



## **D3.5 – UPDATED CPSWARM WORKBENCH AND ASSOCIATED TOOLS**

Deliverable ID	<b>D3.5</b>
Deliverable Title	<b>Updated CPSwarm Workbench and associated tools</b>
Work Package	<b>WP3 – Architecture design and Component Integration</b>
Dissemination Level	<b>PUBLIC</b>
Version	<b>1.0</b>
Date	<b>2018-10-31</b>
Status	<b>Final</b>
Lead Editor	<b>FRAUNHOFER</b>
Main Contributors	Junhong Liang ( <b>FRAUNHOFER</b> ), Farshid Tavakolizadeh ( <b>FRAUNHOFER</b> ), Davide Conzon ( <b>ISMB</b> )

**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
0.01	2018-10-02	Junhong Liang (FRAUNHOFER)	First Draft with TOC and initial content
0.02	2018-10-13	Farshid Tavakolizadeh (FRAUNHOFER)	Added content to continuous integration and delivery
0.03	2018-10-15	Junhong Liang (FRAUNHOFER)	Added content to automatic component tests
0.04	2018-10-15	Davide Conzon (ISMB)	Added content to SOO continuous integration
0.05	2018-10-19	Junhong Liang (FRAUNHOFER)	Fixed text and formatting errors
0.06	2018-10-24	Junhong Liang (FRAUNHOFER)	Incorporated review feedback from LAKE
0.07	2018-10-26	Junhong Liang (FRAUNHOFER)	Incorporated review feedback from UNI-KLU
0.08	2018-10-26	Farshid Tavakolizadeh (FRAUNHOFER)	Document restructure based on the current progress
1.0	2018-10-31	Junhong Liang (FRAUNHOFER)	Released for final submission

## Internal Review History

Review Date	Reviewer	Summary of Comments
2018-10-24	Micha Rappaport (LAKE)	Accepted with minor comments
2018-10-26	Midhat Jdeed (UNI-KLU)	Accepted with minor comments

## Table of Contents

Document History .....	2
Internal Review History .....	2
Table of Contents .....	3
1 Introduction.....	4
1.1 Scope .....	4
1.2 Related documents.....	4
2 CPSwarm Updated Components .....	5
2.1 Updated Modelling Tool .....	5
2.2 Updated Modelling Library .....	6
2.3 Initial CPSwarm Abstraction Library.....	8
2.4 CPSwarm Communication Library.....	9
2.5 Initial Deployment Tool .....	9
2.6 Initial Monitoring Tool .....	10
2.7 CPSwarm Launcher .....	10
3 Continuous Integration and Delivery .....	12
3.1 Automatic Component Building and Testing .....	12
3.2 Build and Test Monitoring.....	16
3.3 Continuous Delivery .....	17
4 Conclusions.....	23
Acronyms .....	24
List of figures.....	24
References .....	25

# 1 Introduction

In the previous deliverable D3.4, the methodology and infrastructure for continuous integration (CI) of the CPSwarm software was documented. In addition, the set up for the initial phase of the CPSwarm components integration was presented in the same document. As an updated version of D3.4, this deliverable focuses on documenting the second phase integration of the CPSwarm components. A brief introduction of all components to be integrated is given in chapter 2. Component and integration test setups are documented in chapter 0.

## 1.1 Scope

The second phase integration involves both updated components from the previous integration as well as newly developed components. More detailed descriptions can be found in corresponding deliverables (D3.2, D4.2, D4.5, D4.7, D5.3, D6.5, D7.1, D7.3, and D7.5). However, it is important to note that the integration did not follow strictly the plan defined in the Description of Action (DoA), since the project has evolved dramatically in the period and many components could not be matched one-to-one to those defined in the original DoA. This integration will be carried out based on the components that are currently available within the CPSwarm system.

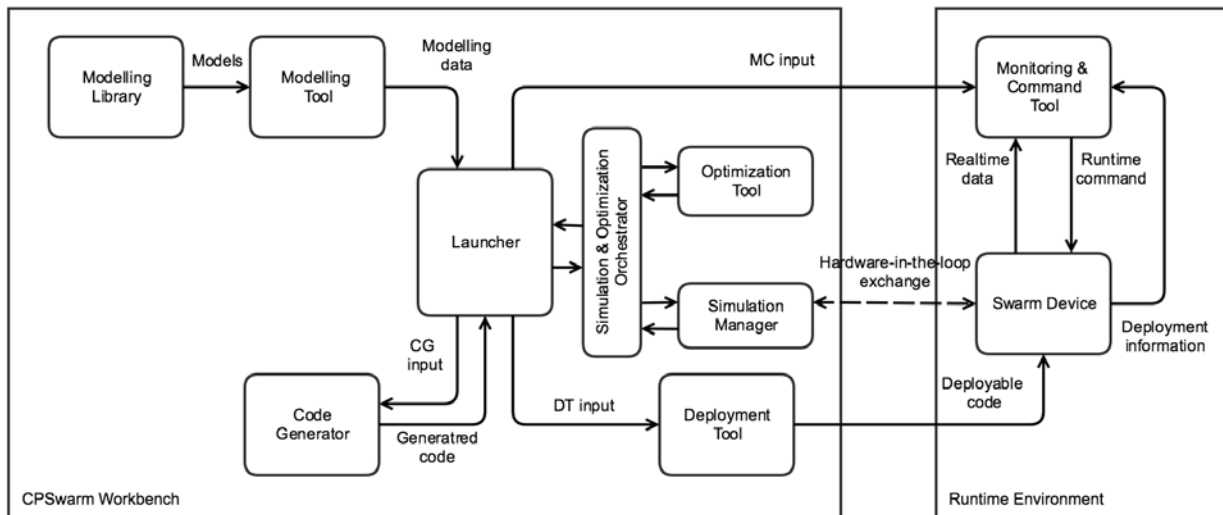
## 1.2 Related documents

ID	Title	Reference	Version	Date
[RD.1]	Updated System Architecture Analysis & Design Specification	D3.2	1.0	2018-06-30
[RD.2]	Updated CPS modeling library	D4.2	1.0	2018-09-30
[RD.3]	Updated Swarm modelling library	D4.5	1.0	2018-10-31
[RD.4]	Initial Security threat and attack models	D4.7	1.0	2018-10-31
[RD.5]	Updated CPSwarm Modelling Tool	D5.3	1.0	2018-06-30
[RD.6]	Initial Integration of external simulators	D6.5	1.0	2018-06-30
[RD.7]	Initial CPSwarm Abstraction Library	D7.1	1.0	2018-06-30
[RD.8]	Initial Bulk deployment tool	D7.3	1.0	2018-09-30
[RD.9]	Initial Monitoring and configuration framework	D7.5	1.0	2018-10-31

## 2 CPSwarm Updated Components

An updated architecture design for the CPSwarm system was presented in D3.2; see Figure 1. The updated architecture reflects the newest concept of the CPSwarm system. New components are defined, which were not foreseen during the time of writing for the original DoA. As a result, it was no longer feasible to strictly follow the integration plan defined in DoA. Therefore, in the second phase of CPSwarm component integration, we'll adapt the new integration plan with the newly evolved architecture, while still trying to follow the spirit of the original plan. With the new plan, we have defined the following components to be integrated in this phase of CPSwarm:

- Updated Modelling Tool (documented in D5.3)
- Updated Modelling Library (three sub-libraries documented in D4.2, D4.5, D4.7)
- Simulation & Optimization Orchestrator, Optimization Tool and Simulation Manager (documented in D6.5)
- Initial Abstraction Library (documented in D7.1)
- CPSwarm Communication Library (documented in D7.1)
- Initial Deployment Tool (documented in D7.3)
- Initial Monitoring Tool (documented in D7.5)
- Launcher (documented in D3.2)



**Figure 1. Updated architecture design for CPSwarm system (Extracted from D3.2).**

In the following sub-chapters, a brief introduction and recap of each of the aforementioned components are presented to remind the reader about the context of the integration.

### 2.1 Updated Modelling Tool

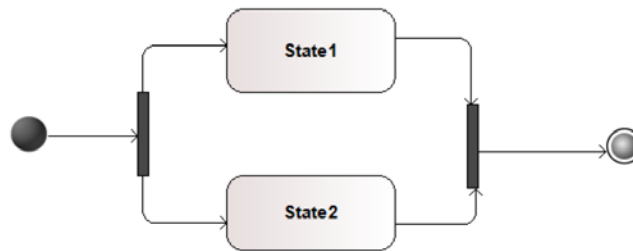
The CPSwarm Modelling Tool is built on top of the Modelio open source modelling environment as previously described in Deliverable D5.2. Swarm modelling can be succinctly described as the creation and population of several diagrams.

Compared to the initial Modelling Tool, a lot of previously discussed features have been implemented in the updated version, including the following functionalities:

- CPS Population Design Tool (CPDT): Modelling a swarm from scratch can be complicated. Therefore, the CPDT is implemented to help a user to start modelling quickly. The CPDT is a wizard-like GUI which helps the user to create swarm models from scratch by stepwise guiding of the user to specify

common parameters such as size of the swarm, members of the swarm, etc. After all parameters are defined, a set of models for the specified swarm is automatically generated, available to the user for further modification.

- More aspects of swarm modelling: In the updated Modelling Tool, more aspects of a swarm can be modelled, such as the composition of the swarm, the architecture of a swarm member, the mission goal of the swarm, behaviour of swarm members, etc.
- Modelling of behaviour state-machine: The consortium has decided to use state-machines to model the behaviour of a swarm member in different states, e.g. during start-up, in different operation modes, during shutdown, etc. The updated Modelling Tool provides the possibility to model such a state-machine with its GUI. See Figure 2.
- SCXML generation from behaviour state-machine: The updated Modelling Tool also provides the possibility to convert the modelled state-machine into a standard SCXML file, which can be used by the Code Generator to produce an executable for this behaviour state-machine. Through this method, the modelling language can be automatically turned in to executable program without manual conversion by the user.



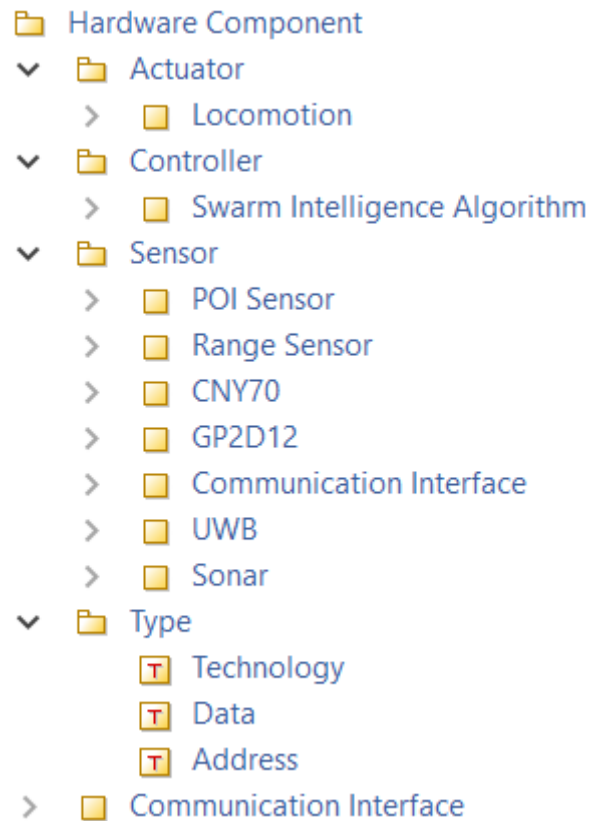
**Figure 2. State-machine modelled with Modelling Tool (Extracted from D5.3).**

## 2.2 Updated Modelling Library

The updated Modelling Library corresponds to the updated CPS modelling library, updated swarm modelling library and the initial security threat and attack models, whose detailed description can be found in D4.2, D4.6, and D4.7, respectively.

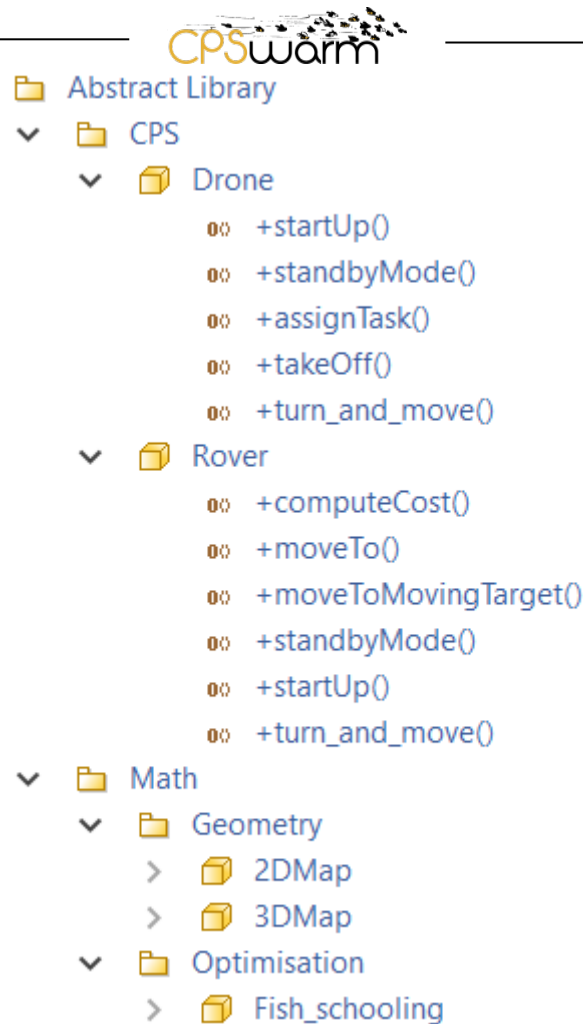
The CPSwarm Modelling library is composed of a set of predefined elements, which can be reused in the CPSwarm design context. In this updated version, models in the following categories are included:

- Hardware Components: The hardware component category groups all the elements available for the CPS hardware description. This category consists of three sub-categories, respectively named "Actuator", "Controller", and "Sensor". In each sub-category a set of components representing the physical devices are listed which can be used to build a CPS. Figure 3 shows an example of possible hardware component models.



**Figure 3. Example of hardware components (Extracted from D4.2).**

- Behaviour: The behaviour category groups all swarm algorithms available for the CPS behavioural description. Each swarm algorithm or behaviour is described as a UML state machine and added to the CPSwarm modelling library.
- Abstraction Library: The third category of elements consists of a model representation of the Abstraction Library as defined in D5.3. According to this deliverable, *"the Code Generator relies on the Abstraction Library that provides a set of platform-independent functionalities"*. These functionalities or services has to be abstracted as model usable for the behaviour modelling. Figure 4 shows part of the Abstraction Library in which different folders are defined to sort the functionalities. Under each folder a set of nodes is available. For example, the CPS folder presents two nodes respectively named "Drone" and "Rover". Finally, functionalities are defined under the nodes.



**Figure 4. Abstraction Library model example (Extracted from D4.2).**

Currently these library packages are part of the CPSwarm extension for Modelio. The correctness of such packages will be tested within the CPSwarm extension component test. Besides, tests can also be set up to test the behavioural algorithms within these library packages.

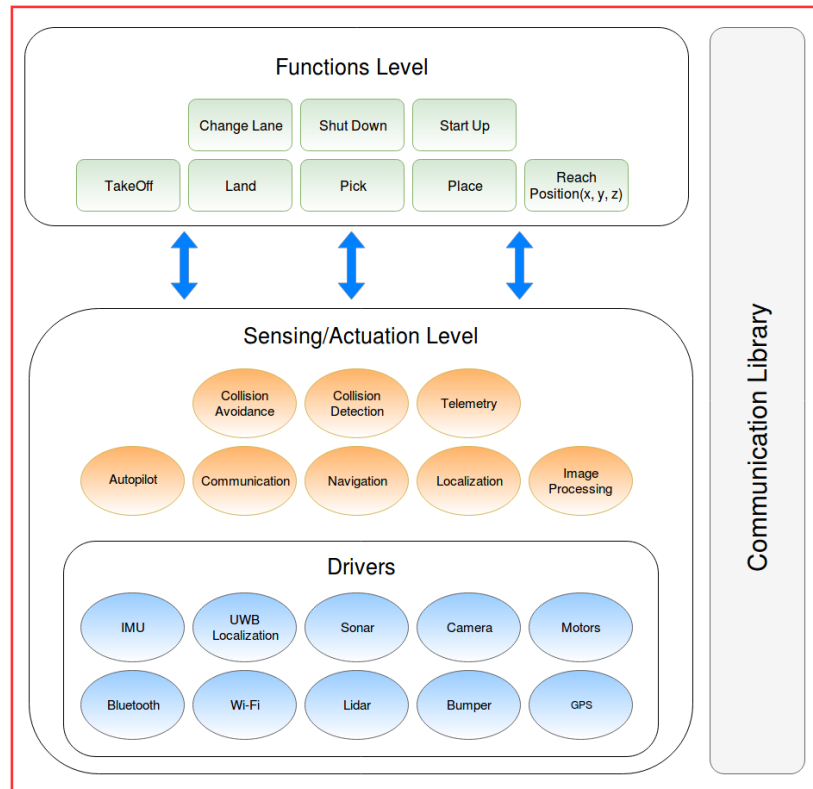
### 2.3 Initial CPSwarm Abstraction Library

One of the greatest challenges in working with CPSs are the wildly different platforms and devices available on targets. Therefore, the CPSwarm Abstraction Library is developed to hide away all platform and device specific details and provide a platform independent interface for the higher layer application logic. The CPSwarm Abstraction Library covers two different roles inside the project: the first is to provide a set of CPS-specific adaptation libraries in order to access heterogeneous robotic system in a standard and coherent way. The second is to provide support for the development of algorithms using model-driven approaches that can be deployed on CPSs using an automated code generation process.

To achieve these purposes, the abstraction library is organized as a composition of tree main overlapping components, called levels:

- Functions Level
- Sensing/Actuation Level
- Drivers Level





**Figure 5. The CPSwarm Abstraction Library (Extracted from D4.1).**

Currently, multiple components in the Abstraction Library have been developed to be used on different target CPSs. An example can be seen in Figure 5.

## 2.4 CPSwarm Communication Library

The CPSwarm Communication Library (also known as Swarmio) is a reference implementation of runtime communication for swarm members as well as the Monitoring Tool. It provides unified interface tools for communication in an efficient and secure fashion. It takes on the duty to ensure that all communication happens with the desired reliability, security level, and latency, so that higher level applications do not have to be concerned with such details. The communication library provides the following functionalities:

- Swarm members discovery within the network
- Events and commands transmission
- Remote parameter adjustment
- Message signature and encryption

Currently, the Communication Library is deployed along with the Abstraction Library on swarm members. In addition, the Monitoring Tool relies on it exchange data with the swarm members.

## 2.5 Initial Deployment Tool

To ease the effort in deploying programs to a large number of swarm members, an over-the-air (OTA) bulk deployment tool is designed and developed within this phase of the project. The Deployment Tool consists of two main components: the deployment manager and the deployment agent. A deployment agent is a lightweight service which runs on each swarm member. It subscribes to specific channels for program updates. The deployment manager is a service that manages such channels. It is also a service to which users can post deployment instructions as well as artefacts to be deployed on target swarm members. When a new update is submitted, the manager transmits necessary information to all involved agents through a publish-

subscribe mechanism. The involved agents execute the deployment instructions specified in the update. Figure 6 shows the architecture of the initial Deployment Tool.

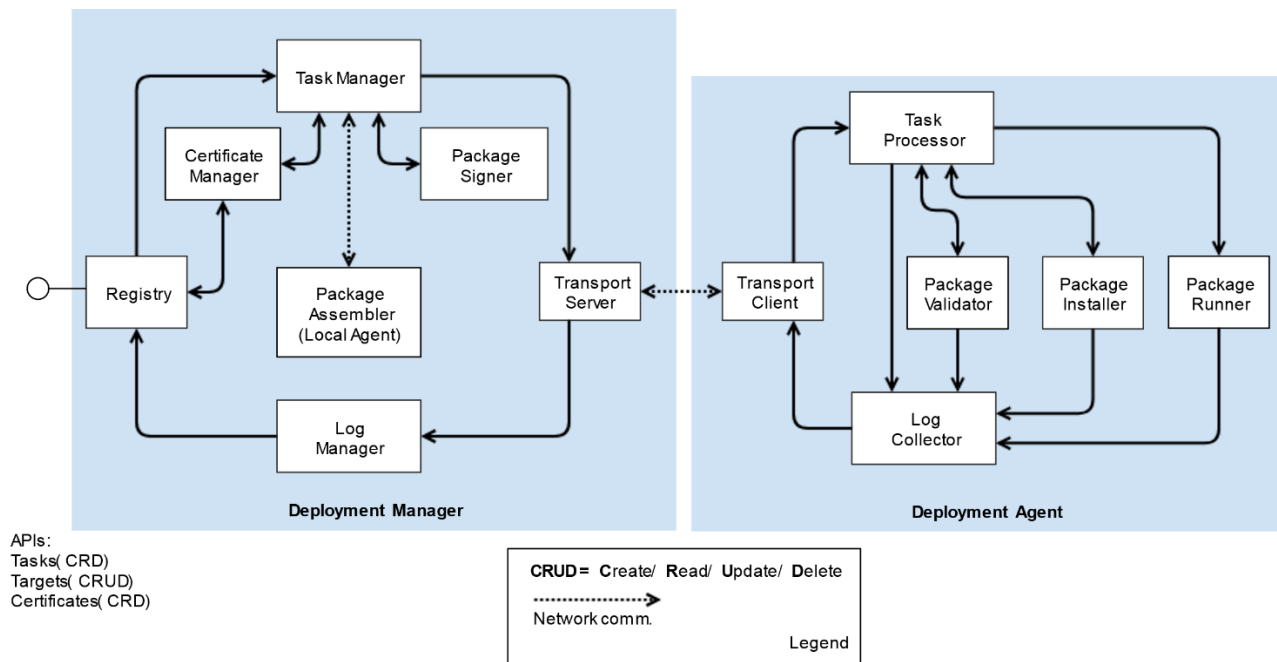


Figure 6. Deployment Tool architecture (Extracted from D7.3).

## 2.6 Initial Monitoring Tool

It is important for the user to get real-time feedback of the status of all swarm members during runtime. The Monitoring and Command Tool is developed to address this issue. This tool is a runtime GUI application which is used for monitoring the real-time status of swarm members as well as sending commands to the swarm. The current implementation of the initial Monitoring Tool provides a web interface as the GUI for the operator. On the backend, it utilizes the CPSwarm Communication Library for secure communication with swarm members.

## 2.7 CPSwarm Launcher

As mentioned in D3.2, the consortium has decided in favor of building up the CPSwarm workbench with highly decoupled software components instead of developing a monolithic system. Each of such decoupled components specializes in doing specific dedicated tasks, which enables high reusability and flexibility. However, managing such a high number of decoupled components together with all the asset files they consume or produce can be tedious and error-prone. Hence, the consortium has adopted the idea of creating a CPSwarm Launcher, which helps the user to solve this problem. The CPSwarm Launcher is a desktop GUI application that has the following tasks:

- Launching different components with configurations specified by the user.
- Managing all asset files needed and created by different components.
- Helping the user to navigate through the swarm design process, by checking if proper input is available for each step. If no proper input is available, that step will be disabled, hinting user to finish prerequisite steps first.

Figure 7 shows a screenshot of the CPSwarm Launcher prototype.

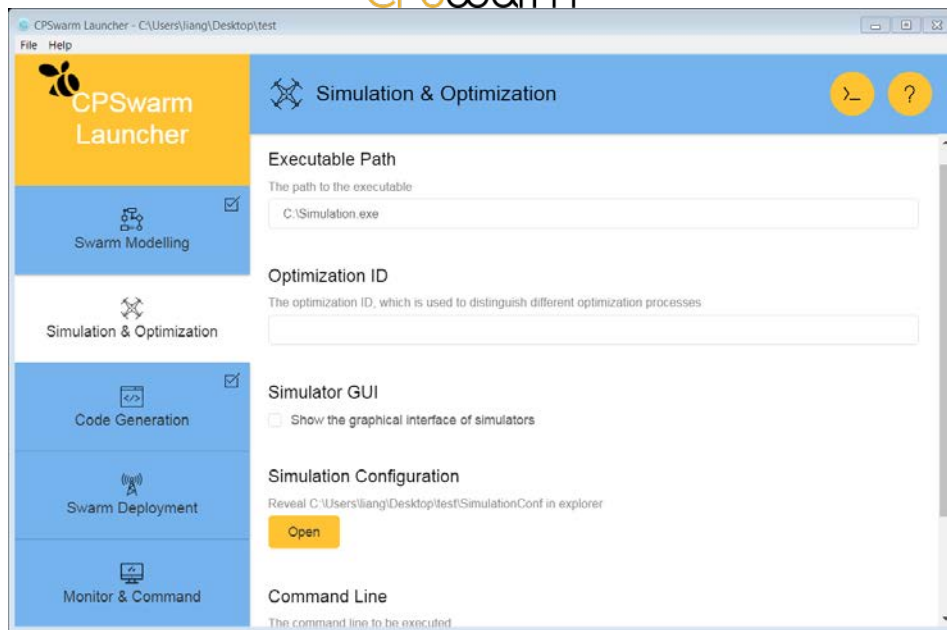


Figure 7. CPSwarm Launcher screenshot.

### 3 Continuous Integration and Delivery

The deliverable D3.4 provided a detailed description of the CPSwarm CI platform consisting of Git and build infrastructures. The same document also provided guidelines on how to use the platform based on common best practices. Following the test and integration plan reported in D3.7, this chapter reports the usage of the platform and additional components that are added to the platform until M22.

#### 3.1 Automatic Component Building and Testing

As per design, all components are built and tested in isolation using Docker Containers. A Docker Container is an isolated environment in which software can run without being influenced by the host computer's environment. This enables clean iterations of building and testing without affecting the host environment or consecutive builds. In addition, the containers allow execution of components in parallel and perform integration tests that rely on several components at the same time.

On the CPSwarm CI project<sup>1</sup>, several plans have been established for individual components to automatically build and run tests whenever new changes have been pushed to the related code repositories.

The tests not only verify the correctness of individual components, but also ensure the integration of components with each other. This is possible, because of how the CPSwarm system is built up. As mentioned in D3.2, instead of building a monolithic CPSwarm workbench, the consortium has decided to build up the CPSwarm workbench with highly decoupled software components, each of which is dedicated to do a specific task. Building the CPSwarm system this way not only enables high flexibility and reusability, but also simplifies the integration testing process. Since the information exchange between components is mostly file-based and the schema of such files has been defined, it is not necessary to run all components together to test for integration. Instead, each component can now run an end-to-end test independently against the given sample input and output files to verify its integration with other components. If the component could generate a well-formed file from the given sample input, we can verify that the component is correctly implemented.

As a result, this testing approach is used for the second phase of CPSwarm development to test the integration between components, since it can achieve the same testing effect with significantly less effort required in writing test cases and setting up test plans.

At the time of writing, few plans have been created to carry out such tests on different components. The plans are explained in detail in the following sub-chapters.

##### 3.1.1 Modelling Tool

The testing of Modelio is out of the scope of the CPSwarm project. However, the Modelio CPSwarm extension is the plugin which interacts directly with other CPSwarm components and it should be covered by the testing. A Bamboo plan named *Modeling Tool (Modelio CPSwarm Extension)* has been established to test the CPSwarm extension. The plan has the following tasks:

- Build Modelio CPSwarm extension from source
- Copy the extension into a sample Modelio project
- Load the project as well as the extension into Modelio
- Generate output files from sample project data using the CPSwarm extension
- Verify the output files against reference output files

Currently, a SCXML file is configured as the reference output file for verification. The SCXML file is the interface between the Modelling Tool and the Code Generator.

<sup>1</sup> <https://pipelines.linksmart.eu/browse/CPSW>

### 3.1.2 Simulation Optimization Orchestrator

A Bamboo plan named *Simulation and Optimization Orchestrator* is created to test the Simulation Optimization Orchestrator (SOO). The plan has the following tasks:

- Build the SOO from source
- Run end-to-end test on the built SOO

The end-to-end test focuses on verifying the interaction between the SOO, the Simulation Manager (SM) and the Optimization Tool (OT). There are three sub-tests for this end-to-end test. Figure 8 shows the interaction between components in a sequence diagram:

#### Creation Test

This test is used to verify:

- The creation of the eXtensible Messaging and Presence Protocol (XMPP<sup>2</sup>) user of the SOO on the XMPP server.
- The creation of the XMPP user of the Dummy Simulation Manager used to test the features of the SOO.
- The creation of the rosters (list of known users) of the two components used to receive their presence and to know their availabilities.

The test is passed if after the respective creation of the two users, the manager has been successfully added to the roster of the SOO.

#### Simulation Test

This test is used to verify:

- The start of the SOO with a set of requirements for the simulation to be run (dimensions, number of agents).
- The ability to collect the list of available managers and the features they provide, through the presences.
- The ability to match the requirements with the features provided by the Dummy Simulation Manager.
- The ability to select the Dummy Simulation Manager and to send to it the "Run Simulation" message with the right parameters.

The test is passed if the Dummy Simulation Manager is selected and it receives the correct XMPP message to run the simulation.

#### Optimization Test

This test is used to verify:

- The start of the SOO with a set of requirements for the simulation to be run (dimensions, number of agents) and the request to do optimization.
- The ability to collect the list of available managers and the features they provide, through the XMPP presences.
- The ability to match the requirements with the features provided by the Dummy Simulation Manager.
- The ability to send a "Start Optimization" message to the Dummy Optimization Tool, indicating the list of managers to be used (in this case only the one of Dummy Simulation Manager).
- The ability to receive correctly the result of the optimization, from the Optimization Tool, when the process is finished.

The test is passed when the SOO receives correctly the indication of the finished optimization process.

<sup>2</sup> <https://xmpp.org/>

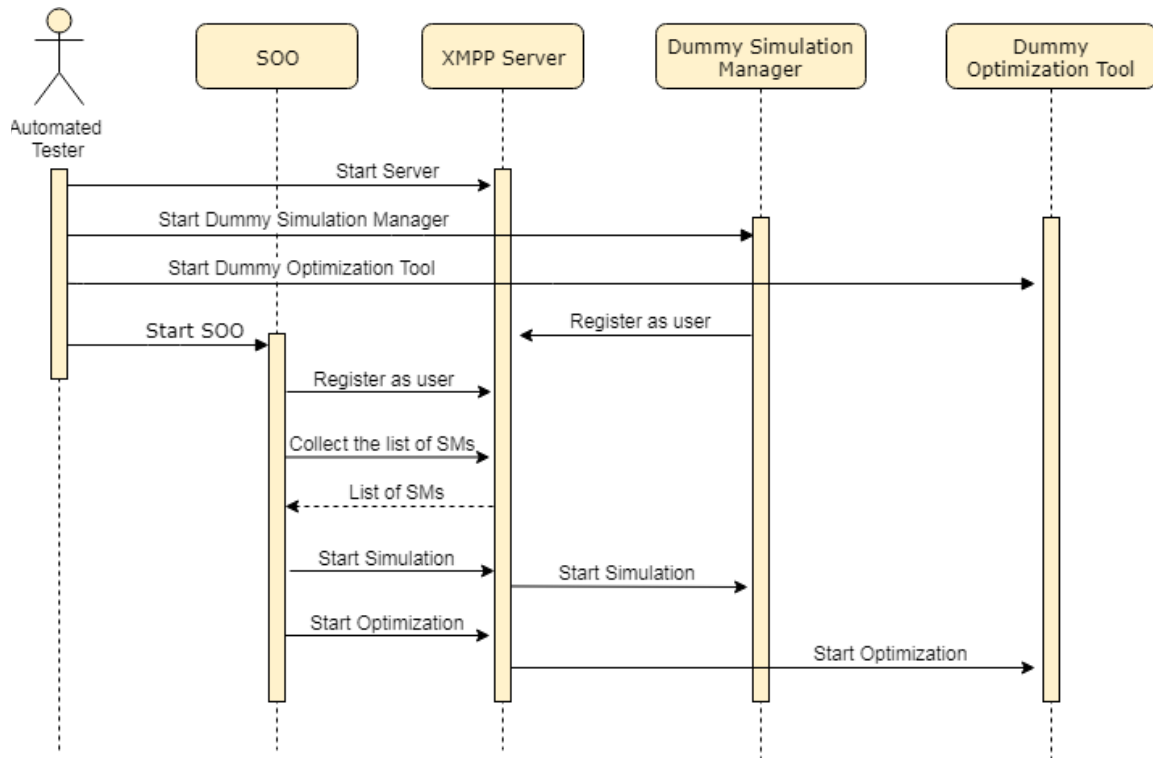


Figure 8. Integration test process, validating the interaction among Simulation Optimization Orchestrator, Simulation Manager and Optimization Tool

### 3.1.3 Code Generator

A Bamboo plan named *Code Generator* has been created for the Code Generator. The plan has the following tasks:

- Build the Code Generator from source
- Run the Code Generator against a given SCXML file as sample input to produce an output file
- Verify the output file against the reference output

### 3.1.4 Deployment Tool

A Bamboo plan named *Deployment Tool* has been created for the Deployment Tool. The plan has the following tasks:

- Build the Deployment Tool from source
- Run the end-to-end test on Deployment Tool

The end-to-end test on Deployment Tool focuses on ensuring the correctness of deployed code. When using the CPSwarm workbench, the Deployment Tool takes reference inputs that are produced by the Code Generator and the user responsible for deployment. The Deployment Tool must accept the input in a predefined format (as defined in Figure 9), process it, and perform the deployment regardless of the internal implementation logic. The end-to-end test for Deployment Tool ensures that the component stays compliant to the specifications. Assuming that the Code Generator and the user follow the same specifications, the success of the end-to-end test infers the integration of Deployment Tool with Code Generator and the workbench as a whole. The test performs a full deployment cycle and validates the final results on a target device. This validation happens at every development iteration and is different from tamper detection that guarantees data integrity in production settings.

The test starts with providing reference source code and deployment task description based on the specification expected by the architecture. It then sends a request to the Deployment Tool, asking for the deployment of the code to a preconfigured virtual target device. After deployment, the tester starts a Test Server and a Test Client on host and target devices respectively. For the sake of this test, both host and target devices are the same machine, the former running the last built instance of Deployment Manager and the latter running the latest Deployment Agent. The Test Server and Client read their own copies of local files and produce checksums. The Test Client sends the checksum to the Test Server where it performs the validation, reporting the results back to the tester. The tester is either a scripted process executed at the CI Server (Automated Tester) or the developer performing the steps manually.

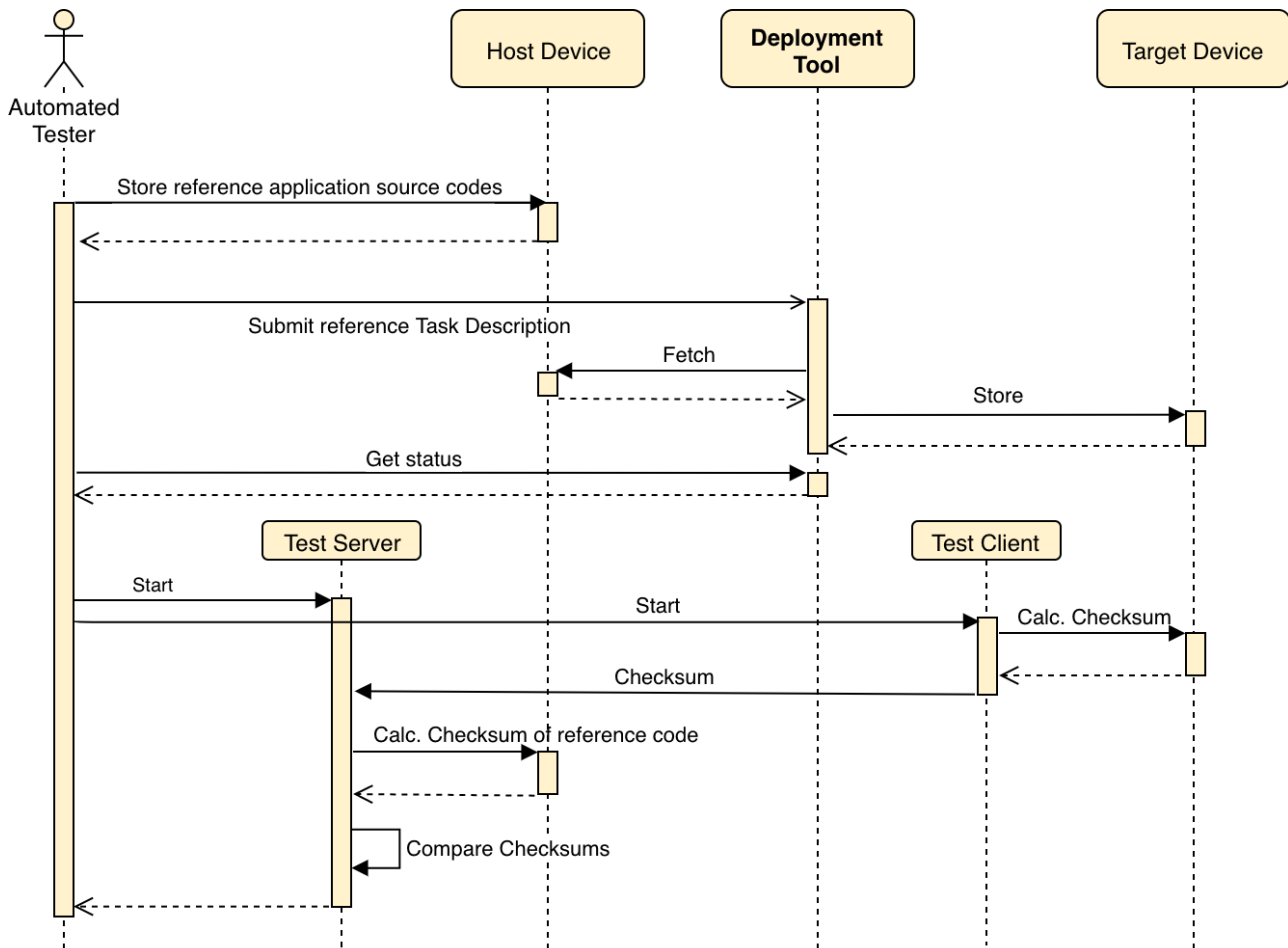


Figure 9. End-to-end test process, validating the behaviour and compliance of Deployment Tool to the specification.

### 3.1.5 Communication Library

A Bamboo plan named *Communication Library (Swarmio)* has been created for the Communication Library. Currently, the plan has the following tasks:

- Build the Communication Library from source

This build plan is still a work in progress. In future, an end-to-end test will be created to validate the functionalities of Swarmio.

### 3.1.6 CPSwarm Launcher

A Bamboo plan named *CPSwarm Launcher* has been set up for the CPSwarm Launcher. Currently, the plan has the following tasks:

- Run unit test cases to verify code correctness
- Build the CPSwarm Launcher from source
- Run end-to-end test on the CPSwarm Launcher

Sample Launcher projects are prepared as the reference inputs for the end-to-end test. The test involves the following aspects:

- 1) Testing of UI behaviour. For this purpose, sample Launcher projects are loaded into the Launcher. UI Behaviours such as whether a tab is displayed in correct state according to the sample projects, whether the list of files is shown correctly, etc. are tested. Such testing is possible with the help of the Spectron<sup>3</sup> framework.
- 2) Testing of Launcher output. For this purpose, sample Launcher projects with pre-configuration are loaded into the Launcher. The Launcher tries to launch testing scripts instead of real components from the pre-configuration. The testing scripts verify whether the Launcher is launching them with correct parameters according to the pre-configuration.

### 3.1.7 Monitoring Tool

The plan for testing the Monitoring Tool is work in progress. The following tasks are planned for this plan:

- Build the Monitoring Tool from source
- Run a dummy swarm member to communicate with the Monitoring Tool to verify the correctness of communication

### 3.1.8 Future Plans

Currently, the test plans cover all the components relevant in the second phase of integration, except the Modelling Libraries. While the Modelling Libraries are not testable executable software, they may include algorithms such as swarm behavioural algorithms, which are testable. As a result, in the future, more test plans will be set up to test the correctness of individual behavioural algorithms, which are included in the Modelling Library.

## 3.2 Build and Test Monitoring

The results of integration tests are automatically reported to developers who contribute to individual source code repositories. Others who are interested in these results have to manually subscribe to receive notifications. In addition, users can refer to the CI project for CPSwarm<sup>4</sup> to monitor the progress and see the status. Figure 11 shows a screenshot of the CI project home page.

---

<sup>3</sup> <https://electronjs.org/spectron>

<sup>4</sup> <https://pipelines.linksmart.eu/browse/CPSW>







Event	Notification recipient	Actions
Notify After 3 Consecutive Failures	Committers ( <i>users who have committed to the build</i> )	 
Failed Builds And First Successful	<a href="#">Junhong Liang</a> ( <i>user</i> )	 

Figure 10. Sample notification settings for a plan (i.e. Launcher)












 Build dashboard <b>CPSwarm</b> <a href="https://www.cpswarm.eu/">https://www.cpswarm.eu/</a>		
Plan	Build	Completed
<a href="#">Code Generator</a>	 #2	2 weeks ago
<a href="#">Communication Library (Swarmio)</a>	 #12	2 weeks ago
<a href="#">Deployment Tool</a> 	 #80	2 weeks ago
<a href="#">Launcher</a>	 #18	1 hour ago
<a href="#">Modeling Tool (Modelio CPSwarm Extension)</a>	 #46	1 month ago
<a href="#">Optimization Tool (Frevo)</a>	 #69	1 month ago
<a href="#">Optimization Tool (Frevo) - Initialization</a>	 #27	1 month ago
<a href="#">Simulation and Optimization Orchestrator</a>	 #44	1 hour ago
<a href="#">Simulation Environment - Minisim Build and Test</a>	 #7	10 months ago

Figure 11. Screenshot of the build overview at the time of writing.

### 3.3 Continuous Delivery

In software engineering, continuous delivery (CD) is an approach which enables the release of software in short cycles, delivering latest features, improvements, and bug fixes. The CD integrates with CI and makes it easy to release the code that is added to version control and successfully built and tested. Figure 12 illustrates various cycles of integration and delivery.

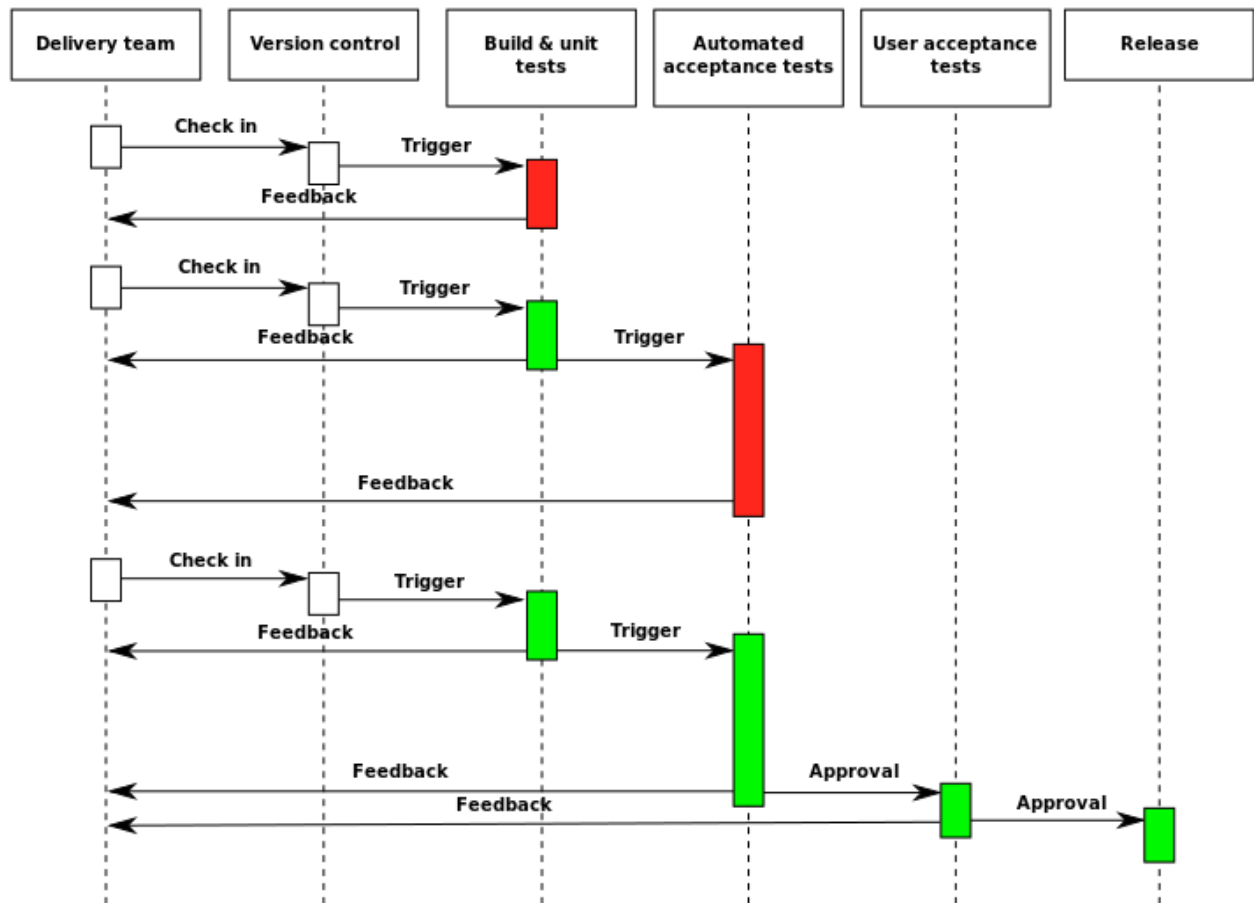


Figure 12. The process of continuous integration and delivery (CICD).<sup>5</sup>

In CPSwarm, the CD ensures that components can be released to users as quickly as possible, following a predefined process. A successful build will result in snapshot builds which only get released after approval of the developer. We only keep three builds on the server, however released builds are kept without any expiry. Certain components are also delivered in Docker Images for excellent portability across numerous platforms.

### 3.3.1 CPSwarm Launcher

The Launcher is built whenever changes are pushed to the corresponding Git repository. The builds happen in isolated Docker containers, producing executables for various platforms. As requested by the stakeholders, the system currently produces builds for the following platforms:

- Windows 10 – amd64
- Linux – amd64
- macOS – amd64

<sup>5</sup> [https://en.wikipedia.org/wiki/Continuous\\_delivery](https://en.wikipedia.org/wiki/Continuous_delivery)

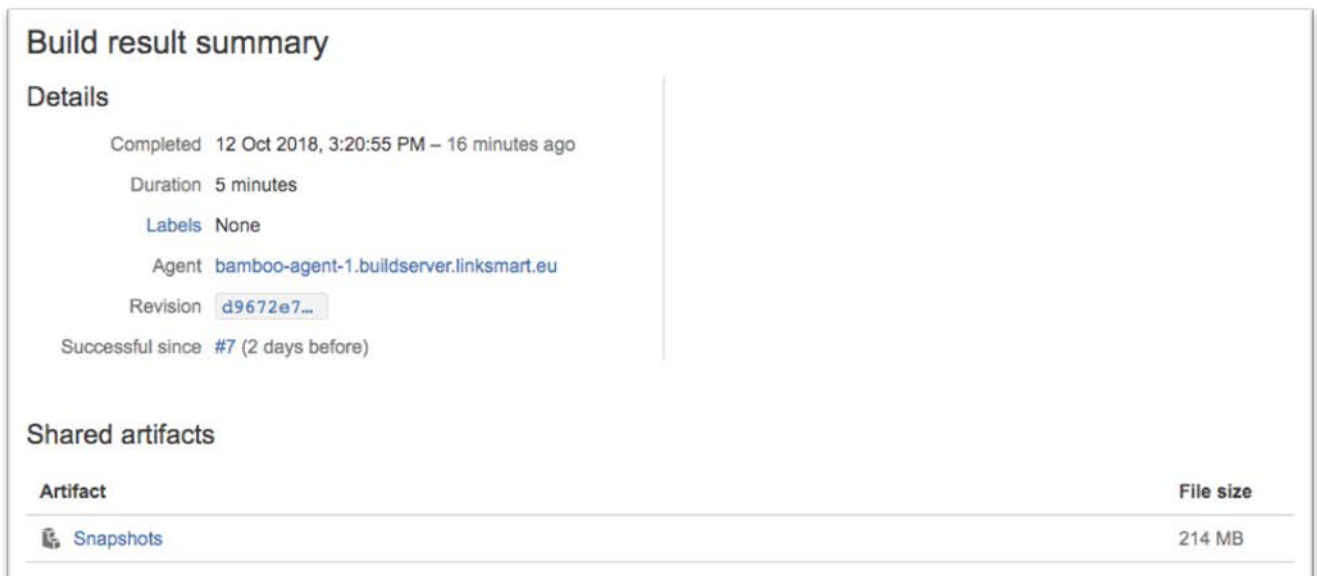


Figure 13. Build result summary for CPSwarm Launcher.

### 3.3.2 Simulation and Optimization Orchestrator

The builds on this component produce two artefacts, one with and another without dependencies. This component is developed in Java and shipped as jar artefacts for all platforms that meet the dependencies.

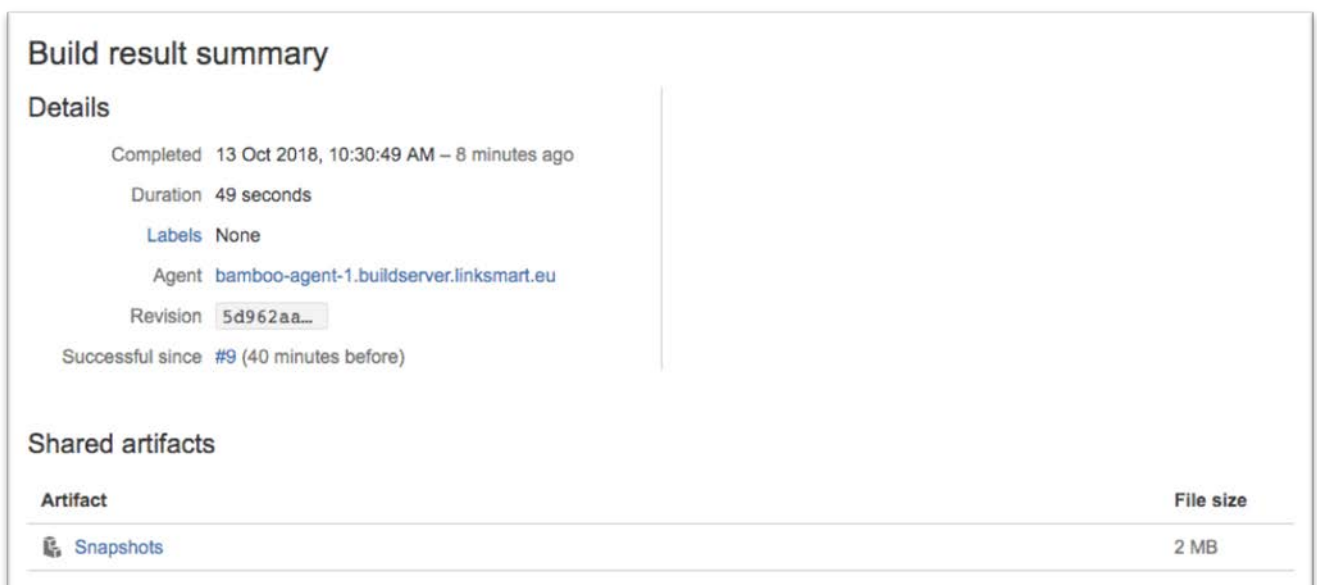


Figure 14. Build result summary for Simulation and Optimization Orchestrator.

### 3.3.3 Deployment Tool

The deployment tool CD involves producing snapshots for both Deployment Manager and Deployment Agent. For the Deployment Manager, we currently generate snapshots for the following platform:

- Linux – amd64 (Used also in Docker Image to support a wide range of operating systems)

For the Deployment Agent which is intended for target devices, we produce binary distributions as well as Debian packages for:

- Linux – armhf
- Linux – amd64

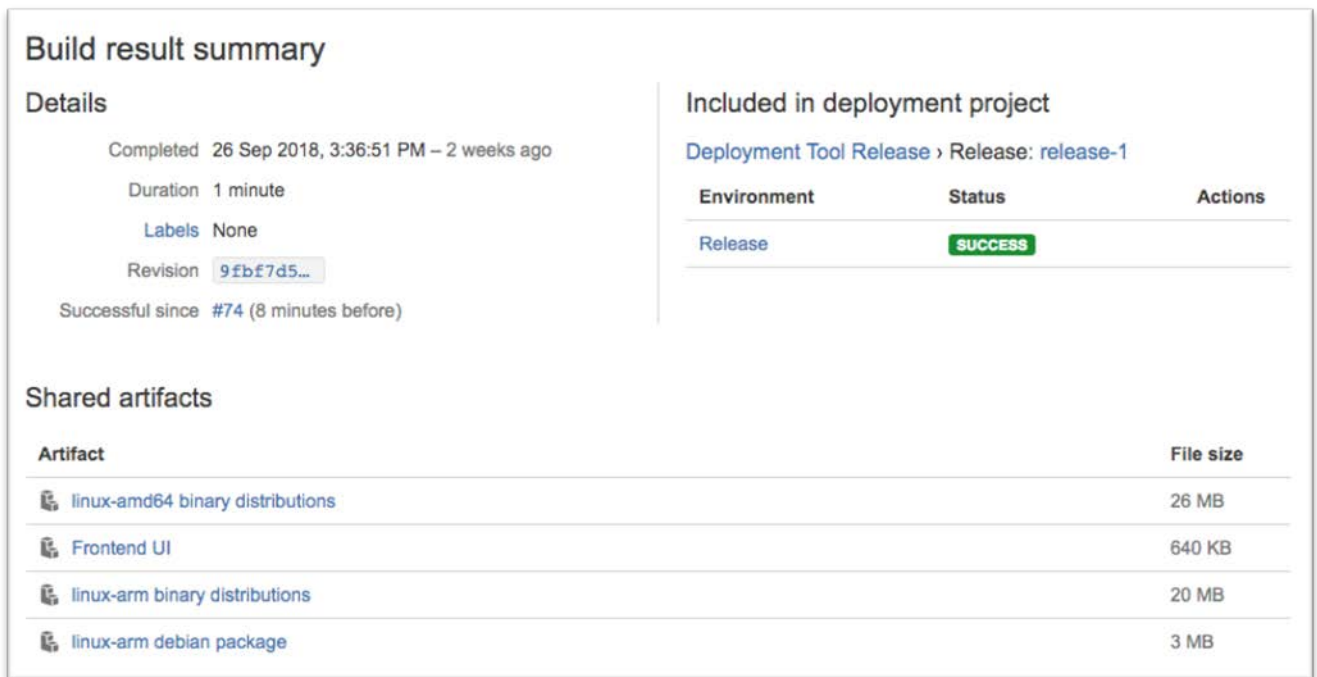


Figure 15. Build result summary for CPSwarm Deployment Tool.

### 3.3.4 Abstraction Library – Communication Library

The communication library is needed on target devices as well as on the device that hosts the monitoring tool. As such, we provide snapshots for a wider range of platforms:

- Linux – armhf
- Linux – amd64
- macOS – amd64
- Windows 10 – amd64

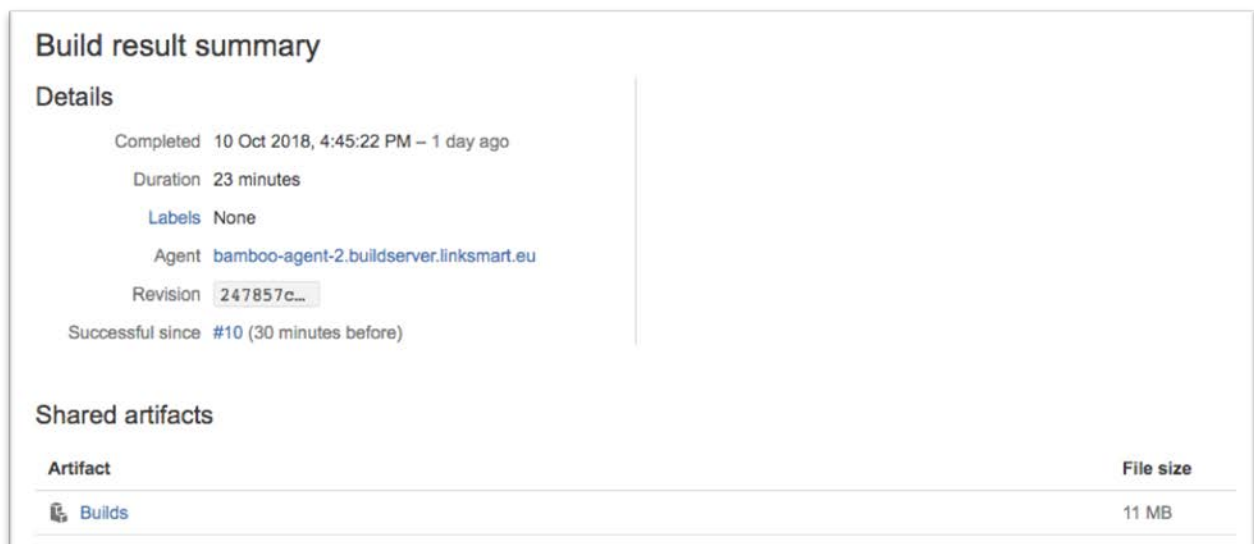


Figure 16. Build result summary for the communication library.

Sometime it is necessary to create different snapshots for Linux distributions or for specific ROS support. However, at the time of writing, the continuous delivery is not yet set up for those variants.

### 3.3.5 Modelling Tool

The Modelling Tool within the CPSwarm project includes the software Modelio and its CPSwarm extension. Modelio is a standalone desktop application that provides a UI for user to do generic modelling work. The CPSwarm extension is a plugin of Modelio which provides CPSwarm-related functionality. The CD hence includes two separate processes, one for Modelio and the other one for the CPSwarm extension.

Modelio has its own continuous delivery system controlled by its development team SOFTEAM, therefore is not included in the Bamboo continuous delivery plan. Figure 17 shows the Modelio artefact delivery page<sup>6</sup>. At the time of writing, Modelio supports the following platform:

- Windows 7/8/10 (32-bit and 64-bit)
- RedHat/CentOS (32-bit and 64-bit)
- Debian/Ubuntu (32-bit and 64-bit)
- Mac OS X (64-bit)

Latest release
Extensions
Previous versions
Alternative download links

### Download Modelio [Latest version: 3.8.0]

The latest version of Modelio 3.8.0 (Build 201810021138) is now available *(Last update on October 2nd, 2018)*.

Please select the right file for your system.

Platform	Architecture	File
		Modelio
Windows 7/8/10	64-bit	<a href="#">Modelio 3.8.0 - Windows 64-bit (309.53 MB)</a>
Windows 7/8/10	32-bit	<a href="#">Modelio 3.8.0 - Windows 32-bit (300.28 MB)</a>
RedHat/CentOS	64-bit	<a href="#">Modelio 3.8.0 - Red Hat/centOS 64-bit (307.97 MB)</a>
RedHat/CentOS	32-bit	<a href="#">Modelio 3.8.0 - Red Hat/CentOS 32-bit (313.76 MB)</a>
Debian/Ubuntu	64-bit	<a href="#">Modelio 3.8.0 - Debian/Ubuntu 64-bit (296.34 MB)</a>
Debian/Ubuntu	32-bit	<a href="#">Modelio 3.8.0 - Debian/Ubuntu 32-bit (301.66 MB)</a>
MacOS X	64-bit	<a href="#">Modelio 3.8.0 - MacOS X (172.63 MB)</a>

Figure 17. Modelio artefact delivery page.

Artefacts of the CPSwarm extension are produced by a Bamboo plan. The builds on this component produce the CPSwarm Extension Package, which can be loaded into Modelio during runtime. This component is developed in Java and shipped as jar artefacts for all platforms that meet the dependencies.

<sup>6</sup> <https://www.modelio.org/downloads/download-modelio.html>

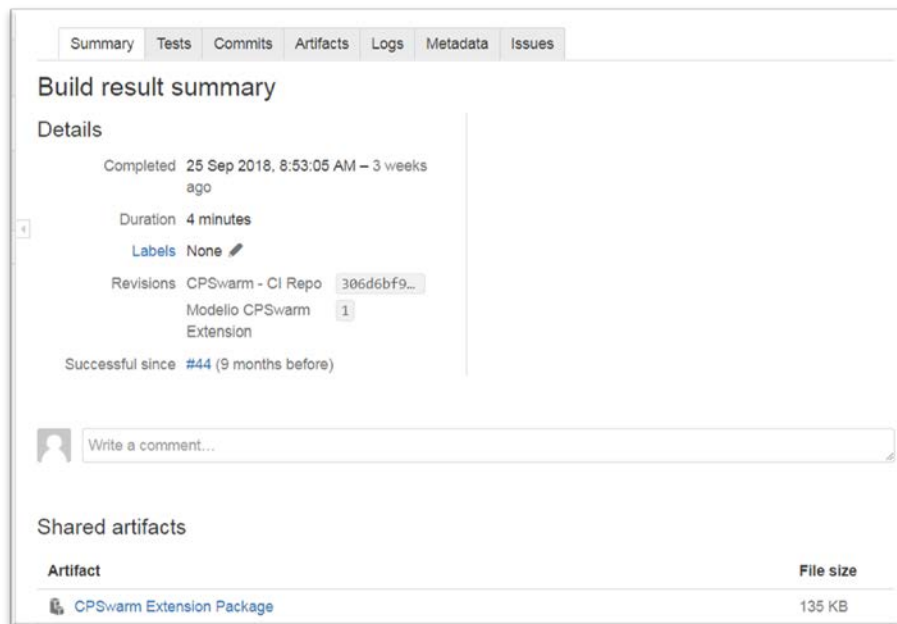


Figure 18. Build result summary for the Modelio CPSwarm extension.

### 3.3.6 Optimization Tool

The Optimization Tool (FREVO) has its own continuous delivery system managed by its development team at LAKE and UNI-KLU, hence is not included in the Bamboo continuous delivery plan. Figure 17 shows the FREVO artefact delivery page<sup>7</sup>. This component is developed in Java and shipped as jar artefacts for all platforms that meet the dependencies.

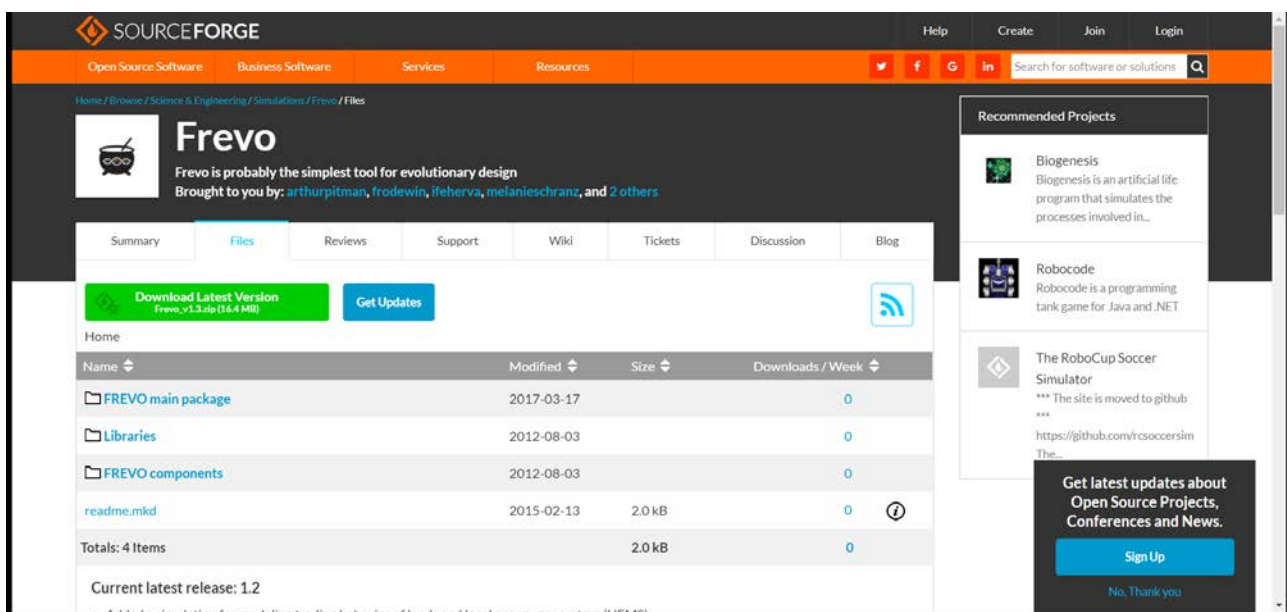


Figure 19. FREVO artefact delivery page.

<sup>7</sup> <https://sourceforge.net/projects/frevo/files/>

## 4 Conclusions

This deliverable is a report documenting the implementing of the updated CPSwarm Workbench and associated tools as well as the integration result of the second development phase in project CPSwarm. In this document, the available second phase components are reviewed. Details regarding the current setup for automatic component testing, continuous integration testing, build and test monitoring and continuous delivery are presented.

In the upcoming months, development of other CPSwarm components will continue and the second phase components will be further revised and modified according to newly identified requirements. The next integration phase will end in M34 of the CPSwarm project. By the end of M34, the deliverable 3.6 will be submitted. D3.6 will further document the final integration results of the CPSwarm components.

## Acronyms

Acronym	Explanation
API	Application Programming Interface
BPMN	Business Process Model and Notation
CD	Continuous Delivery
CI	Continuous Integration
CPDT	CPS Population Design Tool
FREVO	Framework for evolutionary design
MARTE	Modeling and Analysis of Real-time and Embedded systems
VM	Virtual Machine

## List of figures

Figure 1. Updated architecture design for CPSwarm system (Extracted from D3.2).....	5
Figure 2. State-machine modelled with Modelling Tool (Extracted from D5.3).....	6
Figure 3. Example of hardware components (Extracted from D4.2).....	7
Figure 4. Abstraction Library model example (Extracted from D4.2).....	8
Figure 5. The CPSwarm Abstraction Library (Extracted from D4.1). ....	9
Figure 6. Deployment Tool architecture (Extracted from D7.3). ....	10
Figure 7. CPSwarm Launcher screenshot.....	11
Figure 8. Integration test process, validating the interaction among Simulation Optimization Orchestrator, Simulation Manger and Optimization Tool .....	14
Figure 9. End-to-end test process, validating the behaviour and compliance of Deployment Tool to the specification.....	15
Figure 10. Sample notification settings for a plan (i.e. Launcher) .....	17
Figure 11. Screenshot of the build overview at the time of writing. ....	17
Figure 12. The process of continuous integration and delivery (CICD). ....	18
Figure 13. Build result summary for CPSwarm Launcher. ....	19
Figure 14. Build result summary for Simulation and Optimization Orchestrator. ....	19
Figure 15. Build result summary for CPSwarm Deployment Tool. ....	20
Figure 16. Build result summary for the communication library. ....	20
Figure 17. Modelio artefact delivery page. ....	21
Figure 18. Build result summary for the Modelio CPSwarm extension.....	22
Figure 19. FREVO artefact delivery page.....	22



## References

- [1] Atlassian, "Bamboo Best Practice - Using Stages," 11 2017. [Online]. Available: <https://confluence.atlassian.com/bamboo/bamboo-best-practice-using-stages-388401113.html>.
- [2] Atlassian, "Atlassian Bamboo Open Source Project License Request," 11 2017. [Online]. Available: <https://www.atlassian.com/software/views/open-source-license-request>.
- [3] X.-S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008.
- [4] H. H. Thomas Schmickl, "Bio-inspired Computing and Networking," CRC Press, 2011, pp. 95-137.
- [5] X.-S. Yang, "Firefly Algorithms for Multimodal Optimization," *Stochastic Algorithms: Foundations and Applications*, pp. 169-178, 2009.
- [6] A. Sobe, I. Fehévari and W. Elmenreich, "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, 2012.