization simulator
- running simulations

- calculating the fitness score

This is achieved by implementing the functions of the abstract class `AbstractSingleProblem`:

- `loadParameters`: reads the parameters from the XML file and provide them to the interface to the optimization simulator
- `calcFitness`: calculates the fitness score from the log files provided by the interface to the optimization simulator
- `evaluateCandidate`: evaluates a specific controller candidate using the interface to the optimization simulator
- `replayWithVisualization`: displays the behavior of a specific controller candidate using the interface that calls the optimization simulator in visual mode
- `getMaximumFitness`: returns the maximum possible fitness

An exemplary problem definition is given in Figure 21.



```
package emergencyexit;

import java.util.ArrayList;
import java.util.Random;
import core.AbstractRepresentation;
import core.AbstractSingleProblem;


/**
 */
public class EmergencyExit extends AbstractSingleProblem {

  // properties of a single problem
  int maxSteps;
  int seed;
  int numEvaluations;

  // properties of the environment
  int numAgents;
  int numExits;
  int numObstacles;
  int height;
  int width;

  // objects required by the problem
  agent[] agents;
  exit[] exits;
  obstacle[] obstacles;

  // candidate representation
  AbstractRepresentation c;

  /**
  * Get the maximum possible fitness.
  * @return double: maximum fitness.
  */
  @Override
  public double getMaximumFitness() {
  // TODO Auto-generated method stub
    return Double.MAX_VALUE;
  }

  /**
   * Class representing one agent in the problem.
   */
  public class Agent {
    public AbstractRepresentation r;
    Public int xpos;
    Public int ypos;
    Public boolean reachedExit;
```

Figure 21 – Problem definition example.

4.2.2 Optimization parameters (XML)

The parameters are available in a .xml file. They describe the setup of Optimization Tool. These parameters include configuration, properties, and requirements (see Figure 22).

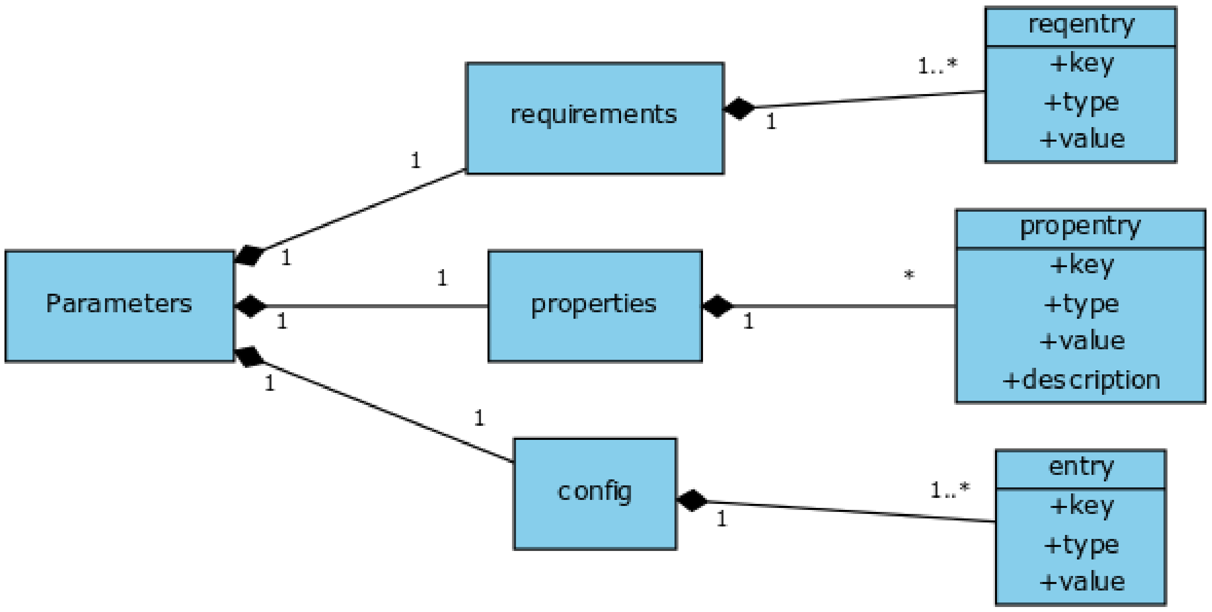


Figure 22 – Parameters for the optimization tool.

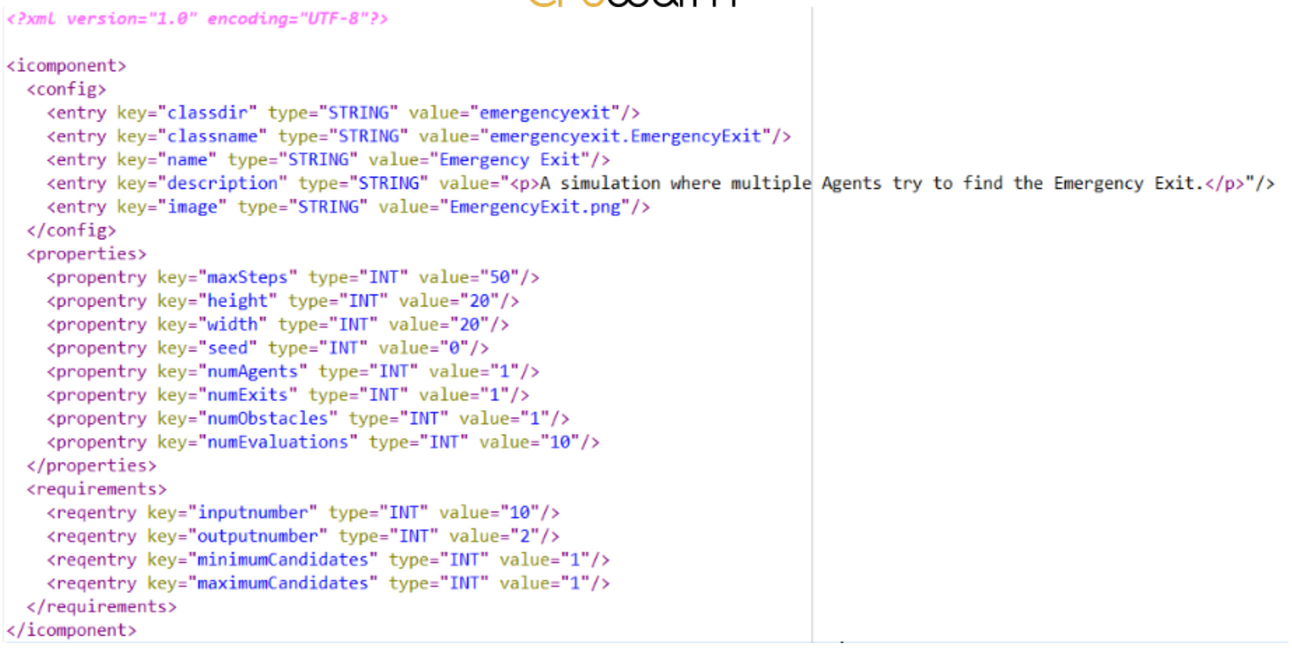
The *configuration parameters* describe the problem that is modeled with a set of strings.
The *properties* are a set of parameters that are not fixed during the modeling phase but left open for the optimization phase. However, these parameters have a default value set by the modeling tool. These parameters are used to evaluate the algorithm with different setups. This is useful to achieve a robust algorithm that works in different environments. There are no mandatory parameters. Exemplary parameters in the properties section are:

- description of agents, e.g. cardinality, positioning, size
- description of environment, e.g. size, obstacles, POI

The *requirements* define important parameters for the evolutionary optimization process. These parameters are:

- *inputnumber*: the number of inputs for the controller, i.e. the number of sensed inputs, each input has the type of a single scalar value, but depending on the problem, the input signal could also use only a part of the possible values.
- *outputnumber*: the number of outputs of the controller, i.e. the number of actuator controls, each output corresponds to a single scalar value, which might be interpreted further.
- *minimumCandidates*: the minimum number of agents - aka swarm members - necessary to address the problem.
- *maximumCandidates*: the maximum number of agents – aka swarm member - that can be used in the problem.

An exemplary parameter's .xml file coming from the EmergencyExit example is given in Figure 23.



```xml
<?xml version="1.0" encoding="UTF-8"?>

<icomponent>
  <config>
    <entry key="classdir" type="STRING" value="emergencyexit"/>
    <entry key="classname" type="STRING" value="emergencyexit.EmergencyExit"/>
    <entry key="name" type="STRING" value="Emergency Exit"/>
    <entry key="description" type="STRING" value="<p>A simulation where multiple Agents try to find the Emergency Exit.</p>"/>
    <entry key="image" type="STRING" value="EmergencyExit.png"/>
  </config>
  <properties>
    <propentry key="maxSteps" type="INT" value="50"/>
    <propentry key="height" type="INT" value="20"/>
    <propentry key="width" type="INT" value="20"/>
    <propentry key="seed" type="INT" value="0"/>
    <propentry key="numAgents" type="INT" value="1"/>
    <propentry key="numExits" type="INT" value="1"/>
    <propentry key="numObstacles" type="INT" value="1"/>
    <propentry key="numEvaluations" type="INT" value="10"/>
  </properties>
  <requirements>
    <reqentry key="inputnumber" type="INT" value="10"/>
    <reqentry key="outputnumber" type="INT" value="2"/>
    <reqentry key="minimumCandidates" type="INT" value="1"/>
    <reqentry key="maximumCandidates" type="INT" value="1"/>
  </requirements>
</icomponent>
```

Figure 23 – Example of parameter .XML file.

5 Conclusion

This deliverable describes the status of the CPSwarm Modelling Tool after the nine months of the project. The actual component supports CPS swarm model-based design, their simulation and optimization.
Even if these features have not yet been fully developed or integrated, they already show first interesting results and provide a good basis for the next developments.

6 References

[1] CPSwarm workbench project development http://forge.modelio.org/projects/cpswarm
[2] Softeam. Modelio Open-Source Modeling Environment. 2016. Available at: http://www.modelio.org/.
[3] Object Management Group (OMG). System modeling language specification v1.5. 2017. Available: http://www.omg.org/spec/SysML/1.5/

Acronyms

Acronym	Explanation
BDD	Block Definition Diagram
DoA	Description of Action
IBD	Internal Block Diagram
JDK	Java Development Kit
JRE	Java Runtime Environment
KPI	Key Performance Indicators
OMG	Object Management Group
SysML	System modeling language

List of figures

7 Annexes

7.1 Modelling Tool Installation Guide

7.1.1 Download

Modelio binary distribution archives are available on the Modelio download page https://www.modelio.org/downloads/download-modelio.html. Extension dedicated to CPSwarm project is downloadable from http://forge.modelio.org/projects/cpswarm-modelio36/files

7.1.2 Requirement

The minimum hardware configuration is:

- Pentium IV 1 Ghz or higher,
- Minimum 1GB memory, 3GB recommended,
- 700 MB hard drive or more.

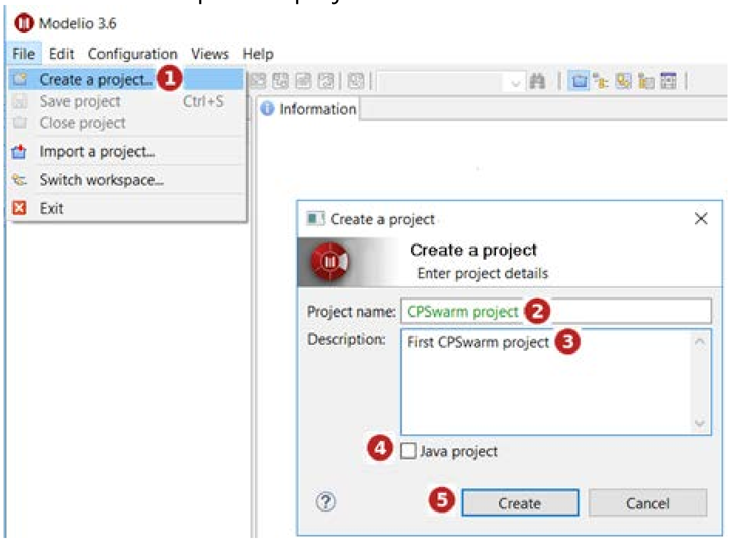
The CPSwarm modelling tool also need:
Java Runtime Environnement (JRE) 8 *(from Modelio 3.3)*. Type java -version in the console to see if you have the compatible version installed.

-
- MAC OS X requires the Java Development Kit (JDK) 8 (not only JRE)
- IE10+ (or any other web browser: Chrome, Firefox, ...) as system web browser for supporting HTML notes

7.1.3 First steps

To be able to model a swarm of CPS, the first step will be to create, as depicted in Figure 24, a Modelio project. To do so start by launching Modelio then:

1. Click on File > Create a project....
2. Enter the name of the project.
3. Enter the description of the project.
4. If it is envisaged that the project will be connected to a Java development workflow in the future (unrelated to CPSwarm), you can choose to include the Java Designer module by selecting Java Project, otherwise de-select this option.
5. Click on "Create" to create and open the project.

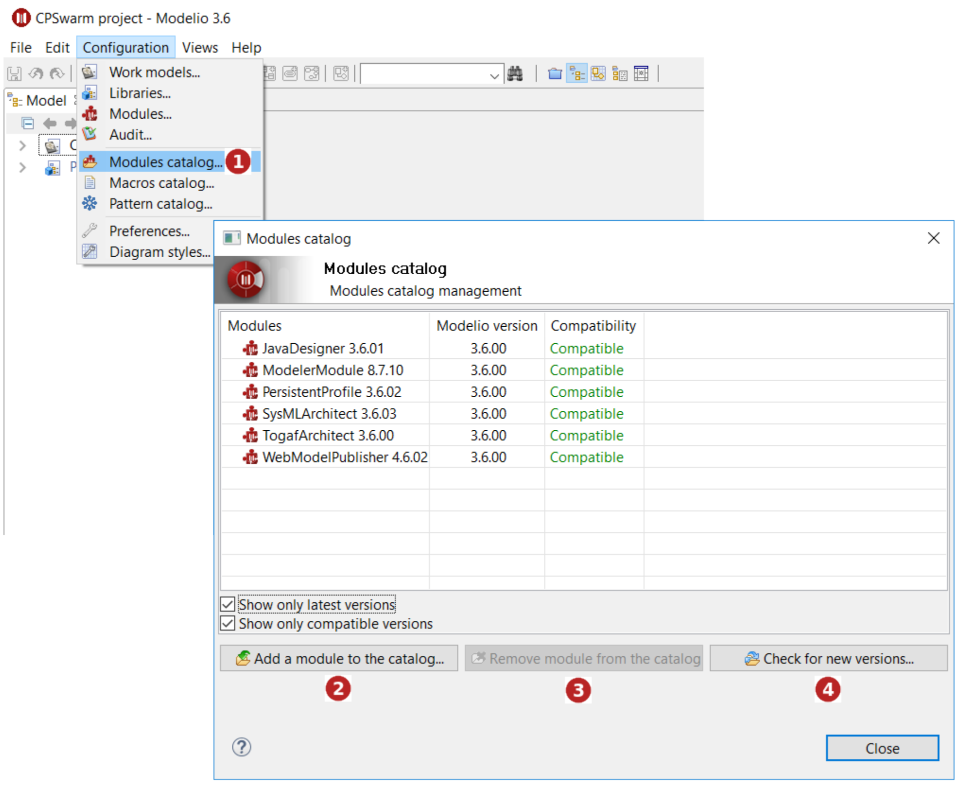


Project name: CPSwarm project
Description: First CPSwarm project

Figure 24 – Creating a new Modelio/CPSwarm project

Once you have successfully created a Modelio project, you must install the Modelio modules required for CPSwarm modelling, i.e. both Modelio SysML and CPSwarm modules. To deploy a given module into a Modelio project, it must be part of the module catalogue. To add modules into the module catalogue, you must, as depicted in Figure 25:

1. Run the Configuration > Modules catalog... command.
2. To add a module, click on "Add a module to the catalog..." and use the file browser to select the modules (*.jmdac files).
3. To remove a module, select the module in question and click on the "Remove module from the catalog" button.
4. To download new versions of modules into the catalog, click on "Check for new versions...."



Modules catalog

Modules catalog management

Modules	Modelio version	Compatibility
JavaDesigner 3.6.01	3.6.00	Compatible
ModelerModule 8.7.10	3.6.00	Compatible
PersistentProfile 3.6.02	3.6.00	Compatible
SysMLArchitect 3.6.03	3.6.00	Compatible
TogafArchitect 3.6.00	3.6.00	Compatible
WebModelPublisher 4.6.02	3.6.00	Compatible

☑ Show only latest versions
☑ Show only compatible versions

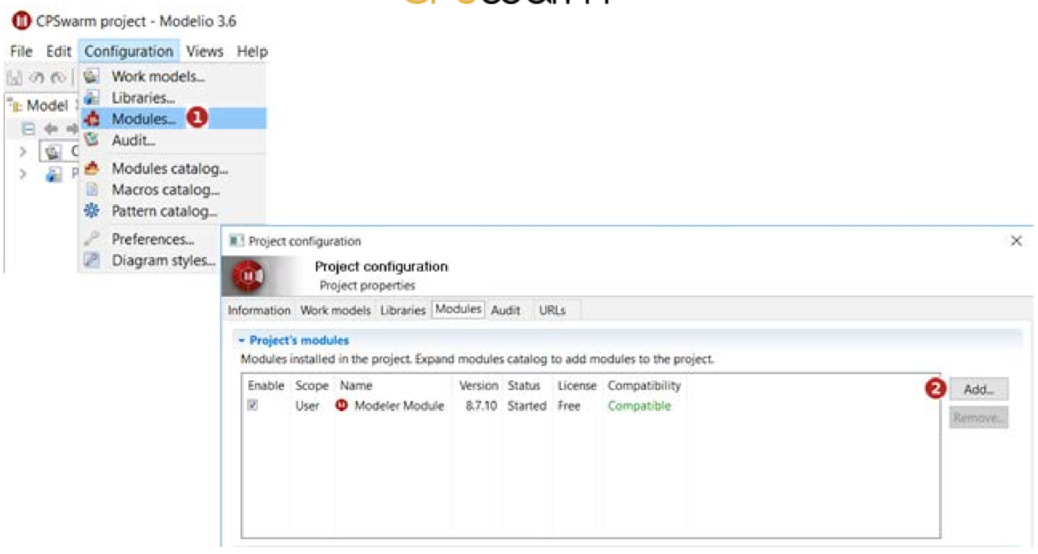
Add a module to the catalog... | Remove module from the catalog | Check for new versions...

Close

Figure 25 – Add CPSwarm module to Modelio catalog

Once the module is added to the catalogue, you must deploy it inside the project as shown in Figure 26. This is done by:

1. Run the Configuration > Modules... command;
2. Click on "Add..." to expand the Modules catalog.



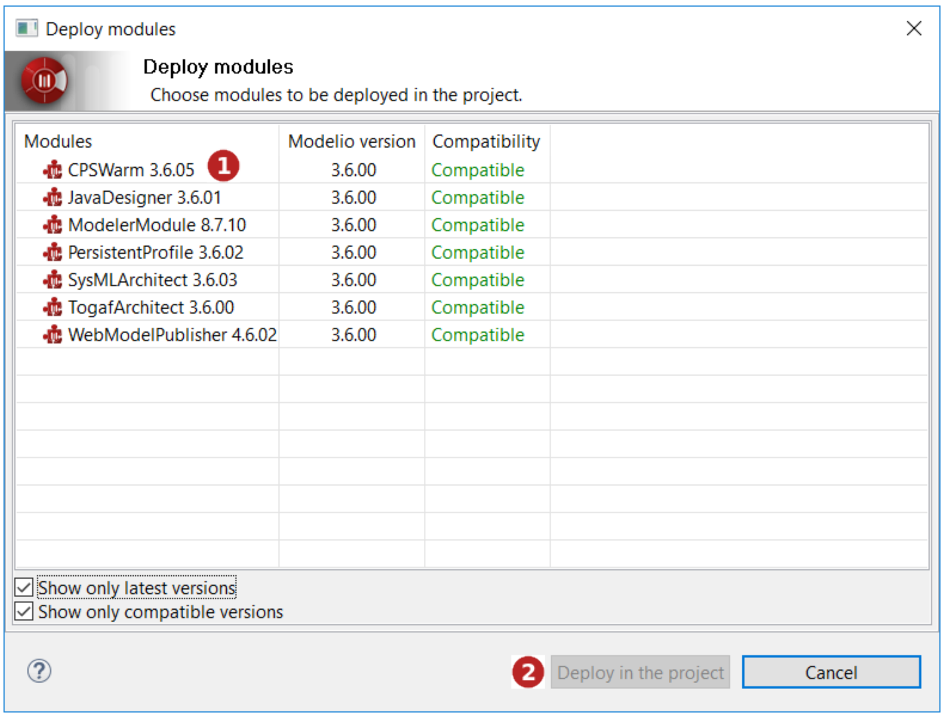
Modules installed in the project. Expand modules catalog to add modules to the project.

Enable	Scope	Name	Version	Status	License	Compatibility
☑	User	Modeler Module	8.7.10	Started	Free	Compatible

Figure 26 – Open the Module configuration

And in the modules catalog cf. Figure 27:

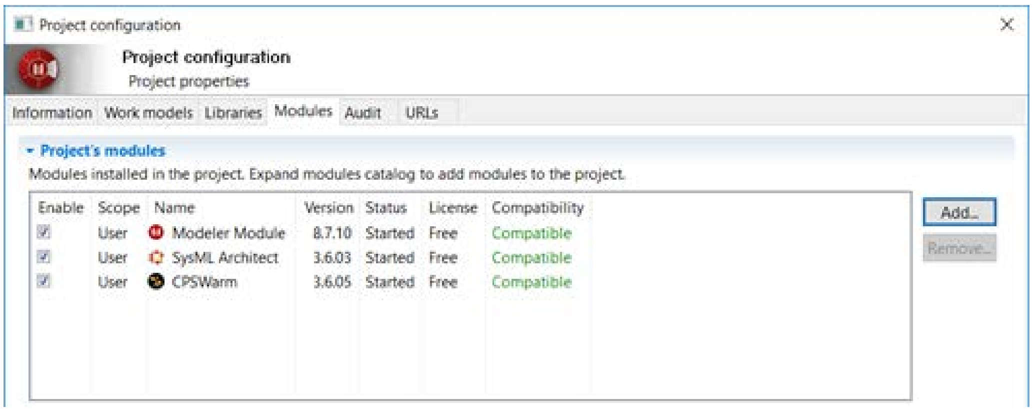
1. Select the module (CPSwarm) you want to install;
2. Click on “Deploy in the project” to install the module in the project.


Deploy modules
Deploy modules
Choose modules to be deployed in the project.
1
Show only latest versions
Show only compatible versions
2
Deploy in the project
Cancel

Modules	Modelio version	Compatibility
CPSWarm 3.6.05	3.6.00	Compatible
JavaDesigner 3.6.01	3.6.00	Compatible
ModelerModule 8.7.10	3.6.00	Compatible
PersistentProfile 3.6.02	3.6.00	Compatible
SysMLArchitect 3.6.03	3.6.00	Compatible
TogafArchitect 3.6.00	3.6.00	Compatible
WebModelPublisher 4.6.02	3.6.00	Compatible

Figure 27 – Deploy the CPSwarm module

By deploying the CPSwarm module, SysML module should also be deployed as depicted in Figure 28.


Project configuration

Project configuration
Project properties

Information | Work models | Libraries | Modules | Audit | URLs

Project's modules
Modules installed in the project. Expand modules catalog to add modules to the project.

Enable	Scope	Name	Version	Status	License	Compatibility
✓	User	Modeler Module	8.7.10	Started	Free	Compatible
✓	User	SysML Architect	3.6.03	Started	Free	Compatible
✓	User	CPSWarm	3.6.05	Started	Free	Compatible

Figure 28 – Deployed modules

7.1.4 Issue report

Issues can be reported by creating a new topic in the following discussion forum https://www.modelio.org/forum/index.html.

7.2 The Optimization Tool Installation Guide

7.2.1 Download

Download Optimization Tool from: https://sourceforge.net/projects/Optimization Tool/files/

We provide a short video for the installation steps of Optimization Tool (base of the CPSwarm Optimization Tool): https://www.youtube.com/watch?v=1wTyozYGG4I

7.2.2 Requirement

The optimization Tool requires the Java Runtime Environment (version min. 1.6) to be installed. Type java -version in the console to see if you have the compatible version installed.

7.2.3 First steps

7.2.3.1 Running the Optimization Tool

To start Optimization Tool with its GUI, follow these steps:

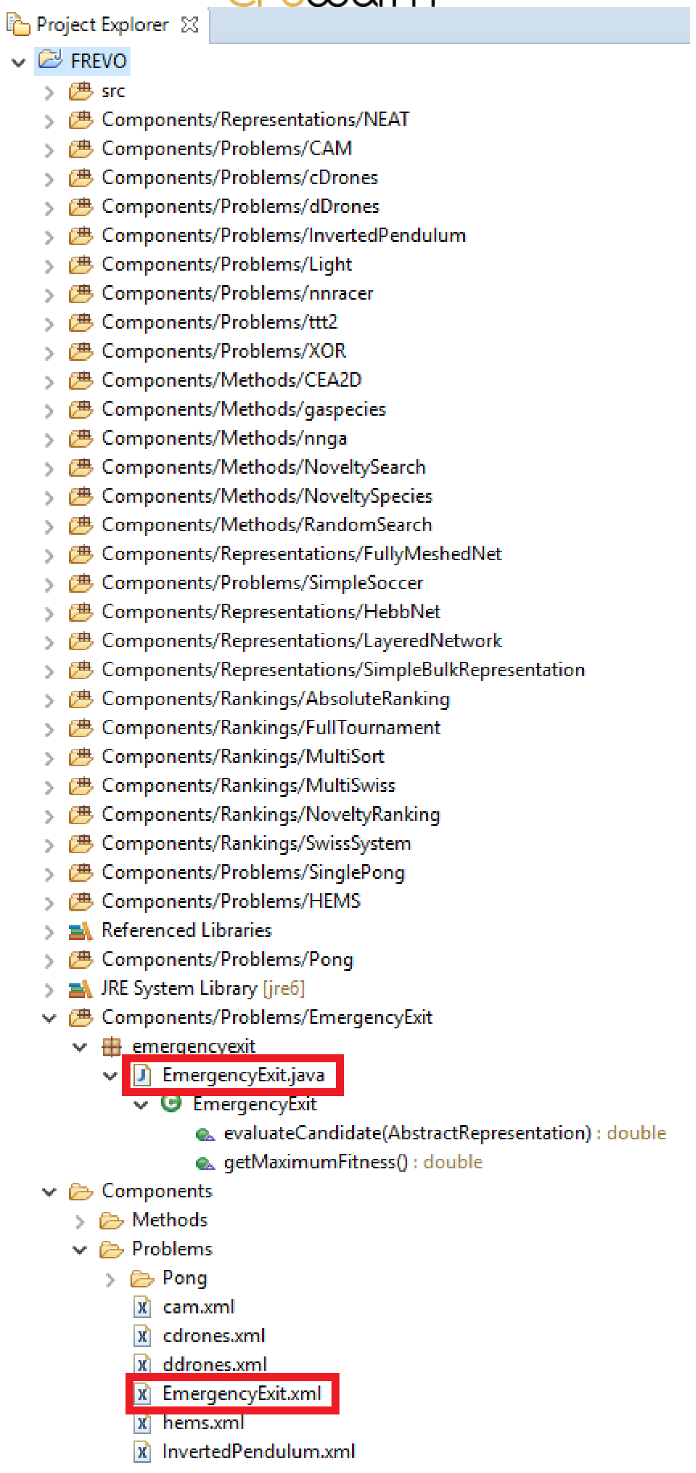
1. Start a console window and type *java -jar createscripts.jar*
   This will create the script files used to start the Optimization tool.
2. Based on your operating system, run the following file:
   - *Windows: launch_Optimization Tool.bat*
   - *Unix/Linux/Mac OSX: ./launch_Optimization Tool.sh*

7.2.4 Issue report

Issues can be reported by creating a new topic in the following discussion forum https://sourceforge.net/p/Optimization Tool/discussion/

7.2.5 Graphical User Interfaces

When a new problem needs to be modelled, one can use the component creator from the top menu. After selecting component type, name, package, and description the code skeleton is created. The code skeleton is placed in a subdirectory of the directory *components* in the optimization tool. It provides a programming interface with comments explaining where further implementation is required (see Figure 29). In addition, an .xml file with the name of the component is generated, where configuration parameters as well as sensor inputs and actuator outputs of the CPS can be defined.



```
Project Explorer
FREVO
  src
  Components/Representations/NEAT
  Components/Problems/CAM
  Components/Problems/cDrones
  Components/Problems/dDrones
  Components/Problems/InvertedPendulum
  Components/Problems/Light
  Components/Problems/nnracer
  Components/Problems/ttt2
  Components/Problems/XOR
  Components/Methods/CEA2D
  Components/Methods/gaspecies
  Components/Methods/nnga
  Components/Methods/NoveltySearch
  Components/Methods/NoveltySpecies
  Components/Methods/RandomSearch
  Components/Representations/FullyMeshedNet
  Components/Problems/SimpleSoccer
  Components/Representations/HebbNet
  Components/Representations/LayeredNetwork
  Components/Representations/SimpleBulkRepresentation
  Components/Rankings/AbsoluteRanking
  Components/Rankings/FullTournament
  Components/Rankings/MultiSort
  Components/Rankings/MultiSwiss
  Components/Rankings/NoveltyRanking
  Components/Rankings/SwissSystem
  Components/Problems/SinglePong
  Components/Problems/HEMS
  Referenced Libraries
  Components/Problems/Pong
  JRE System Library [jre6]
  Components/Problems/EmergencyExit
    emergencyexit
      EmergencyExit.java
        EmergencyExit
          evaluateCandidate(AbstractRepresentation) : double
          getMaximumFitness() : double
  Components
    Methods
    Problems
      Pong
      cam.xml
      cdrones.xml
      ddrones.xml
      EmergencyExit.xml
      hems.xml
      InvertedPendulum.xml
```

Figure 29 – OPTIMIZATION TOOL directories.

As both, the .java and .xml file were already generated in the modelling tool, they are replaced by them with same name as provided by the Optimization Tool. The way over the component creator is still necessary to setup the right folder structure.

The main task is to implement the *evaluateCandidate* method, where the given candidate representation needs to be evaluated. This is done by simulation, either by implementing it directly within Optimization Tool or by calling an external simulator. Therefore, environment and CPS need to be implemented. The sensor input(s) of the CPS are passed to the *getOutput* method of the candidate representation, which returns the output for the actuator(s). A suitable performance measure needs to be implemented that is used to return the fitness value of the simulation run.

After the implementation of a component is complete, the component needs to be compiled. It is then automatically loaded upon the launch the Optimization Tool.