Modelling a CPS Swarm System: A Simple Case Study

Melanie Schranz[1], Alessandra Bagnato[2], Etienne Brosse[2] and Wilfried Elmenreich[3]
[1]*Lakeside Labs, Klagenfurt, Austria*
[2]*Softeam, Research and Development Department, Paris, France*
[3]*Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria*

Keywords: Cyber Physical Systems(CPS), Unified Modeling Language (UML), Swarm of CPS, System Modeling Language (SysML), Swarm Algoritms, Optimization.

Abstract: The CPSwarm workbench is a toolchain that facilitates the entire design process of swarms of CPS including modelling, design, optimization, simulation and deployment. This paper highlights part of the work of the CPSwarm workbench in the context of the CPSwarm H2020 project. In particular, the CPSwarm workbench allows to create a generic swarm library that can be customized by developers to design new swarm environments, new swarm members and new swarm goals. This paper shows an application of the initial CPSwarm workbench by the example of a reference problem called EmergencyExit. In this example a swarm of robots needs to find an exit in an unmapped environment and leave this room through the exit as soon as possible. The example problem is further used to show the integration of Modelio, a UML/SysML modelling tool, and FREVO, an optimization tool in the CPSwarm workbench.

1 INTRODUCTION

Cyber Physical Systems (CPSs) are characterized by the strong integration of software and hardware (Martins and McCann, 2017). Their concept enriches embedded devices by interacting with the physical world through sensors and actuators. A typical characteristic is the strong focus on a high interconnection among CPSs. As tasks are getting complex with swarms of autonomous drones, cars, or ground rovers, systems of swarms of CPSs become challenging to design, implement, optimize, and deploy (Lee, 2008). To address this problem, we suggest to build a toolchain covering the development steps, starting with design of a swarm of CPS, optimization and simulation of the used algorithms, and deployment on the final piece of hardware.

In this work, we focus on modelling a swarm of CPS. This includes adapting and extending existing approaches as well as developing new CPS models where no reference model is available. Of special interest are available open source approaches and examples like developed in the INTO-CPS project[1]. The INTO-CPS project makes use of UML and SysML (Bagnato et al., 2016) profile. It aims to develop a model-based tool chain for CPS development, applying model-based systems engineering (MBSE): high level simulation using the FMI standard between the simulation engine and the related modelling tools such as Overture, OpenModelica, and 20-sim; C code generation for specific platforms; and test automation is also in the scope of the project via the usage of tools such as RT-Tester Model-Checker (Larsen et al., 2016), (Bagnato et al., 2015). In order to design swarm systems, we refer to related approaches from designing self-organizing systems for technical applications (Elmenreich and de Meer, 2008), which involves bio-inspired design, trial and error and learning from optimal solutions that use additional information, like perfect sensors, which are removed later (Auer et al., 2008).

This work contributes to the EU-H2020 project CPSwarm[2] by focusing on the modelling and optimization of swarms of CPSs (Bagnato et al., 2017). Therefore, we introduce the EmergencyExit example that is used as use case for the initial library of CPS models. In the EmergencyExit example, multiple swarm members move in a simple 2D discrete environment and try to find one out of the two emergency exits. In each discrete time step, a CPS senses the neighbou-

[1]http://projects.au.dk/into-cps/

[2]https://cpswarm.eu

Schranz, M., Bagnato, A., Brosse, E. and Elmenreich, W.
Modelling a CPS Swarm System: A Simple Case Study.
DOI: 10.5220/0006731106150624
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 615-624
ISBN: 978-989-758-283-7

ring cells and moves to a free cell. When a CPS reaches an emergency exit, it is removed from the environment. The goal is that all CPSs exit the environment.

For this case study, we built upon the following tools: the modelling tool Modelio developed by Softeam and the optimization tool FREVO (Sobe et al., 2012) developed by Lakeside Labs/Alpen-Adria-Universitt Klagenfurt.

The rest of this paper is organized as follows: Section 2 provides a background about CPSwarm, Modelio and FREVO. Section 3 summarizes the CPS swarm model indroduced by CPSwarm. Section 4 describes the workbench approach and shows how Modelio and FREVO are used to achieve the CPSwarm goals. Section 4 depicts the results and discusses how the evolved algorithms can be evaluated. Section 5 concludes the paper.

2 BACKGROUND

2.1 CPSwarm Project

The main motivation of the EU-H2020 CPSwarm project is to provide a workbench to explicitly manage behavior and emerging properties of swarms of CPS. This CPSwarm workbench CPSwarm tools will ease development and integration of complex herds of heterogeneous CPS that collaborate based on local policies and that exhibit a collective behavior capable of solving complex, industrial-driven, real-world problems. Related to this paper, the main goals of the project are to significantly improve the support to design complex, autonomous swarms of CPS; provide a self-contained, yet extensible library of reusable models for describing CPS; enable a sensible reduction in complexity and time of the CPS development workflow; and to establish reference patterns and tools for integration of CPS artefacts. Further details to the CPSwarm project are already presented in (Bagnato et al., 2017).

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 - the Modelling Tool and Modelling Library - that are strongly linked. The Modelling Tool also uses the Optimization Tool API to communicate with the Optimization Tool - part of the Algorithm Optimization Environment as shown in Figure 1.

The current CPSwarm Workbench includes tools provided by the contributing partners: Modelio as the modelling tool and FREVO as the optimization tool that are presented in the following subsections.

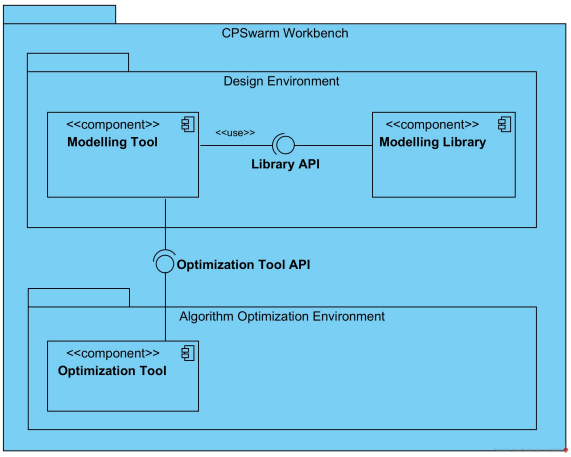

Figure 1: The Design Environment Architecture.

2.2 Modelio

Modelio[3] is a modelling environment developed by Softeam under a dual license, commercial and GPLv3 open-source. Modelio is based on Eclipse Rich Platform (RCP). It is used to create and manage models in various formats and notations. In addition, it provides many software features that compliments its modelling features such as: model transformation, code-to-model reverse engineering, and code generation. The CPSwarm Modelling Tool is built on top of the open source modelling environment (supporting UML2, BPMN2, MARTE, and SysML standards among others) named Modelio. Modelio Architecture is built around a central repository, around which a set of extension are defined cf. Figure 2.

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 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;

[3]https://www.modelio.org/

Figure 2: The Modelio architecture.

- 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 3, 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 project, only the SysML extension has been pick for its functionalities related to System modelling.

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.

2.3 FREVO

FREVO[4] - the FRamework for EVOlutionary design - is a software tool to design self-organizing systems. It is a general purpose framework that optimizes a generic representation of a given problem using evolutionary methods. Such an evolutionary approach requires a testbed that allows extensive and safe testing at low cost, for example a simulation of the target system with a model of the environment, the agents and their sensor/actor interface (Elmenreich et al., 2009). As it supports agent-based modelling, it is well suited to evolve the controllers for CPSs, when the controller is implemented as a generic, evolvable representation, e.g., an artificial neural network (ANN). An iterative heuristic search is applied to find an optimized configuration of the controller for a CPS with respect to a system level optimization measure, called fitness. The result is a controller that exhibits the local interaction rules to reach the desired global behaviour of the system. The controller can be evaluated on a large scale of parameters under predefined conditions. FREVO uses a modular approach, where the distinct steps of evolutionary design are split into different components. Its graphical user interface (GUI) simplifies the design process and offers statistics and graph generation for easy evaluation of the chosen design. The main purpose of FREVO is to support the optimization process as it guides through the individual steps of the evolutionary design, whereas it requires work by the software developer to implement the modelled problem details.

FREVO is implemented in the Java programming language and makes strong use of the object-oriented programming paradigm. FREVO's architecture is component-based (see Figure 4) and where each component implements a distinct feature of the evolutionary approach. The problem component defines the specifics of a CPS controller, the environment, and the fitness function. The representation component defines how the controller of the CPS is represented. The optimization component defines the method for finding the optimal candidate representation. The mutation and crossover operators are encapsulated with the representation, while population size and number of generations defined in the optimization component. FREVO provides a number of different implementations for each component type, for example there are different evolutionary optimization strategies available (Dittrich and Elmenreich, 2015; Zhevzhyk and Elmenreich, 2015).

The ranking component defines how the candidate representations are ordered based on their performance according to the fitness function. Each component is defined by an abstract class so that the interfaces between the components are well defined. Therefore, new components can easily be implemented requiring only the implementation of the core functionality. This step is guided by the built-in component generator that assists the software developer by generating the required code skeleton in the context of the class hierarchy.

[4]http://frevo.sourceforge.net/

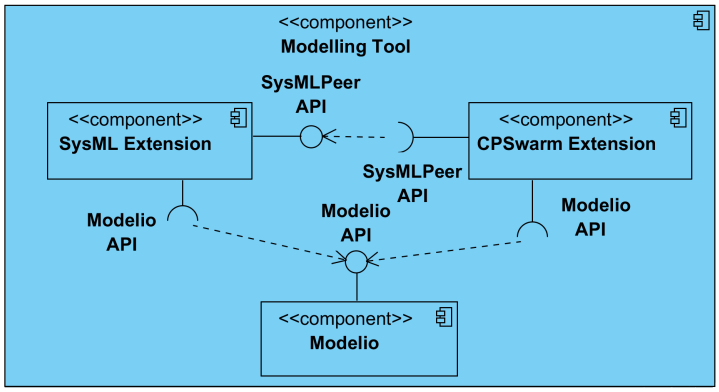

Figure 3: The modelling tool architecture.

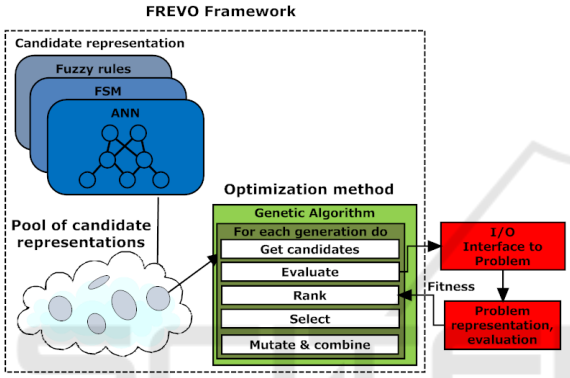

Figure 4: The FREVO architecture.

3 LIBRARIES TO MODEL CPS SWARMS

The overall idea is to have a 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. The models in the libraries are separated into three groups of libraries: swarm members, environments, and goals. These three libraries are used to model a problem as shown in Figure 5. Each model in the library (swarm member, environment, and goal) has the following mandatory parts:

Unique name:

- each model needs to be distinguishable from other models by name,
- each models name needs to be given in a way that is associated with the models functionality.

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).

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.

The following sections describe in more details of each library by referring to the EmergencyExit example already described in section 1.

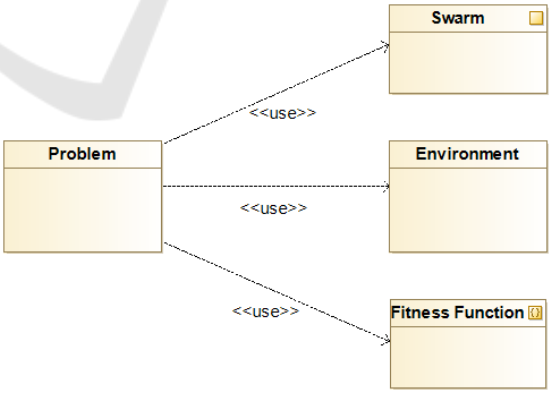

Figure 5: Problem Statement.

The modelling is performed by using SysML language and, more precisely, a Block Definition Diagram (BDD). The CPSwarm initial library on the EmergencyExit is stored on the Modelio Forge[5].

[5]https://forge.modelio.org/projects/cpswarm-modelio36/files

3.1 Swarm Member

The swarm member library describes a single CPS in the swarm. To better specify the characteristics of a CPS, we introduce several sub-libraries (together with examples) including:

- library "local memory": local status, e.g. the current x/y position, available energy, etc.
- library "behaviour": collecting data from sensor, performing calculations, sending data to actuators,
- library "physical aspects": sensors and actuators,
- library "security"
- library "human interaction"

Related to the EmergencyExit example the swarm member is a ground rover. The corresponding swarm member model architecture is depicted in Figure 6. Each rover has a local state (sub-library local memory) to express its current x/y position. Further, each rover needs sensors and actuators (sub-library physical aspects). The first sensor "POI Sensor" returns a vector, describing the path to the next point of interest (POI). The second sensor "Range Sensor" returns the map around the swarm member. The actuator is called "Locomotion" and is a motor that moves the swarm member to a given x/y coordinate. Finally, the behaviour (sub-library behaviour) is modelled as "EmergencyExit Behavior" and describes the EmergencyExit problem definition (see Section 1).

Finally a swarm can be modelled by a compostion of a set of swarm members (see Figure 7).

3.2 Environment

The environment library describes the environment in which the swarm of CPSs is acting. Several aspects express an environment, whereby the following ones are modelled (see Figure 8):

- 2D map of the environment,
- size of the environment,
- resolution,
- points of interests (emergency exits).

3.3 Fitness Function

The fitness function represents the goal of the specified CPS swarm. By maximizing this fitness function, the Algorithm Optimization Environment (cf. Section 4) can provide the best CPS swarm configuration. Figure 9 corresponds to the modelled goal of the EmergencyExit example. The goal of the swarm of CPSs is to find the two points of interest - the emergency exits - as fast as possible. Achieving this goal is expressed by the fitness value, computed by following equation:

$$-\sum_i \sqrt{(POI.x - LocalState_i.x)^2 + (POI.y - LocalState_i.y)^2} \quad (1)$$

In this fitness function the distances of all swarm members i to a point of interest are summed up and negated. Since the sum of the distances would be a cost function, the negation is used to express it as a fitness function. To modify the intended behavior it is sufficient to change the fitness function.

4 CPS SWARM MODELLING WITH THE CPSWARM WORKBENCH

The initial CPS models that are developed in Section 3 are the first parts of the CPS modelling library. As introduced in Figure 1 the modelling library represents just one component within the design environment architecture of the CPSwarm workbench. This library is used by the modelling tool to model swarms of CPSs. Via the optimization tool API these models are used as input for the optimization tool for further optimization. For the Modelling Tool, we use Modelio (introduced in Section 2.2). For the Optimization Tool, we use FREVO (introduced in Section 2.3). The integration of the tools together with the API are described in the following sections.

4.1 Integration of Modelio to the Architecture

Modelling is always a difficult task if you want to start from scratch. In this case, guidance is helpful. With Modelio we are able to generate a swarm template for the end users. 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 Figure 10.

Figure 11 shows the result of the CPS swarm template generation. The swarm template generator produces a set of initial diagrams that have been identified as necessary to completely model a CPS swarm. As described in the previous section this includes the swarm and its member, the environment, and the goal or fitness function.

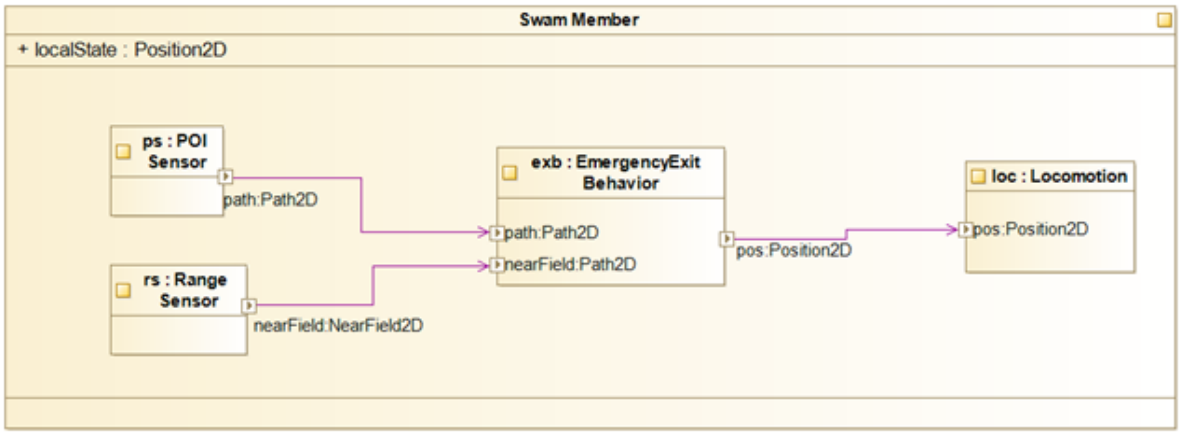

Figure 6: The swarm member architecture.

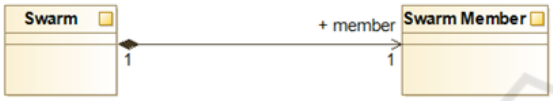

Figure 7: The swarm architecture.

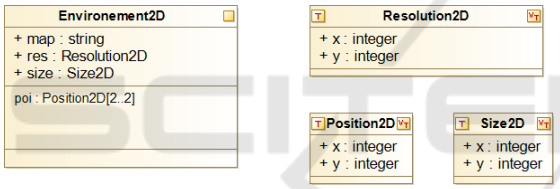

Figure 8: The environment model.

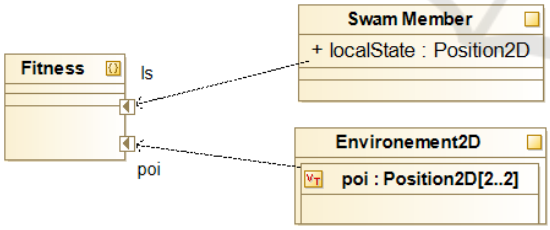

Figure 9: Modelling of the Fitness Function.

The modeler can modify their 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. Environment Definition and Problem Statement mainly uses UML Class Diagram and related concepts. Swarm Architecture and Swarm Member Architecture are specified by using, respectively, SysML Block Diagram Defition (BDD) and Internal

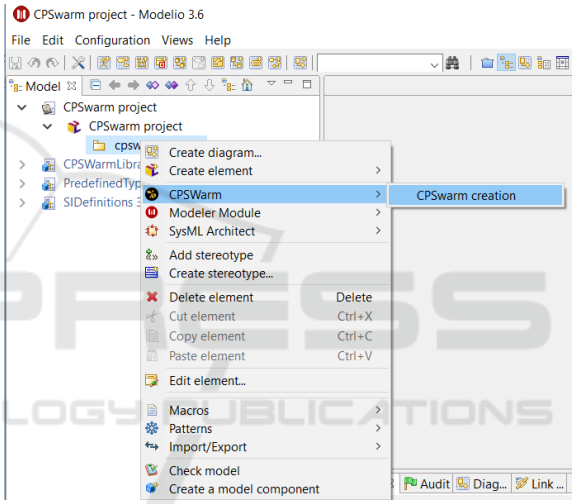

Figure 10: Creating a new swarm model.

Block Diagram (IBD) elements. SysML Parametric Diagram and MathML are both used for Goal Definition. Figure 12 is showing the specific modelling environment for the Swarm Member Architecture Diagram. The diagram is meant to show the high-level structure of the swarm member.

4.2 Design of the Optimization Tool API

The optimization tool API passes configuration files from the modelling tool to FREVO. This configuration consists of two files: The problem description and the optimization parameters. 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 parameters are available in a .xml file. They describe the setup of FREVO. These parameters

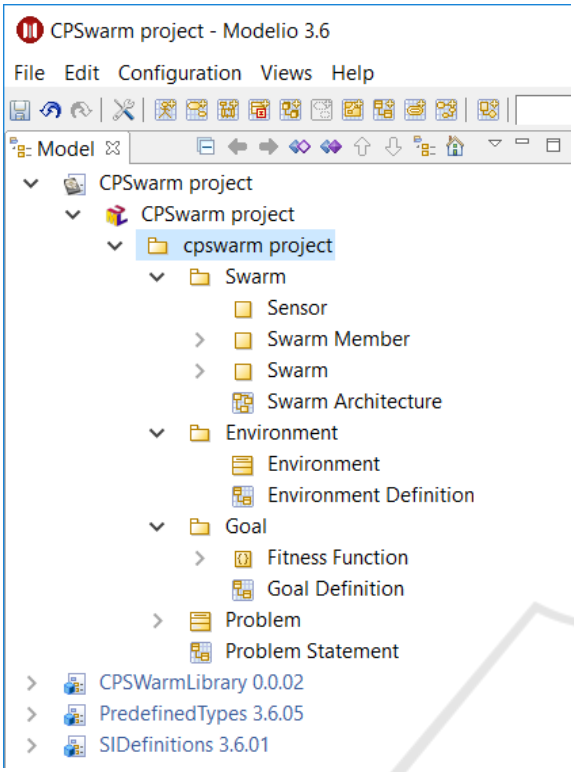

Figure 11: New swarm result.

include configuration, properties, and requirements. Both parts are exported from the Modelio (see Figure 13). Once these files are passed to FREVO, a recompilation of FREVO is required before starting the optimization process.

4.3 Integration of FREVO to the Architecture

FREVO requires configuration and code of a new problem to be implemented via a .java and .xml file. Both files have been already generated in the Modelling Tool, although the implementation part needs to be extended. 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 FREVO or by calling an external simulator. Therefore, a model of the environment and the CPS capabilities, i.e. its sensors and actuator behavior 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). Additionally, 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 FREVO.

After launching FREVO, following settings for the simulation need to done:

- Selected Problem: EmergencyExit
- Selected Method: CEA2D
- Selected Representation: FullyMeshedNet
- Selected Ranking: Absolute Ranking

The method CEA2D is a cellular evolutionary algorithm (EA) that arranges all candidates on a 2D torus surface. Genetic operations are performed in a local context. It features better diversity with slower convergence compared to a standard EA (Dittrich and Elmenreich, 2015). For the representation, we select FullyMeshedNet. It is a recurrent, fully meshed ANN with one hidden layer. During evolution, the biases of the neurons as well as the connection weights are changed. Adaptive mutations are also supported. The ranking algorithm sorts candidate representations based on their performance, i.e. their fitness value. For this example absolute ranking is selected. A ranking algorithm that sorts candidates by the fitness value returned from the problem component. It supports multi-threading to decrease the time needed for optimization.

After a click on the play button the optimization runs until the selected termination criterion has been fulfilled. Figure 14 shows the simulation result of the EmergencyExit example - the fitness value is maximized towards 0. FREVO has several more features: The entire simulation run is saved automatically. Nevertheless, you could click the ”Pause” button and ”Save” the simulation at a specific generation. This is not the final evolved one, but a specific intermediate state of evolution. Moreover, when you click the ”Replay” button, to get more information on the evolutionary produced results. FREVO allows also to evaluate the evolved results with a simple graphical representation of the simulation.

5 EVALUATION

The evolved result from FREVO provides the behavior of the swarm members for the given problem. These evolved results come as a black box algorithm, so it is necessary to evaluate the performance of the algorithm under different environments. FREVO allows to evaluate the evolved results of the related problem definition with different variations, including changing the number of candidates, the size of the 2D

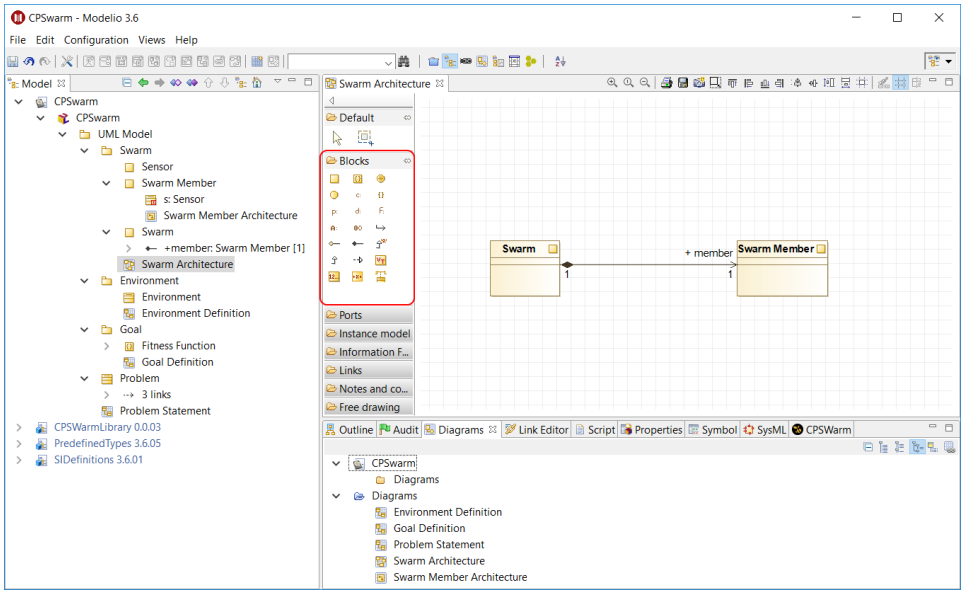

Figure 12: Swarm Member Architecture Modelling Elements.

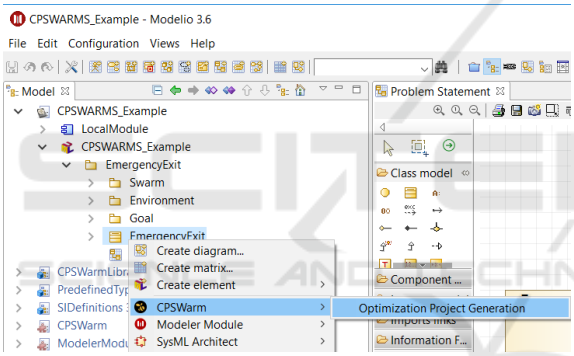

Figure 13: Exporting the files for the optimization tool from the design environment.

environment and to make a statistical evaluation of the behavior using Monte-Carlo simulation over different random seeds.

In the EmergencyExit example, parameters like success rate or the average or worst case time until the simulated robots have left the scene through the exits can be used as a numeric assessment of the algorithm. Furthermore, a graphical depiction of the simulation can be helpful to evaluate the algorithm. The EmergencyExit problem in FREVO has been implemented with a very basic simulation interface showing the movement of agents on a grid (see Figure 15). The EmergencyExit problem was designed as test problem for the integration of two open source tools in our CPSwarm workbench. Due to the relatively simple setup, the solutions to the problem were always successful in leaving the area.

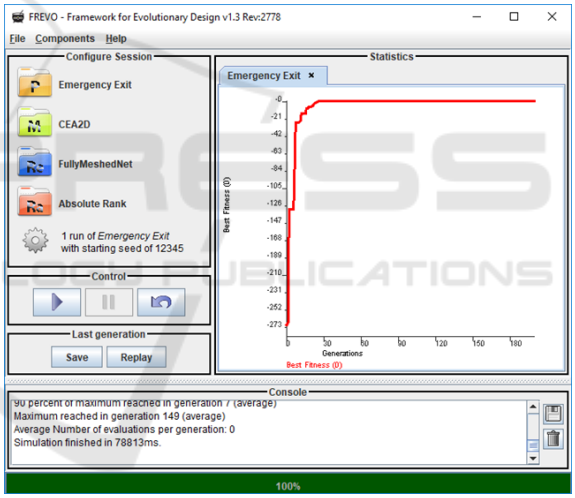

Figure 14: Simulation Settings of FREVO.

As mentioned in Section 4.3, it is also possible to use FREVO with an external simulator - in this case the Stage[6] simulator - with FREVO, as shown in Figure 16.

6 CONCLUSION

This paper has demonstrated the functionality of an initial version of the CPSwarm workbench. The main focus was put on the modeling and optimization part. Therefore, the workbench integrated two open-source tools, Modelio and FREVO. As a case study, a pro-

[6]http://playerstage.sourceforge.net/index.php?src=stage

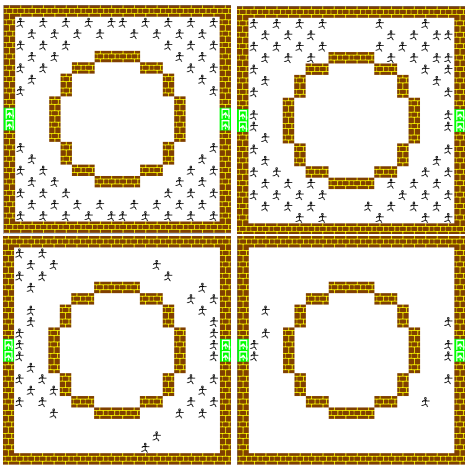

Figure 15: Replay of a simulation the EmergencyExit example by FREVO.

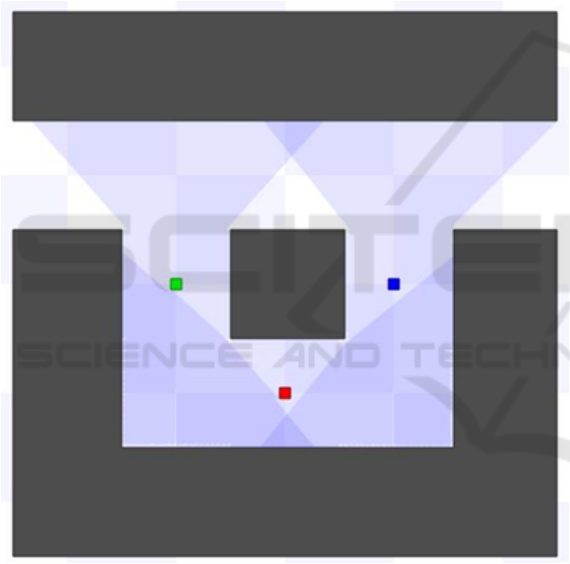

Figure 16: Simulation of the EmergencyExit example with Stage/ROS.

blem named emergency exit was chosen. In this example a swarm of robots needs to find an exit in a unmapped environment and leave this room through the exit as soon as possible. A possible scenario matching this problem could be fire or collapsing buildings. In the case study, the problem was first modeled using Modelio and then exported to FREVO for optimization. FREVO was then used to generate a possible solution by applying an evolutionary search algorithm in order to optimize the weights for an ANN. In this work solution had been evaluated using simulation. Future work will include an extension of the workbench towards deploying the generated solution on real systems, which will involve code generation for embedded target systems.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive comments on the submission version of this paper. The research leading to these results has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No 731946, CPSwarm Project.

REFERENCES

Auer, C., Wüchner, P., and de Meer, H. (2008). A method to derive local interaction strategies for improving cooperation in self-organizing systems. In *Proceedings of the Third International Workshop on Self-Organizing Systems*, Vienna, Austria.

Bagnato, A., Bíró, R. K., Bonino, D., Pastrone, C., Elmenreich, W., Reiners, R., Schranz, M., and Arnautovic, E. (2017). Designing swarms of cyber-physical systems: The H2020 cpswarm project: Invited paper. In *Proceedings of the Computing Frontiers Conference*, CF'17, pages 305–312, New York, NY, USA. ACM.

Bagnato, A., Brosse, E., Quadri, I., and Sadovykh, A. (2015). Into-cps: An integrated tool chain for comprehensive model-based design of cyber-physical systems. *Génie Logiciel*, pages 31–35.

Bagnato, A., Brosse, E., Quadri, I., and Sadovykh, A. (2016). SysML for modeling co-simulation orchestration over FMI: the INTO-CPS approach. *Ada User Journal*, 37(4):215–218.

Dittrich, T. and Elmenreich, W. (2015). Comparison of a spatially-structured cellular evolutionary algorithm to an evolutionary algorithm with panmictic population. In *Proceedings of the 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES'15)*, pages 145–149, Ancona, Italy.

Elmenreich, W. and de Meer, H. (2008). Self-organizing networked systems for technical applications: A discussion on open issues. In K.A. Hummel, J. S., editor, *Proceedings of the Third International Workshop on Self-Organizing Systems*, pages 1–9. Springer Verlag.

Elmenreich, W., D'Souza, R., Bettstetter, C., and de Meer, H. (2009). A survey of models and design methods for self-organizing networked systems. In *Proceedings of the Fourth International Workshop on Self-Organizing Systems*, volume LNCS 5918, pages 37–49. Springer Verlag.

Larsen, P. G., Fitzgerald, J. S., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., and Sadovykh, A. (2016). Integrated tool chain for model-based design of cyber-physical systems: The INTO-CPS project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS, CPS Data 2016, Vienna, Austria, April 11, 2016*, pages 1–6.

Lee, E. A. (2008). Cyber physical systems: Design challenges. In *11th IEEE International Symposium on Object*

Oriented Real-Time Distributed Computing (ISORC), pages 363–369.

Martins, P. and McCann, J. (2017). Chapter 7 - network-wide programming challenges in cyber-physical systems. In Song, H., Rawat, D. B., Jeschke, S., and Brecher, C., editors, *Cyber-Physical Systems*, Intelligent Data-Centric Systems, pages 103 – 113. Academic Press, Boston.

Sobe, A., Fehérvári, I., and Elmenreich, W. (2012). FREVO: A tool for evolving and evaluating self-organizing systems. In *Proceedings of the 1st International Workshop on Evaluation for Self-Adaptive and Self-Organizing Systems*, Lyon, France.

Zhevzhyk, S. and Elmenreich, W. (2015). Comparison of metaheuristic algorithms for evolving a neural controller for an autonomous robot. *Transactions on Machine Learning and Artificial Intelligence*, 2(6):62–76.