D6.4 – FINAL CPS SYSTEM DESIGN OPTIMIZATION AND FITNESS FUNCTION DESIGN GUIDELINES

Deliverable ID	**D6.4**
Deliverable Title	**Final CPS System Design Optimization and Fitness Function Design Guidelines**
Work Package	**WP6**
Dissemination Level	**PUBLIC**
Version	**2.0**
Date	**07-12-2019**
Status	**Final**
Lead Editor	**Davide Conzon (LINKS)**
Main Contributors	**Midhat Jdeed (UNI-KLU), Arthur Pitman (UNI-KLU), 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	29-04-2019	Davide Conzon (LINKS)	Initial TOC
0.2	14-05-2019	Davide Conzon (LINKS)	Integrated Fitness Function Design in CPSwarm and Simulator API sections
0.3	01-06-2019	Davide Conzon (LINKS)	Integrated UNI-KLU contributions
0.4	13-06-2019	Davide Conzon (LINKS)	Integrated the fitness function example
0.5	30-06-2019	Davide Conzon (LINKS)	Integrated SOFTEAM and UNIKLU contributions, ready for internal review
1.0	08-07-2019	Davide Conzon (LINKS)	Integrated the revisions from the reviewers
1.1	08-08-2019	Davide Conzon (LINKS)	New version to be used to collect update
1.2	31-09-2019	Arthur Pitman (UNI-KLU)	Described parameters optimization
1.3	31-10-2019	Etienne Brosse (SOFTEAM)	Described fitness function design
1.4	02-12-2019	Davide Conzon (LINKS)	Integrated the final contributions of the partners
1.5	03-12-2019	Davide Conzon (LINKS)	Minor improvements
1.6	03-12-2019	Davide Conzon (LINKS)	Final version ready for internal review
2.0	08-12-2019	Davide Conzon (LINKS)	Integrated revisions from internal reviewers

Internal Review History

Review Date	Reviewer	Summary of Comments
05-07-2019 – v0.5	Renè Reiners (FIT)	Approved with minor comments.
05-07-2019 – v.0.5	Etienne Brosse (SOFTEAM)	Approved with minor comments.
04-12-2019 – v1.6	Renè Reiners (FIT)	Approved with minor comments.
06-12-2019 – v1.6	Etienne Brosse (SOFTEAM)	Approved with minor comments.

Table of contents

Contents

1 Executive summary

Deliverable *D6.4 - Final CPS System Design Optimization and Fitness function Design Guidelines* updates the concepts presented in *D6.3 - Initial CPS System Design Optimization and Fitness function Design Guidelines* about the optimization of Cyber Physical Systems (CPSs) using heuristic search approaches and methods for assessing their performance. Special emphasis is given to the development of fitness functions that guide the optimization process. First, this deliverable presents the method used in CPSwarm to support the optimization of swarm algorithms. Then, it outlines the best practices for designing fitness functions and describes how this is done in the CPSwarm Workbench. Finally, a case study on the logistics scenario is introduced to examine the effectiveness of the technique.

2 Introduction

2.1 Scope

This deliverable considers the design of optimal swarms of CPSs and the corresponding fitness function used in optimization process and how this can be accomplished using the CPSwarm Workbench. This deliverable presents the interface between the components of the Simulation and Optimization environment, i.e., the Simulation and Optimization Orchestrator (SOO), the Optimization Tool (OT) and the Simulation Managers (SMs) that integrate the external simulators in the CPSwarm Workbench. The implementation of the SMs is covered by the deliverables *D6.5/D6.6/D6.7 - Initial/Updated/Final integration of external simulators*.

2.2 Document organization

The rest of this deliverable is structured as follows: Section 3 recaps the concepts presented in D6.3 about CPS design optimization and then presents the integration of the Optimization Tool (i.e., FRamework for EVOlutionary design - FREVO) in the CPSwarm Workbench, describing the Application Programming Interfaces (API) and the optimization workflow defined. In Section 4 the authors describe the final guidelines for fitness function design and present their application in the CPSwarm Workbench. Finally, Section 5 concludes this deliverable.

2.3 Related documents

ID	Title	Reference	Version	Date
D5.2	Initial CPSwarm Modelling Tool	D5.2	1.0	30-09-2017
D6.1	Initial Simulation Environment	D6.1	1.0	05-10-2017
D6.5	Initial Integration of External Simulators	D6.5	1.0	30-06-2018
D3.2	Updated System Architecture & Design Specification	D3.2	1.0	30-06-2018
D6.3	Final CPS System design optimization and Fitness function design guidelines	D6.3	1.0	30-06-2018
D6.2	Final Simulation Environment	D6.2	1.0	31-04-2019
D6.6	Updated Integration of External Simulators	D6.6	1.0	31-04-2019
D6.7	Final Integration of External Simulators	D6.7	1.0	31-12-2019

3 CPS design optimization

As documented in D6.3, the design of CPSs may lead to a tremendous increase in complexity. CPS development remains a complex and error-prone task, often requiring a collection of separate tools, to follow a CPS design cycle including modelling, simulation, optimization and deployment. Moreover, interactions amongst CPSs might lead to new behaviours and emerging properties, often with unpredictable results. Rather than being an unwanted byproduct, these interactions can become an advantage if explicitly managed since early design stages. CPSwarm tackles this challenge by proposing a new science of system integration and tools to support the engineering of CPS swarms. The purpose of CPSwarm tools is to ease development and integration of complex herds of heterogeneous CPSs that collaborate based on local policies and that exhibit a collective behaviour capable of solving complex, industry-driven, real-world problems.

The CPSwarm approach, using the Simulation and Optimization environment in the Workbench, allows the performance of a swarm solution to be evaluated. This tool includes mainly three components: the Simulation and Optimization Orchestrator (SOO) that coordinates all the simulation and optimization tasks; a set of SMs that interfaces with heterogeneous Simulation Tools (STs) with common Application Programming Interfaces (APIs); and an Optimization Tool, such as FREVO, to perform the optimization processes. This environment can be used either to simulate the behaviour of a designed swarm solution in a ST using its Graphical User Interface (GUI) to evaluate its behaviour or to optimize the controller algorithm/module using evolutionary design methodologies.

3.1 Parameter Optimization

Within the design of swarm systems, optimization may take many forms ranging from the creation of complete controllers implemented using neural networks to the fine-tuning of algorithms. In the CPSwarm project, the partners focus on parameter optimization which can adjust the behaviour of any algorithm by modifying its parameters. Compared with the development of controllers based on neural networks explored in *D6.3 - Initial CPS System Design Optimization and Fitness function Design Guidelines*, parameter optimization requires fewer computationally expensive simulation runs. Combined with effective visualization and easy deployment to real hardware, the authors have considered that this approach can deliver tangible results to experts designing CPSs.

At first glance, designers could search the entire search space by examining every possible combination of parameters, however this is almost always too computationally expensive. More commonly, in an informed optimization task, a heuristic is used to guide the optimization process towards the optimal solution. However, in multi robot or swarm systems operating in diverse environments, such heuristics may be hard or even impossible to design. Evolutionary computation, based on the principles of selection, mutation and cross-over, takes a slightly different approach, manipulating parameters according to their performance measured by a "fitness" function. In many instances this approach can produce satisfactory if not optimal solutions without intimate knowledge of the behaviour itself.

3.2 FREVO as an Optimization Tool

In the initial CPSwarm system design optimization, presented in D6.3, FREVO has been introduced as an optimization tool. The first key element of the optimization process is the choice of representation that describes the structure of a possible solution. Secondly, operators that modify, i.e., mutate or cross-over, the representations must be defined. Thirdly, an optimization method must be specified to manage the selection of representations. Currently FREVO provides an implementation of the NNGA method [1]. It begins by creating n_{pop} candidate solutions. In each of the n_{gen} generations, the solutions are evaluated and ranked according to their performance. Successful solutions, i.e., those with high fitness values, are carried to the next generation as elite, or are crossed or mutated to produce new controllers. In addition, a small proportion of entirely new random candidates is introduced with the intention of maintaining diversity in the population.

Within the CPSwarm project, FREVO delegates the problem specification to the simulation run by the SMs and relies instead on the fitness values calculated at the end of each simulation to guide the optimization. FREVO-XMPP was developed as a wrapper supporting XMPP communication to transfer candidate solutions to SMs and await fitness values. By integrating a work queue, FREVO is able to efficiently balance loads overs a large set of SMs and robustly handle failures that may occur during simulation.

3.3 Integration of FREVO in the CPSwarm Workbench

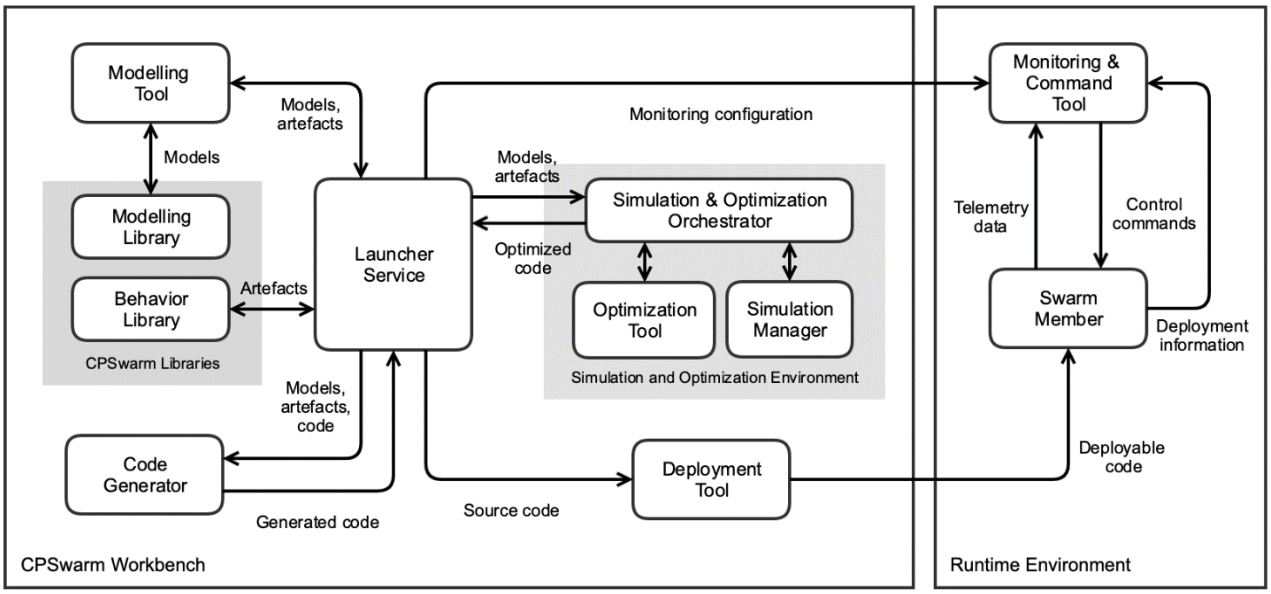

Figure 1 - CPSwarm reference architecture.

The OT and the external simulators are integrated in the CPSwarm Workbench architecture (Figure 1), using a broker-based distributed approach, designed in Tasks 6.1 and 6.2. Deliverable *D6.2 – Final Simulation Environment* has introduced the final version of the Simulation and Optimization Environment architecture. This deliverable will only briefly describe the final version of the architecture, instead it provides a full description of the new APIs for the interaction between the OT and the simulation components, designed to enhance the scalability and reliability of the solution compared to the ones presented in D6.2.

During the CPSwarm project, two different versions of the Simulation and Optimization Environment architecture have been designed: the initial one, described in *D6.1 – Initial Simulation Environment*, which, subjected to a performance analysis, offered limited performances due to an inefficient discovery mechanism and the large number of messages required between the OT and the simulators during optimization process. These issues have been addressed in the final version of the architecture first introduced in *D6.5 Initial Integration of External Simulators* and in [2] and fully described in D6.2, where also some new features that aim to enhance the scalability and the rapid deployment of the solution were also introduced.

As already stated, the architecture is composed of three components: the SOO, which coordinates all the operations of simulation and optimization, the OT, which is the component responsible for developing candidates solutions as explained in Section 3.2 and finally a set of Simulation Tools, e.g., Stage[1] and Gazebo[2] based on the Robot Operating System (ROS) distributed on several machines – namely Simulation Servers (SSs) - each one wrapped by a SM, allowing interaction via a standard interface. D6.2 has also introduced a set of features to improve the deployment and the scalability of the architecture, applying two new technologies: the software components of the Simulation and Optimization Environment have been refactored to run in one

[1] https://github.com/rtv/Stage
[2] http://gazebosim.org

container environment (e.g. Docker[3]). This allows several instances of ST to be run on each SS. Furthermore, the containerized components can be deployed and orchestrated dynamically using a rapid deployment and orchestration tool (e.g., Kubernetes[4]), which has been integrated in the SOO, allowing the user to simply set up the required set of STs to execute the desired simulation and optimization tasks.

A prototype of this architecture has been already produced using Docker, Kubernetes and the eXtensible Messaging and Presence Protocol (XMPP) and all its native security features [3] for the communication among the components (please refer to D6.2 for its description).

The next subsection will introduce the new Simulator API defined after a series tests done on the final Simulation and Optimization Environment architecture, using the prototype presented in D6.2. The main goal of this refactoring is to improve the reliability and scalability of the system. The API are introduced in this deliverable, presenting the relative optimization and simulation workflow. Their implementation is out of scope of this deliverable and will be presented in *D6.7 - Final Integration of external Simulators* due at M36.

3.3.1 Refactored Simulator API

Compared to the ones described in D6.2, these APIs have three main objectives:

- Remove redundant fields: the description field has been removed as it was it unused.
- Reduce the variety of messages: ***OptimizationStarted***, ***OptimizationCancelled*** and ***OptimizationProgress*** have been merged into the ***OptimizationStatus*** message.
- Define a way to restore an optimization process if something goes wrong during an optimization process.

Table 1 will present the messages defined for these APIs. The following details are provided for each API:

- Description of the API.
- Components using this API.
- Type of communication used (i.e., file transfer, text message).
- Data included.

Table 1 - Refactored Simulator API

API	Description	Components involved	Type of communication	Data included
SimulatorConfiguration	Used to configure the selected STs.	Sent by the SOO to all the selected SMs.	File transfer.	ZIP file including: • CPS models. • Environment models. • Other configurations (must include a SCID to be used by selected SMs to set as new presence status according to the following workflow explaination).
SimulatorConfigured	Replies to the configuration files sent from the SOO, indicating if the ST has been	Sent by a SM to the SOO.	Text message.	Fields: • OID: The IDentifier (ID) of the optimization process. • Type: The type of the message (fixed value: SimulatorConfigured). • Success: Boolean value.

[3] ttps://www.docker.com/

[4] https://kubernetes.io/it/

	successfully configured or not.			
StartOptimization	Used to start an optimization task.	Sent by the SOO to the OT.	Text message.	Fields: • OID: a unique ID assigned by the SOO that allows the optimization process to be tracked. Typically, it contains a refence to the corresponding simulation environment as well as the package to be optimized, for example cpswarm_sar. • Type: The type of the message (fixed value: StartOptimization). • Configuration: a JSON string containing the configuration of the OT. • SCID: a simulation configuration ID, which informs the OT, which SMs can participate in the optimization process. The OT monitors SM presence information and will run simulations on any SM configured for this SCID. This allows the SOO to configure and start SMs during an optimization according to available resources and the desired system performance.
GetOptimizationStatus	Used to get the status of a running optimization, specified by OID.	Sent by the SOO to the OT.	Text message.	Fields: • OID: the ID of the optimization process to enquire about. • Type: The type of the message (fixed value: GetOptimizationStatus)
GetOptimizationState	Used to ask a dump of the state of a running optimization, specified by OID, including configuration and list of candidates of the current generation. This state will be used to restart an optimization in case of errors.	Sent by the SOO to the OT.	Text message.	Fields: • OID: the ID of the optimization process to enquire about. • Type: The type of the message (fixed value: GetOptimizationState).
OptimizationState	Used to send a dump of the state of the current optimization, to be used to restart it in case of errors.	Sent by the OT to the SOO, replying to GetOptimizationState and by the SOO to the OT when an optimization process needs to be restarted	File transfer.	Zip file including: • The configuration of the OT. • The list of candidates of the current generation.
OptimizationToolConfigured	Replies to the configuration files in OptimizationState sent from the SOO, indicating if the OT has been successfully configured or not before restarting the	Sent by OT to the SOO.	Text message.	Fields: • OID: The IDentifier (ID) of the optimization process. • Type: The type of the message (fixed value: OptimizationToolConfigured). • Success: Boolean value

	previous optimization.			
CancelOptimization	Used to cancel a running optimization, specified by OID	Sent by the SOO to the OT.	Text message.	Fields: • OID: the ID of the optimization process to cancel. • Type: The type of the message (fixed value: CancelOptimization).
OptimizationStatus	Used to reply to the StartOptimization, GetOptimizationStatus and CancelOptimization messages.	Sent by OT to SOO.	Text message.	Fields: • OID: the ID of the optimization process. • Type: The type of the message (fixed value: OptimizationStatus). • Status: The status of the optimization process: ○ ErrorBadConfiguration: Optimization couldn't be started due to bad configuration. ○ Running: Optimization is running ○ ErrorOptimizationFailed: Optimization stopped due to an unknown error. ○ Stopped: Optimization stopped by request from SOO ○ Complete: Optimization completed successfully • Progress: Progress of the optimization process in. • BestFitnessValue: The current fitness value. • BestController: The current best controller found in optimization.
RunSimulation	Instructs the SM to run a simulation using the specified controller. It can be sent by the OT during optimization or by the SOO to request a simple simulation.	Sent by the SOO or by the OT to the SM.	Text message.	Fields: • OID: the ID of the optimization process starting the simulation. • Type: The type of the message (fixed value: RunSimulation). • SID: ID of the simulation, assigned by the OT.
SimulationResult	Returns the result of a simulation performed for optimization.	Sent by a SM to the OT during optimization.	Text message.	Fields: • OID: The ID of the optimizationprocess. • Type: The type of the message (fixed value: SimulationResult). • SID: ID of the single simulation. • FitnessValue: The fitness of the controller, calculated by the fitness function. • Success: Boolean value.

3.3.2 Optimization workflow

The SOO can perform an optimization using the OT, where each controller candidate is simulated in a SS, as shown in Figure 2.

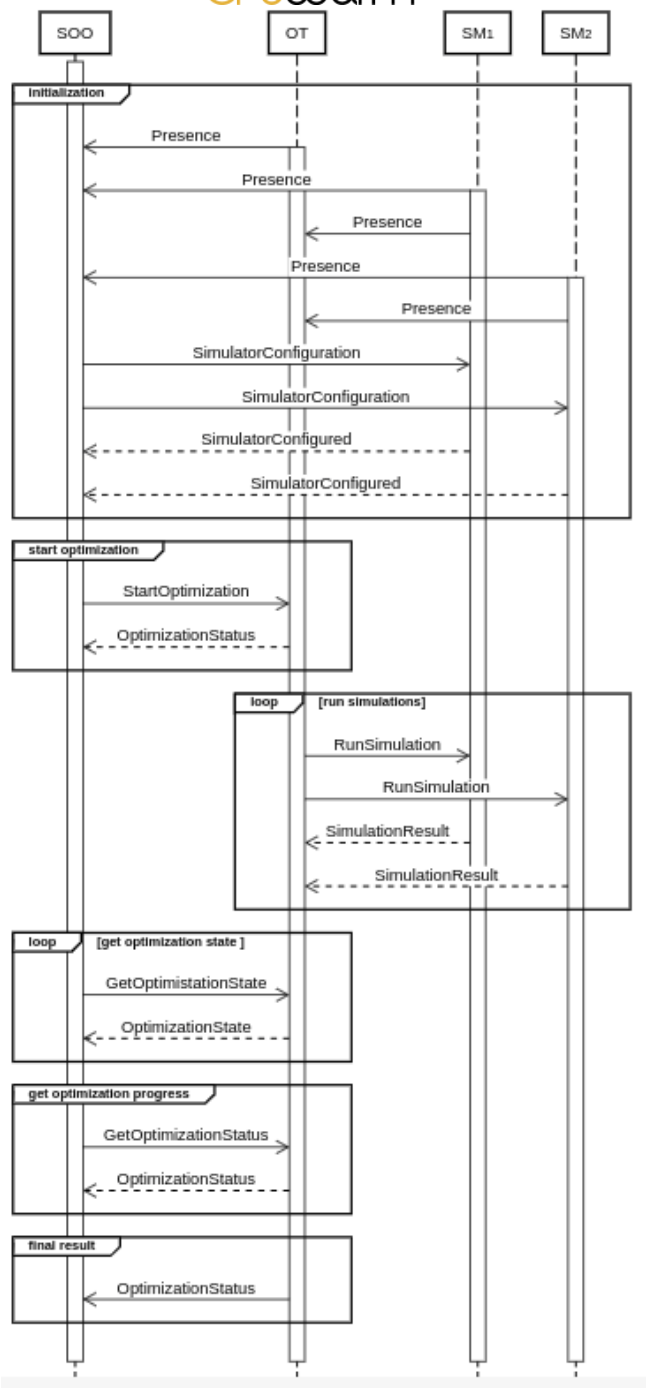

Figure 2 - Optimization workflow

Figure 2 shows the messages' flow among the SOO, the OT, and two exemplary SMs, during an optimization process. In the initialization phase, all components announce their availability by broadcasting presence information. The SOO and the OT collect this information to create a list of available SMs and their capabilities to be used, indeed the list is needed both by the SOO and the OT, because they need to select the SMs to use. Similarly, the OT's presence informs the SOO that it is ready to perform optimization tasks. When the user starts the optimization, the SOO evaluates the available SMs, selecting the ones that fulfil the requirements. Then, it transmits configuration files, to them, to setup the simulation (including, the CPS and environment models received by the modelling tool, the maximum number of simulation steps to be performed, etc.). The SOO assigns to all the configured SMs a unique Simulation Configuration IDentifier (SCID) that they set as status in their presences. Once all the SMs have confirmed to have been configured with a

SimulatorConfigured message, the SOO sends a ***StartOptimization*** message to the OT with the SCID to be used, which replies with an ***OptimizationStatus*** message including a unique Optimization IDentifier (OID), valid for the whole optimization process. The OT selects all the SMs that has sent a presence with the SCID indicated by the SOO, to be used for the optimization and, then, it begins the optimization, sending a sequence of ***RunSimulation*** messages to SMs, including the candidate controller to be evaluated. The SMs use the corresponding STs to evaluate the controllers and after having calculated the fitness score of the candidate, they send it to the OT, through a ***SimulationResult*** message. Throughout the optimization process, the SOO may request the progress of the optimization process intermittently or even cancel it by sending the OT a ***GetOptimizationStatus*** or ***CancelOptimization*** message respectively, receiving in response an ***OptimizationStatus*** message indicating the status of the optimization. Furthermore, periodically, the SOO sends to the OT a ***GetOptimizationState*** message to ask a backup of the current optimization state, when the OT receives this message, it sends back to the SOO a file containing the current configuration and the list of the candidates of the current generation, which can be used to restart the optimization (see Section 0 for details), Once the optimization process completed, the OT sends a final ***OptimizationStatus*** message to the SOO, which includes the optimized candidate.

3.3.3 Simulation workflow

The SOO can also be used to send a specific controller candidate to a SM, for more in depth analysis. In this case, the OT is not involved, the SOO and SM communicate directly (see Figure 3) and the behaviour is the same indicated in the deliverable D6.2. This allows a controller optimized by the OT to be evaluated more thoroughly, e.g., through visual replay using the ST Graphical User Interface (for visual replay, the selected ST must run on a machine directly accessible for the user, so that he/she can see the ST's GUI. In this scenario, the SOO uses the collected presences to select one SM among the ones suitable for that simulation. Then, the SOO sends to it, the required files and, after having received the ***SimulatorConfigured*** message, then sends to it, the candidate controller to be replayed.

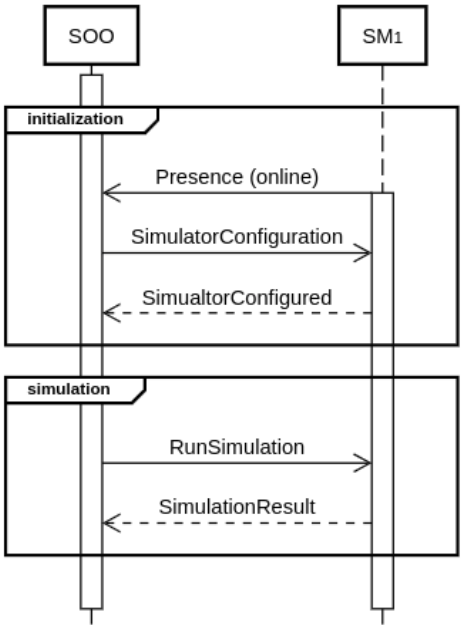

Figure 3 - The messaging sequence when simulating a specific CPS controller.

3.3.4 Error recovery workflow

The introduction of this new version of the API has allowed to introduce a set of new features relative to the reliability of the system and error recovery. This is important because the optimization process, also with the use of the distributed architecture introduced by CPSwarm is a long process and the ability to recover to errors without the need to restart from the beginning is important.

For this reason, the partners have defined API to recover from possible failures in all the components of the system. These APIs are detailed in the following subsections.

3.3.4.1 SOO error recovery

If during an optimization the SOO fails or goes offline, the OT is informed in real-time through the presences. The optimization process is not stopped, because the OT communicates directly with the SMs, but if, when the process finishes the SOO is still offline, the OT instead to the send immediately the ***OptimizationStatus*** message to the SOO, it stores the result internally and sends it only when the SOO is back online.

3.3.4.2 OT error recovery

To address the possibility that the OT fails or goes offline during an optimization process, the partners have defined a mechanism to save and restore the optimization state. As shown in Figure 4 and outlined before, the SOO periodically sends the ***GetOptimizationState*** (identified by its OID) to the OT and store it locally. When the SOO receives a presence that indicate that the OT has failed, the SOO waits until the OT is back online. Then, it sends a ***GetOptimizationStatus***, if the status indicates that the optimization is still ongoing (i.e., if there has been only a loss of connection between the SOO and the OT) the SOO continues to wait the conclusion of the process; instead, if the status indicates that the optimization is not ongoing, the SOO sends back to the OT the last dump file that it has received using the file transfer and when the OT sends back a ***OptimizationToolConfigured*** it send a new ***StartOptimization*** to the OT. In this way, the OT can start from the status previously reached (the last generation of candidate created) and not completely from scratch, largely reducing the time required to complete an optimization process, if some problem occurs.

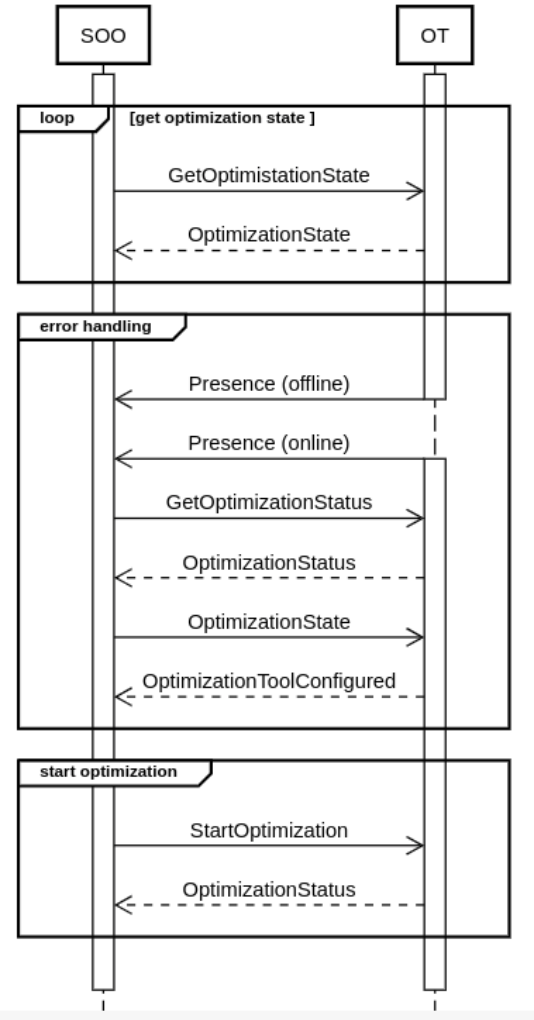

Figure 4 - OT error recovery

3.3.4.3 SM error recovery

If one SM fails or goes offline during the optimization process and it is one of those used by the OT, when the relative offline presence is received, the OT can automatically stop to use it until the SM goes online again.

To handle better this type of situation, the OT implements also a mechanism that allow the OT to add, during the optimization process, new SMs to the list of the ones to be used, just when they go online. This feature can be used to replace another SM that has failed, but also to scale on ongoing optimization, without stopping the process.

The entire flow is shown in Figure 5.

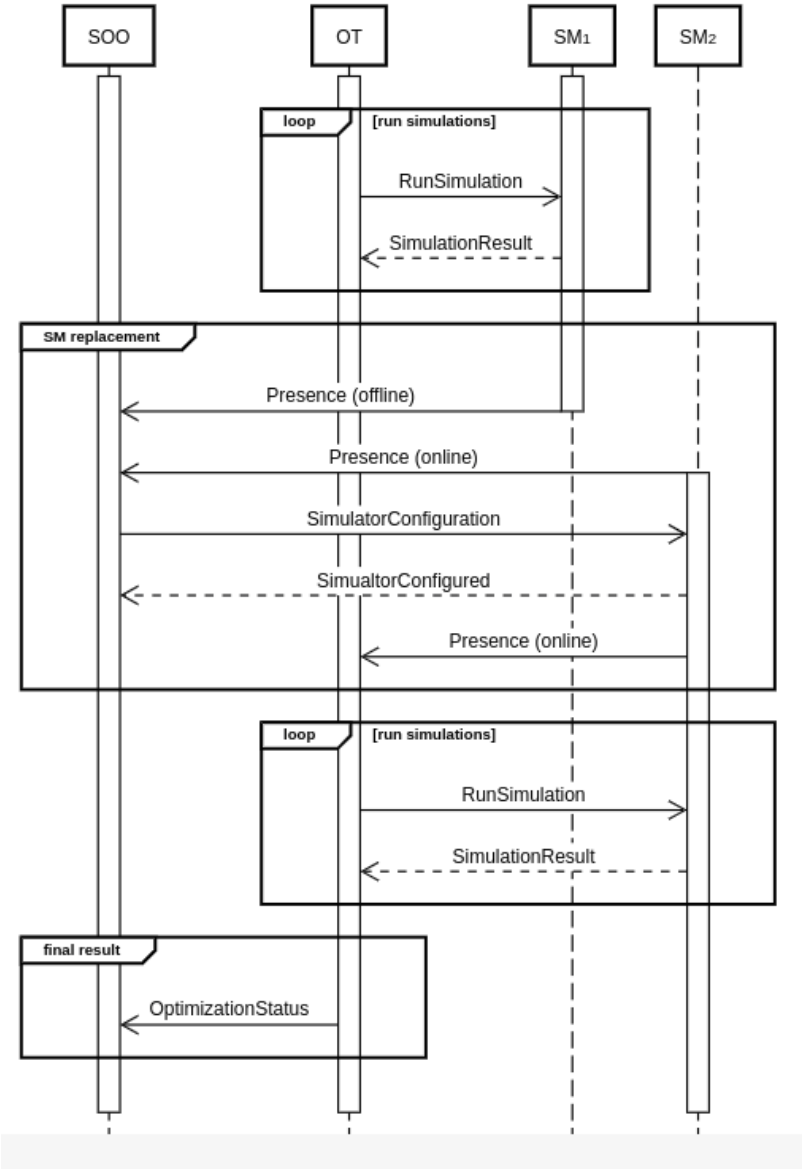

Figure 5 - SM error recovery

4 Fitness function design

According to the initial CPS system design optimization and fitness function design guidelines, presented in D6.3, a fitness function represents the desired behaviour of a swarm of CPSs. The best performance of the design optimization can be achieved by maximizing that fitness function. Such a function is evaluating the closest solution for the optimum on for a problem. While various names are used in the literature, such as fitness function, cost function or utility function, the function seeks to guide the optimization process towards a satisfactory if not optimal solution. While the function itself is highly problem-specific and thus there are no straightforward rules for its design, many studies in the field of evolutionary optimization have considered generic methods for fitness function design. In general, these methods may be categorized into a three-dimensional fitness space [4] [5]:

1. Functional vs. behavioural: A functional fitness is based on components that directly measure the way in which the system functions. A behavioural one rewards the system for displaying a given behaviour.

2. Global vs. local: Global fitness rewards the system based on information that is available to an external observer, while the local one is restricted to information available to a single component.

3. Explicit vs. implicit: An explicit function rewards the way in which a certain goal is achieved), while implicit fitness is focused on how much the goal is reached (e.g. a distance). Implicit functions are also extensively used in search algorithms operating in the behavioural space.

4.1 Fitness function design guidelines

As the fitness function design is completely related to each problem individually, as documented in D6.3, In the scope of the CPSwarm Workbench, the authors have mainly identified the main guidelines for a user for defining working fitness functions:

Defining scope and modelling sub-problems for complex goals: As mentioned above, the fitness function for a problem is directly related to the specifications for that problem. Nevertheless, a good start for designing an effective fitness function is to define the specifications for a give problem. Moreover, if an objective proves to be too difficult for a system, it might help to decompose it into simpler sub-objectives with lower utility values, for example, to evolve robots playing soccer it is good to reward players for kicking the ball since it directly correlates to the number of goals and consequently to the fitness of the solution [5].

Topology of fitness landscape: In general, adversary fitness functions [6], fitness function with a large stochastic component (noise) and fitness functions with local cost minima can affect the optimization time and quality of optimization outcome. While the fitness is initially derived by the problem description, a refinement of the fitness function towards a "smooth topology" can significantly improve the result. Furthermore, the search space can be reduced by assigning high penalties towards unwanted behaviours (an example is a robot car that should go forward and orient itself, in this case, going backward could be excluded as behaviour). However, keep in mind that excluding certain behaviours accidently might cut off solutions which are not obvious but have superior performance in the end.

Combined fitness functions: In many cases, the fitness function comprises orthogonal goals, for example, a robot swarm could have assigned a fitness to stay together, while having a second goal to move forward as a swarm. Typically, these goals can be easily expressed as separate fitness functions but not easily into a single combined one. Some optimization algorithms can perform a multi-dimensional search which yields results in form of a set of non-dominated solutions. After all, this requires a selection based on a combined fitness in before deployment, furthermore not all optimization algorithms in CPSwarm support this approach. Therefore, fitness functions are often described as a weighted sum of criteria, which shifts the problem of defining proper weights. While this ultimately depends on initial requirements, a quick guideline can be to normalize criteria based on their measured variance to get a set of equally matched criteria.

Computational effort to derive the fitness value: In some cases, where the fitness function is derived from a simulation of the target system, the computational effort for computing the fitness function can become the defining part of the overall evolutionary algorithm. In some cases, typically early in the optimization process, a simpler fitness calculation which is considerably faster could significantly speed up the process. For the example where a fitness function is derived by a simulation this could be done with fewer repetitions of simulations (e.g. averaging the results of a few simulations in the beginning and increasing this amount at later generations, where accuracy is needed), shorter simulation time (adjusting simulated time depending on generations) or reduced accuracy (simulating with larger time steps/lower resolution in early generations).

CPSwarm Workbench aims to ease the problem-specific aspects of designing a fitness function as much as possible, as detailed in the following subsection.

4.2 Fitness function design in CPSwarm workbench

The CPSwarm Consorium has developed a tool integrated in the Modelling Tool, named Fitness Function Design Tool, which allows to design a fitness function starting from its mathematical model. This subsection will present the approach followed for the design of the fitness function and then its integration in the Simulation and Optimization Environment, then the next will present a concrete example based on the CPSwarm logistic scenario.

In the CPSwarm Workbench, more specifically the modelling phase, the Workbench allows to design the fitness function to be associated with a behavior, to optimize parameters and simulation settings after that the same Modelling Tool (i.e. Modelio[5]) has been used to define the state machine to address the problem. The difference between parameters and simulation settings is that parameters are used by the algorithms to change the behavior of the CPS, instead simulation settings are used to setup the simulation (see Section 4.2.3 for concrete examples). Furthermore, as shown in the SAR scenario, presented in D6.3, using the state machine approach, the complex scenario can be subdivided in several more simplex states (e.g., the logistics scenario can be subdivided in several states) and then the fitness function can be designed for every state (e.g. only optimizing the parameters for the coverage of the warehouse).

The Modelling Tool allows to describe the fitness function using mathematical expressions. When the fitness function has been modelled, the Modelling Tool uses this model to generate the code that will then be used in the rest of the Workbench.

The model of the fitness function is stored also in the modelling library, to be reusable in different contexts. Furthermore, the library is pre-loaded with a set of default fitness functions associated with the behaviours designed for the proposed scenarios. For example, regarding the SAR scenario, the library could be preloaded, with default fitness functions, for all the possible states.

Based on the model designed, the Mdelling Tool is able to generate a python script, which takes the inputs from the ROS bags6 (log files where during the simulation the messages, published on ROS topics are stored) and calculate the fitness score printing the result on the standard output. This script is executed by the SM each time a simulation is finished during an optimization process and then the value calculated is sent to OT.

Finally, the Modelling Tool allows to the user to select the parameters and simulation settings that he wants to optimize. The list of the parameters and simulation settings are passed to SOO with the ranges to be used for each of them. The SOO instructs the OT to start the optimization on these parameters. After the results are ranked by the OT, the Simulation and Optimization Environment returns the configuration file with the optimized parameters to be deployed on the CPS to optimize the behaviour.

[5] https://www.modelio.org/
[6] http://wiki.ros.org/Bags

After the introduction of the general approach, the next subsections will introduce a concrete example of how this approach is applied to the logistics scenario.

4.2.1 Scenario Overview

To illustrate the CPSwarm approach, the authors introduce the following scenario taken from the logistics domain. The logistics use case envisages a scenario where two classes of robots, scouts and workers, assist in moving boxes in a warehouse. The scout robots, equipped with a QR-code reading camera, rove around the warehouse space searching for boxes. Once a box is located, the scout notifies all workers robots of its location. Idle worker robots bid for the job of transporting the box to a specified location based on their current location, i.e. the distance to the box, and their remaining battery level. The selected workers move to the box, autonomously navigating around obstacles, lifts it using its elevator mechanism, moves to the destination, sets the box down and returns to an idle state. The entire scenario in its final version is described in "D8.4 – Final Swarm Logistics Demonstration" (M36). The simulation of this scenario is shown in Figure 6.

For simplicity, each scout robot follows a random walk behavior: it picks a random direction and "walks" until it encounters a box, the edge of the operating space or a distance threshold has been exceeded. It then picks a new random direction and continues walking as before. As the distance parameter affects the effectiveness of the Scout coverage of operating space, it may be optimized for the specifics of the scenario. In addition, the number of scout and worker robots may be varied to affect the overall performance of the system.

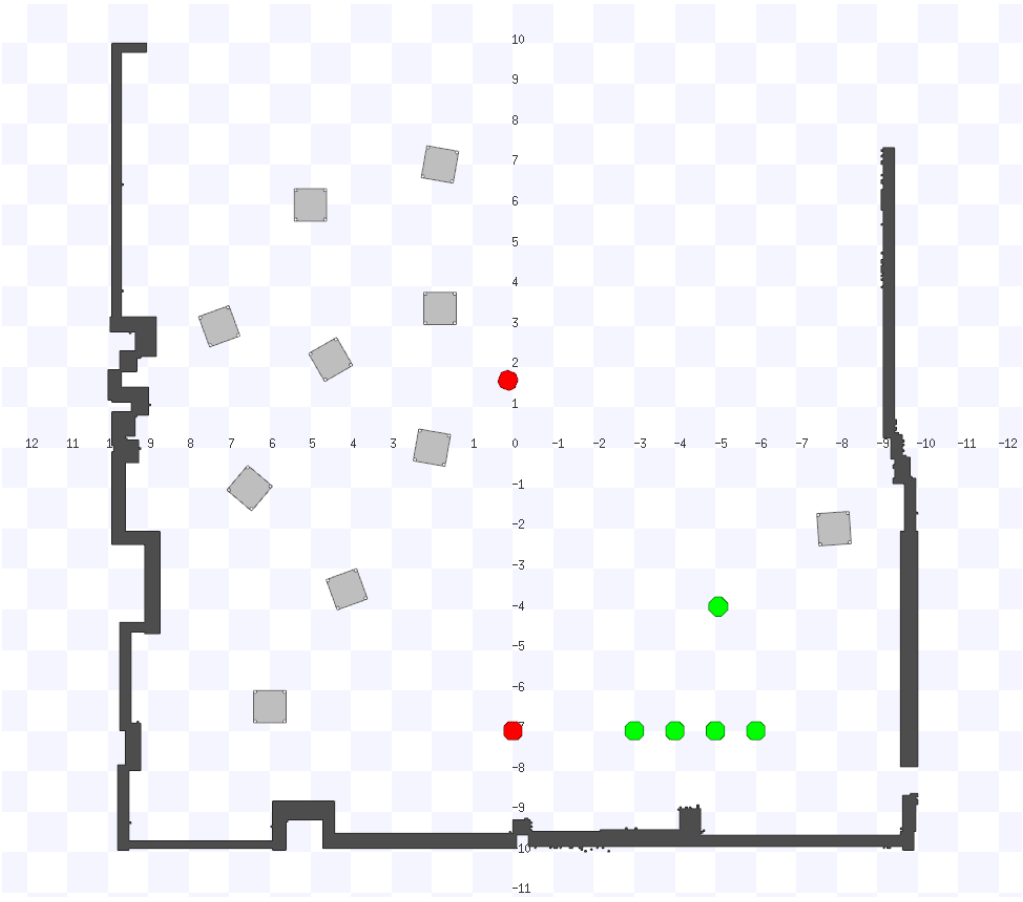

Figure 6 - Logistics scenario simulation

4.2.2 Fitness function

The overall effectiveness of the system may be judged by its productivity, i.e. average time taken to transport all boxes to the goal area. This effectiveness of the system is expressed by a fitness function as described in the following equations.

- Simulation time: t_{sim}.
- Total number of boxes to deliver: nb.
- Delivery time of a given box to the goal area:

$$T = \{t_i | i \in [0, nb)\}$$
$$t_i \in [0, t_{sim}]$$

- Average time of box delivery:

$$t_{avg} = \frac{\sum_{x=1}^{nb} T_x}{nb}$$

- Fitness function is the percentage of average time delivery.

$$C = \frac{t_{sim} - \min(t_{avg}, t_{sim})}{t_{sim}} * 100$$

These equations have modelled using SysML parametric diagram, as shown in Figure 7.

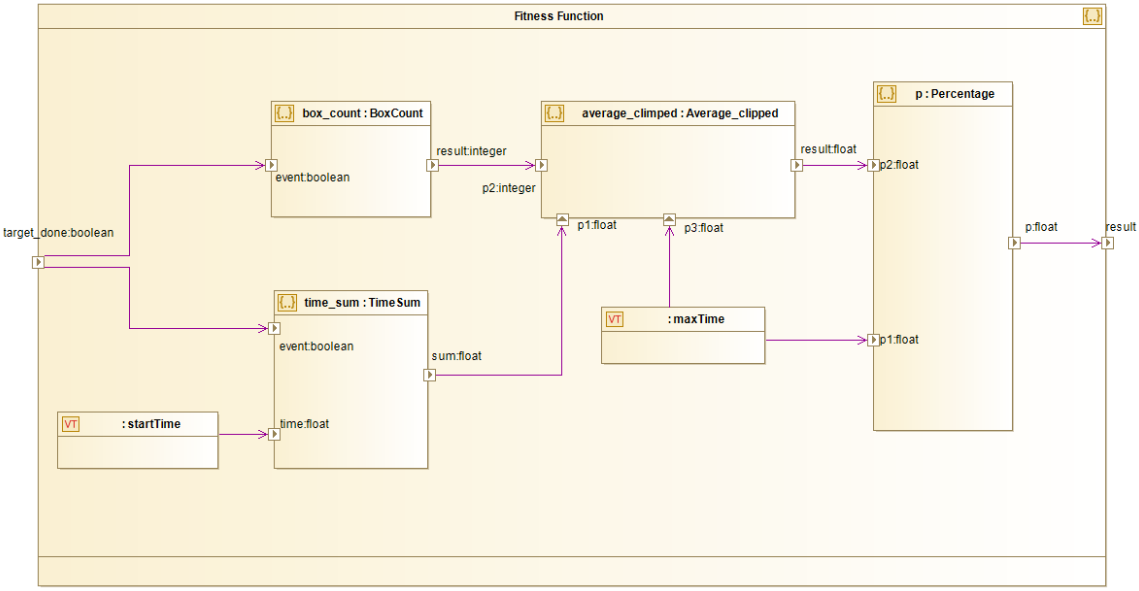

Figure 7 – Fitness function under the Modelling Tool

The model designed is then exported in the python script (see ANNEX A). The script before reads the ROS bags to collect the info about the box moved by the robot, then the script calculates the fitness score as the average time required to move the boxes. This script is passed to the Simulation and Optimization Environment where it is used to calculate the fitness score to be associated to each combination of parameters set in the simulation. The parameters and simulation settings are set using the Modelling Tool as indicated in the next subsection.

4.2.3 Optimization Setup

As mentioned in Section 4.2.1, three different values may be modified during optimization: one is a parameter of the behaviour, namely the walk length; other two are simulation settings, i.e. the scout count and the worker count. Each parameter is tagged, in the modelling tool, as 'optimizable'. By tagging a model element as optimisable, the user must specify a value range (with a minimal and a maximal values) and a step size as shown in Figure 8.

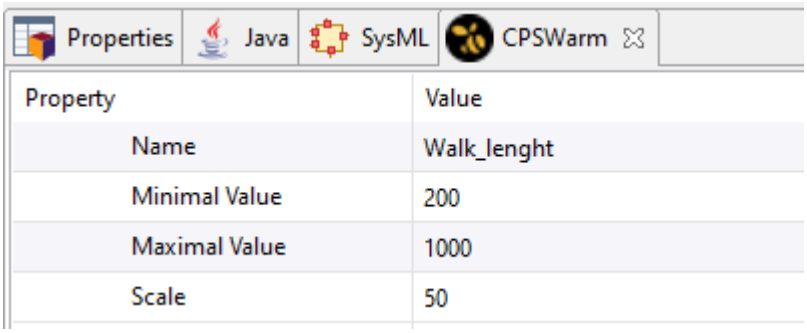

Figure 8 - Optimisable Parameter under the Modelling Tool

Table 2 summarizes the specified of these.

Table 2 - System parameters to be optimized

Parameter name	Minimum value	Maximum value	Step size	Notes
Walk_length	200	1000	50	Random walk length in centimetres
Scout_count	1	10	1	Number of scout robots
Worker_count	1	10	1	Number of worker robots

These values are exported by the Modelling Tool and passed to SOO and from SOO to the OT. The OT will optimize these parameters and simulations settings, producing as a result the combination of values that gives the best fitness score. In this example provided in this deliverable, the one that gives the less average time to move the boxes.

Finished the optimization, the configuration file for the algorithm containing the optimized values for the parameters indicated in the Modelling Tool is produced by the SOO and then can be deployed using the on the CPS using the Deployment Tool. Furthermore, all the fitness scores obtained associated to the values set for the parameters and simulation settings to obtain them, can be checked by the user in a dashboard. In the example given, this can be useful for the user to select how many scout and worker robots to deploy in the warehouse.

5 Conclusion

This deliverable has described the way in which the CPSwarm Workbench supports a swarm of CPSs design optimization using an evolutionary approach and the final guidelines for designing a fitness function to create the desired behaviour of a swarm of CPSs. The document introduces the API and the optimization workflow defined for the CPSwarm solution. Then, the authors present the final fitness function design guidelines and detail how they have been applied in the CPSwarm Workbench through the introduction of the Fitness Function Design tool.

6 References

[1] L. C. J. a. R. P. J. A. J. F. Van Rooij, «Neural Network Training Using Genetic Algorithms,» in *World Scientific Publishing Co.,*, Mar. 1997.

[2] D. C. M. J. M. S. E. F. W. E. Micha Rappaport, «Distributed Simulation for Evolutionary Design of Swarms of Cyber-Physical Systems,» in *ADAPTIVE 2018*, 2018.

[3] D. T. B. P. B. A. L. R. T. a. M. A. S. Conzon, «The VIRTUS Middleware: An XMPP Based Architecture for Secure IoT Communications,» in *21st International Conference on Computer Communications and Networks (ICCCN)*, 2012.

[4] D. a. J. U. Floreano, «Evolutionary robots with on-line self-organization and behavioral fitness.,» *Neural Networks,* vol. 13, n. 4-5, pp. 431-443, 2000.

[5] F. I., *On Evolving Self-organizing Technical System,* PhD thesis, Alpen-Adria-Universität Klagenfurt, 2013.

[6] A. J. Lockett, «Insights From Adversarial Fitness Functions.,» in *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15). ACM,*, New York, NY, USA, 2015.

Acronyms

Acronym	Explanation
ANN	Artificial Neural Network
API	Application Programming Interfaces
CPSs	Cyber-Physical Systems
FREVO	FRamework for EVOlutionary design
GUI	Graphical User Interface
ID	IDentifier
NNGA	Neural Network Genetic Algorithm
OID	Optimization IDentifier
OT	Optimization Tool
ROS	Robot Operating System
SAR	Search and Rescue
SCID	Simulation Configuration IDentifier
SM	Simulation Manager
SOO	Simulation and Optimization Orchestrator
SS	Simulation Server
ST	Simulation Tool
XMPP	eXtensible Messaging and Presence Protocol

List of figures

List of tables

ANNNEX A

```
import rosbag, sys
def fitness():
  if (len(sys. argv) != 3):
    print " Logistics fitness function calculator  Usage:\n fitness.py [bagfile.bag] [maximum simulation time]\n "
    sys.exit(1)

  bag = rosbag.Bag(sys.argv[1])
  max_time = float(sys.argv[2])
  start_time = float(bag.get_start_time())
  time_sum = TimeSum ("target_done", start_time, bag)
  box_count = BoxCount("target_done", bag)
  average_clipped = Average_cliped(time_sum, box_count, max_time)
  fitness = Percentage(max_time , average_clipped)
  print fitness

def Average_cliped(p1, p2, p3):
  return min (p1 / p2, p3)

def Percentage (p1, p2):
  return (p1 - p2) / p1 *100

def BoxCount(event, bag):
  result = 0
  for subtopic, msg, t in bag.read_messages(event):
    result += 1
  return result

def TimeSum(event, time, bag):
  result = 0
  for subtopic, msg, t in bag.read_messages(event):
    timeMsg = TimeMsg(msg)
    result += timeMsg - time
  return result

def TimeMsg(p1) :
  return float(p1.header.stamp.secs)

fitness()
```