



D3.6 – FINAL CPSWARM WORKBENCH AND ASSOCIATED TOOLS

| | |
|---------------------|---|
| Deliverable ID | D3.6 |
| Deliverable Title | Final CPSwarm Workbench and associated tools |
| Work Package | WP3 – Architecture design and Component Integration |
| Dissemination Level | PUBLIC |
| Version | 1.1 |
| Date | 2019-11-30 |
| Status | Final |
| Lead Editor | Farshid Tavakolizadeh (FRAUNHOFER) |
| Main Contributors | Farshid Tavakolizadeh, Junhong Liang (FRAUNHOFER), Davide Conzon, Gianluca Prato (LINKS), Ákos Milánkovich (SLAB), Andreas Eckel (TTTECH), Melanie Schranz, Micha Rappaport (LAKE), Etienne Brosse (SOFTTEAM), Arthur Pitman (UNI-KLU) |

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 | 2019-10-01 | Farshid Tavakolizadeh (FRAUNHOFER) | First Draft with TOC and initial content |
| | 2019-10-02 | Ákos Milánkovich (SLAB) | Added manual test steps for communication library |
| 0.2 | 2019-10-02 | Farshid Tavakolizadeh (FRAUNHOFER) | Added CICD platform info |
| 0.3 | 2019-10-04 | Farshid Tavakolizadeh (FRAUNHOFER) | Updated build and release plans for launcher and deployment tool |
| 0.4 | 2019-10-05 | Farshid Tavakolizadeh (FRAUNHOFER) | Updated intro, added swarm deployment workflow. Added missing subchapters for continuous delivery of components |
| | 2019-10-07 | Davide Conzon (LINKS) | Described the simulation and optimization stage. Updated description of SOO and SM tests |
| 0.5 | 2019-10-07 | Farshid Tavakolizadeh (FRAUNHOFER) | Merged continuous delivery and CI chapters. Described continuous delivery of several components |
| 0.6 | 2019-10-07 | Farshid Tavakolizadeh (FRAUNHOFER) | Updated continuous delivery of SOO and modelling tool |
| | 2019-10-08 | Andreas Eckel (TTTECH) | Added description of monitoring phase. Described monitoring tool's test strategy. |
| 0.7 | 2019-10-08 | Farshid Tavakolizadeh (FRAUNHOFER) | Minor modification and comments on the added contributions |
| | 2019-10-08 | Andreas Eckel (TTTECH) | Addressed comments, updated test description of monitoring tool |
| 0.8 | 2019-10-08 | Farshid Tavakolizadeh (FRAUNHOFER) | Minor modifications to contributed sections. Finalized continuous delivery section for communication library. |
| | 2019-10-09 | Melanie Schranz (LAKE) | Described the test strategy of swarm library |
| | 2019-10-10 | Ákos Milánkovich (SLAB) | Described the test plan for communication library |
| 0.9 | 2019-10-10 | Farshid Tavakolizadeh (FRAUNHOFER) | Integrated and polished the contributions |

| | | | |
|------|------------|------------------------------------|---|
| | 2019-10-14 | Micha Rappaport (LAKE) | Described the release plan of swarm library |
| | 2019-10-15 | Arthur Pitman (UNI-KLU) | Added test plan of Optimization Tool components |
| | 2019-10-15 | Gianluca Prato (LINKS) | Described code generation Added test plan for abstraction library Updated test plan of code generator |
| | 2019-10-16 | Etienne Brosse (SOFTTEAM) | Described modelling phase Updated test plans of modelling tool and modelling library |
| 0.10 | 2019-10-16 | Farshid Tavakolizadeh (FRAUNHOFER) | Integrated and polished the final contributions. Release for internal review. |
| 1.0 | 2019-11-12 | Farshid Tavakolizadeh (FRAUNHOFER) | Addressed internal review comments |
| | 2019-11-18 | Davide Conzon (LINKS) | Updated SOO description and presented new tests |
| | 2019-11-28 | Ákos Milánkovich (SLAB) | Documented latest tests of the communication library |
| | 2019-11-29 | Etienne Brosse (SOFTTEAM) | Updated monitoring tool's build and test plan |
| | 2019-11-29 | Gianluca Prato (LINKS) | Updated code generator's description and test plan |
| 1.1 | 2019-11-30 | Farshid Tavakolizadeh (FRAUNHOFER) | Updated launcher screenshots Added description for the continuous delivery of monitoring tool Prepared for submission |

Internal Review History

| Review Date | Reviewer | Summary of Comments |
|-------------|------------------------|--|
| 2019-10-17 | Andreas Eckel (TTTech) | Only minor typos detected and corrected. |
| 2019-11-07 | Omar Morando (DGSKY) | Only minor changes. |

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 1.1 | <i>Related documents</i> | 5 |
| 2 | CPSwarm Workbench and associated tools..... | 6 |
| 2.1 | <i>Swarm Modelling</i> | 7 |
| 2.2 | <i>Simulation & Optimization.....</i> | 9 |
| 2.3 | <i>Code Generation</i> | 12 |
| 2.4 | <i>Swarm Deployment</i> | 14 |
| 2.5 | <i>Monitoring and Command</i> | 16 |
| 3 | Continuous Integration and Delivery (CICD) | 18 |
| 3.1 | <i>CICD Platform.....</i> | 18 |
| 3.2 | <i>Platform Guidelines</i> | 20 |
| 3.3 | <i>CICD Practices</i> | 21 |
| 3.3.1 | Automated Builds and Tests..... | 21 |
| 3.3.2 | Continuous Delivery..... | 21 |
| 3.3.3 | Build and Test Monitoring | 22 |
| 3.4 | <i>CICD Plans</i> | 23 |
| 3.4.1 | Launcher | 24 |
| 3.4.2 | Modelling Tool..... | 25 |
| 3.4.3 | Modelling Library | 27 |
| 3.4.4 | Behavior Library..... | 28 |
| 3.4.4.1 | Abstraction Library..... | 28 |
| 3.4.4.2 | Swarm Library..... | 28 |
| 3.4.4.3 | Communication Library | 29 |
| 3.4.5 | Simulation & Optimization Orchestrator | 30 |
| 3.4.1 | Optimization Tool | 33 |
| 3.4.2 | Simulation Manager | 35 |
| 3.4.3 | Code Generator | 38 |
| 3.4.4 | Deployment Tool..... | 39 |
| 3.4.5 | Monitoring and Command Tool | 43 |
| 4 | Conclusions | 45 |
| | Acronyms | 46 |
| | List of Figures | 47 |
| | References | 48 |

1 Introduction

The CPSwarm Workbench and associated tools have significantly evolved over the course of the project. The evolutions have resulted in a system with better integration and individual components with high potential for exploitability beyond the CPSwarm project.

The deliverable series for CPSwarm Workbench and associated tools presents an overall picture of the CPSwarm system and underlying development practices. The initial deliverable (D3.4) described the methodology and infrastructure for continuous integration (CI) of the CPSwarm software. In addition, the set up for the initial phase of the CPSwarm components integration was presented in the same document. The second deliverable (D3.5) focused on documenting the second phase integration of the CPSwarm components. This document is the last deliverable in the series which presents the final CPSwarm workflow and the integration at the time of writing. Minor improvements are expected in the following months within and beyond the scope of project. However, such modifications would not affect the overall workflow of the CPSwarm system.

Chapter 2 summarizes the current state of the components and provides a walkthrough of the CPSwarm system. Consecutively, chapter 3 describes the build, test, and release plan as part of the continuous integration and delivery practices of the project.

1.1 Related documents

| ID | Title | Date |
|------|---|------|
| D3.3 | Final System Architecture Analysis & Design Specification | M30 |
| D4.1 | Initial CPSwarm Modeling Library | M09 |
| D4.2 | Updated CPS modeling library | M21 |
| D4.3 | Final CPS modeling library | M33 |
| D4.4 | Initial Swarm modeling library | M10 |
| D4.5 | Updated Swarm modeling library | M22 |
| D4.6 | Final Swarm modelling library | M34 |
| D5.2 | Initial Swarm Modelling Tool | M09 |
| D5.3 | Updated CPSwarm Modelling Tool | M18 |
| D5.4 | Final CPSwarm Modelling Tool | M34 |
| D6.1 | Initial Simulation Environment | M09 |
| D6.2 | Final Simulation Environment | M28 |
| D6.5 | Initial Integration of external simulators | M18 |
| D6.6 | Updated Integration of external simulators | M28 |
| D6.7 | Final Integration of external simulators | M36 |
| D7.1 | Initial CPSwarm Abstraction Library | M18 |
| D7.2 | Final CPSwarm Abstraction Library | M32 |
| D7.3 | Initial Bulk deployment tool | M21 |
| D7.4 | Final Bulk deployment tool | M33 |
| D7.5 | Initial Monitoring and configuration framework | M22 |
| D7.6 | Final Monitoring and configuration framework | M34 |

2 CPSwarm Workbench and associated tools

The final architecture design for the CPSwarm system was presented in D3.3 and is shown Figure 1. The final architecture reflects the latest design of the CPSwarm system with the following set of components:

- Modelling Tool (documented in D5.2-D5.4)
- Modelling Library (documented in D4.1-D4.3)
- Behavior Library
 - Abstraction Library (documented in D7.1-D7.2)
 - Swarm libraries (documented in D4.4-D4.6)
- Simulation & Optimization Orchestrator, Optimization Tool and Simulation Manager (documented in D6.1-D6.2 and D6.5-D6.7)
- Code Generator (documented in D5.3-D5.4)
- Deployment Tool (documented in D7.3-D7.4)
- Monitoring Tool (documented in D7.5-D7.6)
- Launcher (documented in in D3.2-D3.3 as well as this deliverable series)

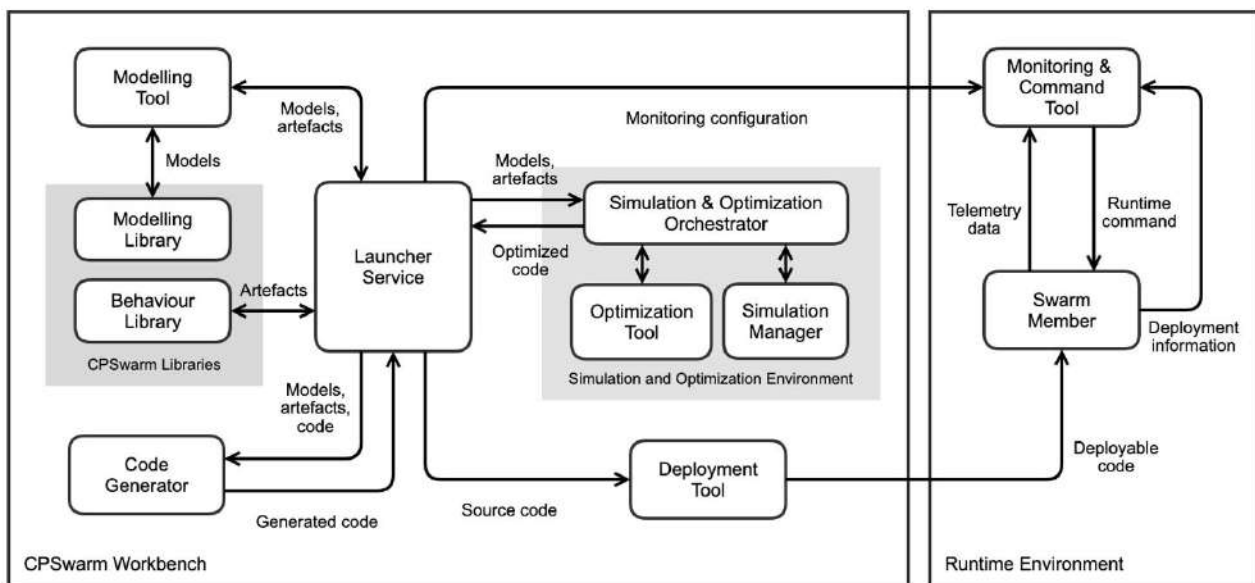


Figure 1. Final architecture design of the CPSwarm system (Extracted from D3.3).

This design divides the CPSwarm system into two logical groups based on the deployment model. The components that are used during design time are part of the CPSwarm Workbench. On the other hand, the Monitoring & Command Tool and Swarm Member are considered as part of the Runtime Environment.

The Launcher Service is a thin, central component that acts as a portal and starting point of interaction with other components of the system. The Launcher Service manages inputs and outputs of the component and offers a minimalistic user interface to guide the user toward different design steps and to set components configurations for launching different components.

The rest of this section provides a walkthrough of the CPSwarm system starting from the launcher and all the way to deploying software and monitoring the swarm runtime.

2.1 Swarm Modelling

During the Swarm Modelling phase, CPSwarm workbench guides the user to define/model several aspects of CPS swarm. The launcher helps the user to manage its different modelling projects by selecting an existing project or creating a new one and open it as depicted in Figure 2.

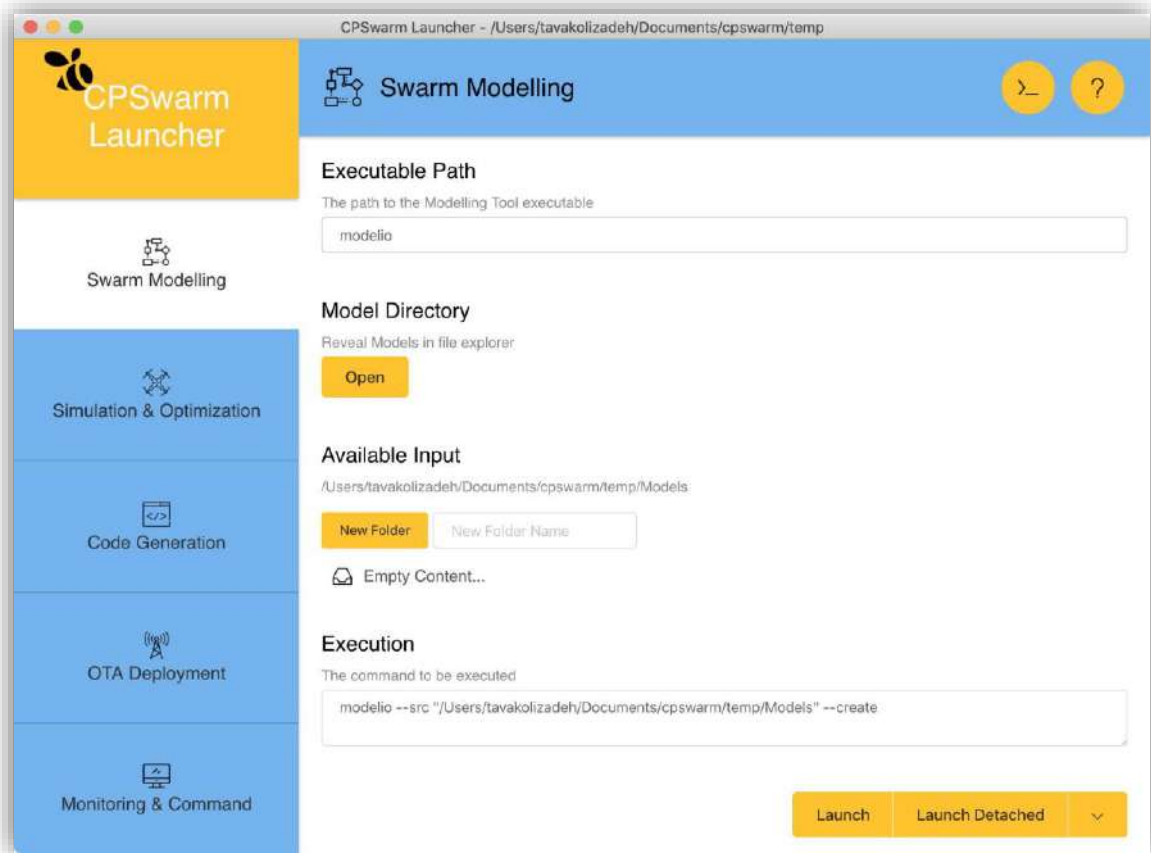


Figure 2. Screenshot of the Swarm Modelling tab in the Launcher.

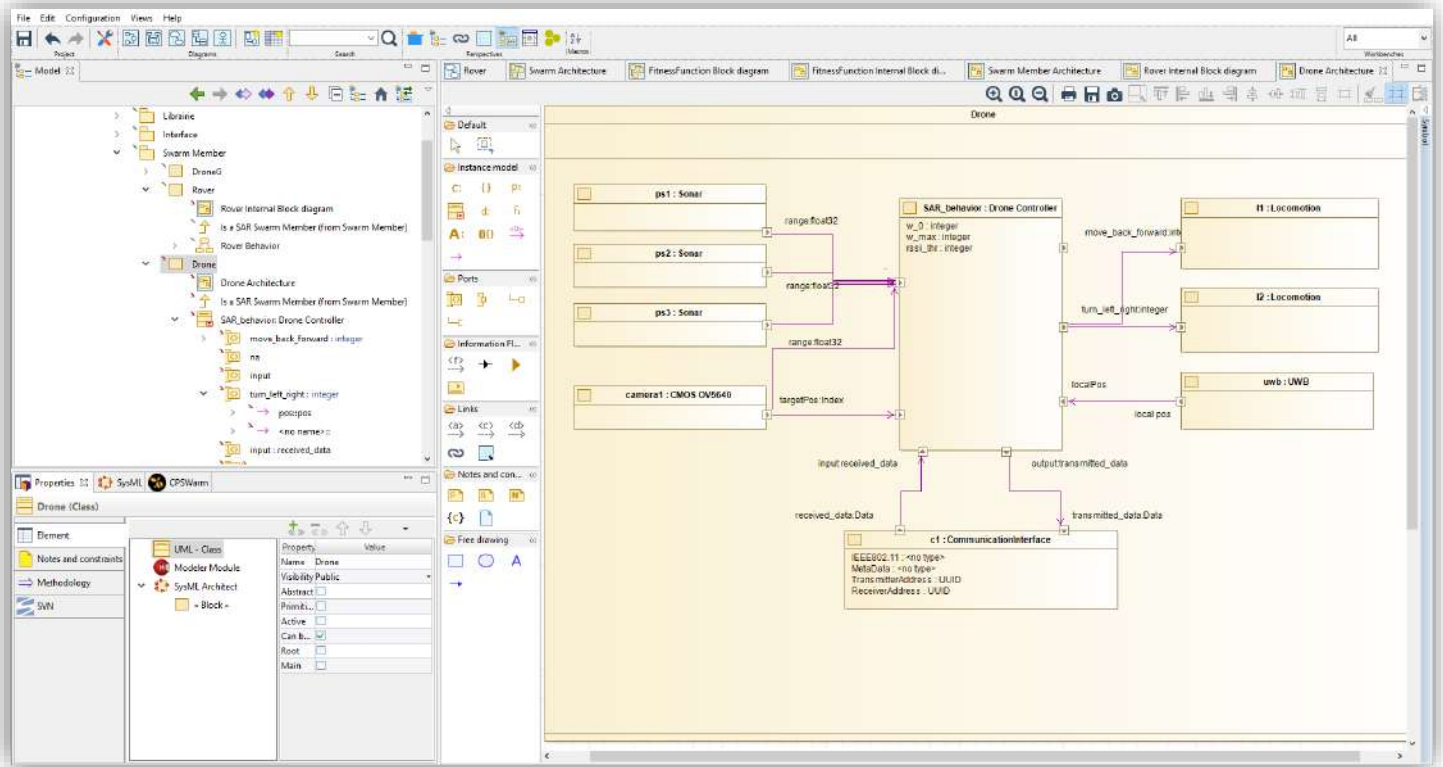


Figure 3. Screenshot of the Modelling Tool while designing the architecture of a drone.

Once a Modelling project is opened (see Figure 3), the user can define or update one of the CPS swarm aspect by using CPSwarm Modelling language (described in D5.1 and its updates). Among these aspects, the first one can be the swarm behavior. In CPSwarm project, behaviors are modelled as UML state machine (see Figure 4).

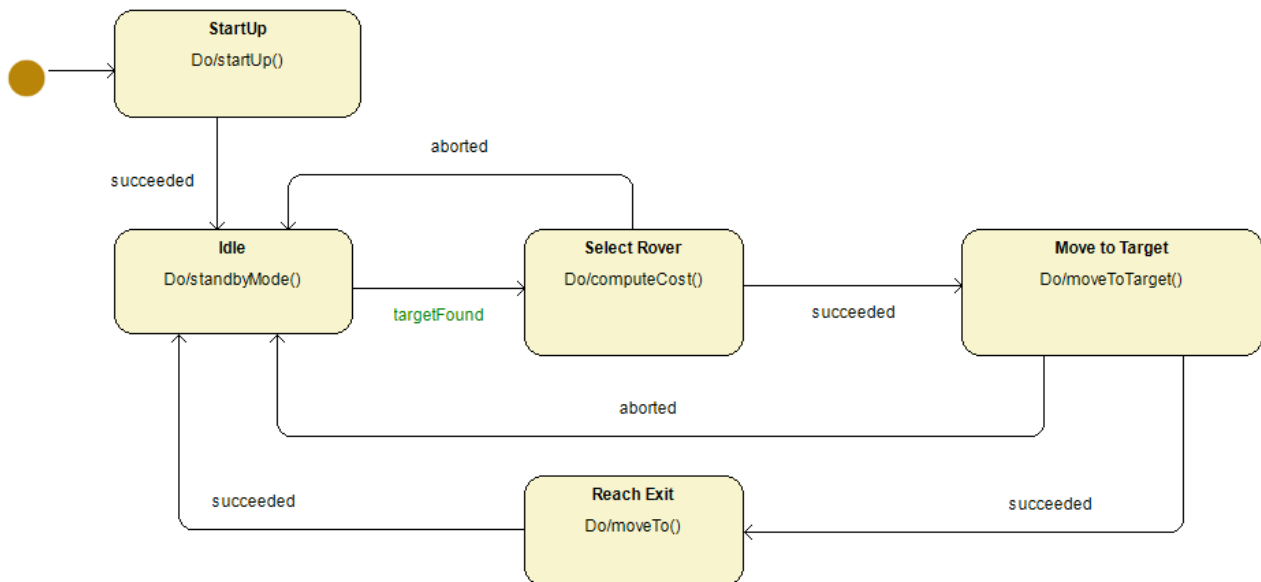


Figure 4. CPSwarm Behavior example

This behavior can be exported as SCXML file for a future Code Generation as described in Section 2.3. Modelling tool also allows to define both swarm composition in terms of CPS as depicted in Figure 5 and the CPS swarm goal (a.k.a. fitness function) as shown in Figure 6.

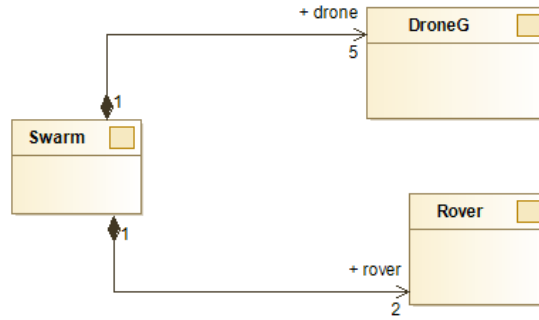


Figure 5. Swarm Composition Modelling

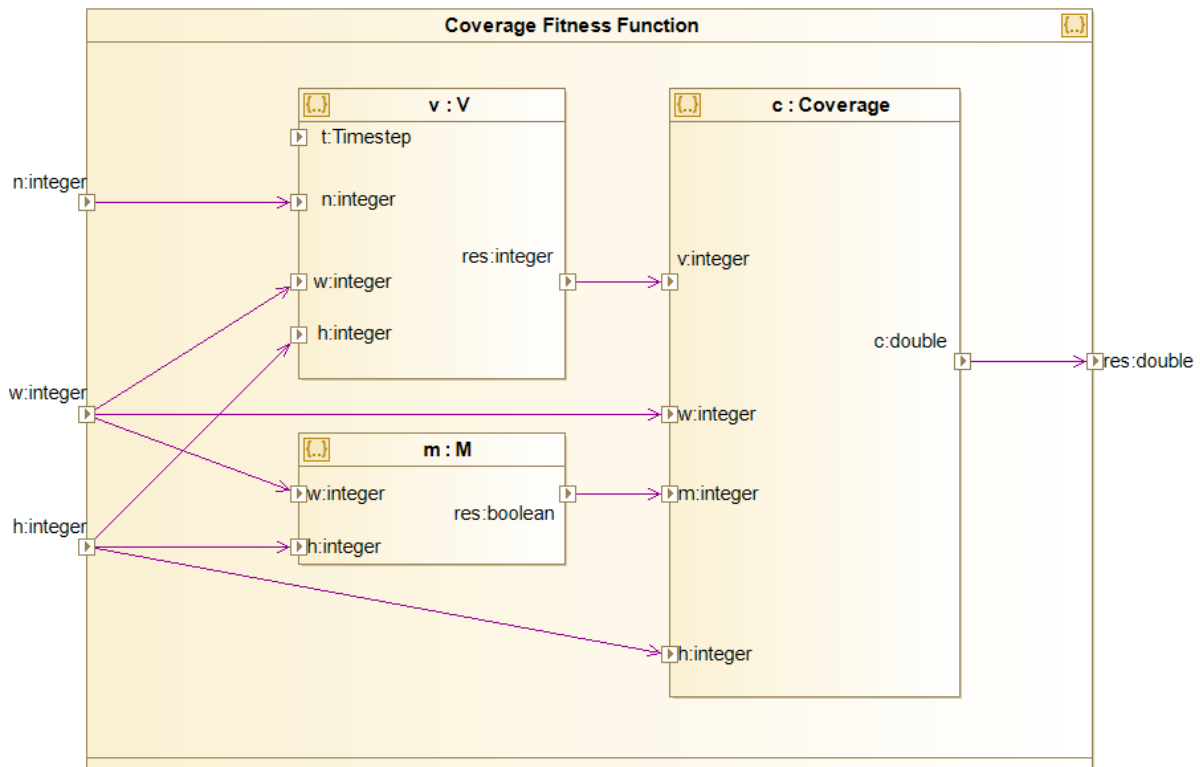


Figure 6. Fitness Function Modelling

These two aspects can be exported to be used during the Simulation and Optimization step described in the following section.

2.2 Simulation & Optimization

The Simulation and Optimization step allows to test the swarm modelled with Modelling Tool (see Section 2.1). The Launcher can be used to run the Simulation and Optimization Orchestrator (SOO) as shown in Figure 7.

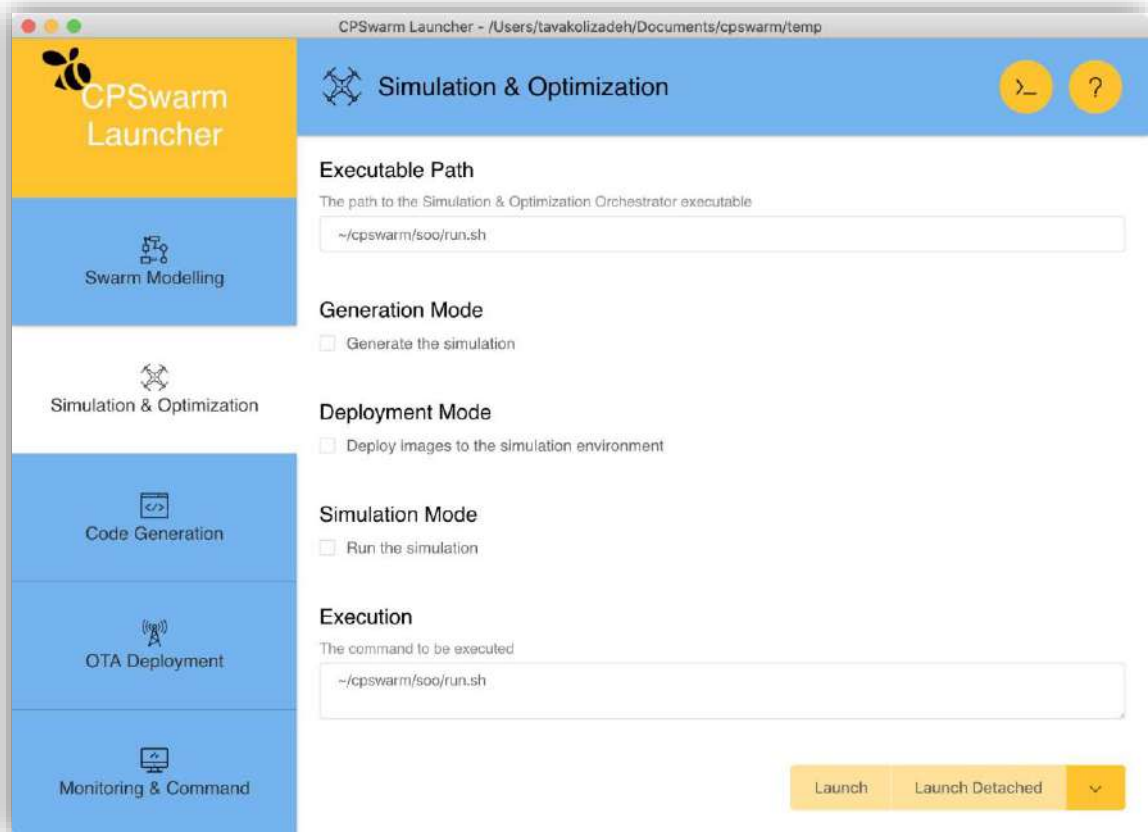


Figure 7. Screenshot of the Simulation & Optimization tab in Launcher.

The main parameters that the user can indicate using the Launcher interface are the following:

- Mode in which the SOO can be run:
 - Generation Mode: the SOO is used to generate the simulation code to be used to run the desired simulation.
 - Deployment Mode: the SOO is used to deploy the Simulation Managers (SM)s and eventually the Optimization Tool (OT), in the nodes of the cluster. This mode depends on a Kubernetes¹ cluster, pre-installed by the user.
 - Running Mode: the SOO is run using the SMs already installed to execute the simulation or optimization.
 - Both Deployment and Running Modes: the SOO deploys the needed SMs in the cluster and then executes the required task.
- The type of the task:
 - Simulation: the default mode in which the SOO simulate the behavior in one SM.
 - Optimization: using an OT, the behavior is optimized running multiple simulation distributed on the available SMs suitable for the required simulation task.
- Parameters to be set for the OT (only in case of optimization).
- Requirements needed to run the task (used to select the more suitable simulators to be used), like number of dimension supported (2D/3D), or the maximum number of agents.

The other important inputs for the SOO are the outputs of Modelling Tool: like the models, the swarm description and in case of optimization the fitness function to be used to optimize the algorithm. These inputs are taken by the SOO from the configuration directories of the Launcher.

¹ <https://kubernetes.io/>

As described in D6.2, the CPSwarm Simulation and Optimization Environment is a distributed environment, where the SOO starts the task, which is executed on a set of distributed Simulation Tools, each one wrapped by a SM, with the use of an OT, if the algorithm must be optimized. The components of the Simulation and Optimization Environment communicate to each other, using the XMPP protocol², specifically the software has been tested with two XMPP server releases: Openfire³ and Tigase⁴, to use the Simulation and Optimization Environment, one instance of these servers must be installed and be accessible by all the distributed machines used.

In case of optimization, the SOO initially configure the OT and the available SMs passing to them all the files produced by the Modelling Tool and the parameters indicated by the user through the Launcher. Then the OT starts the optimization task, to do so, the OT needs to run a high number of simulations to evaluate the best combination of parameters to set. The distributed nature of the Simulation and Optimization Environment allows to run such simulation in parallel using the integrated SMs. This allows reducing the tasks required by the optimization tasks, which are very long processes. During the optimization, the process can be monitored using the data provided by the SMs (see Figure 8 for a chart generated by Thingsboard⁵ to monitor a running optimization).

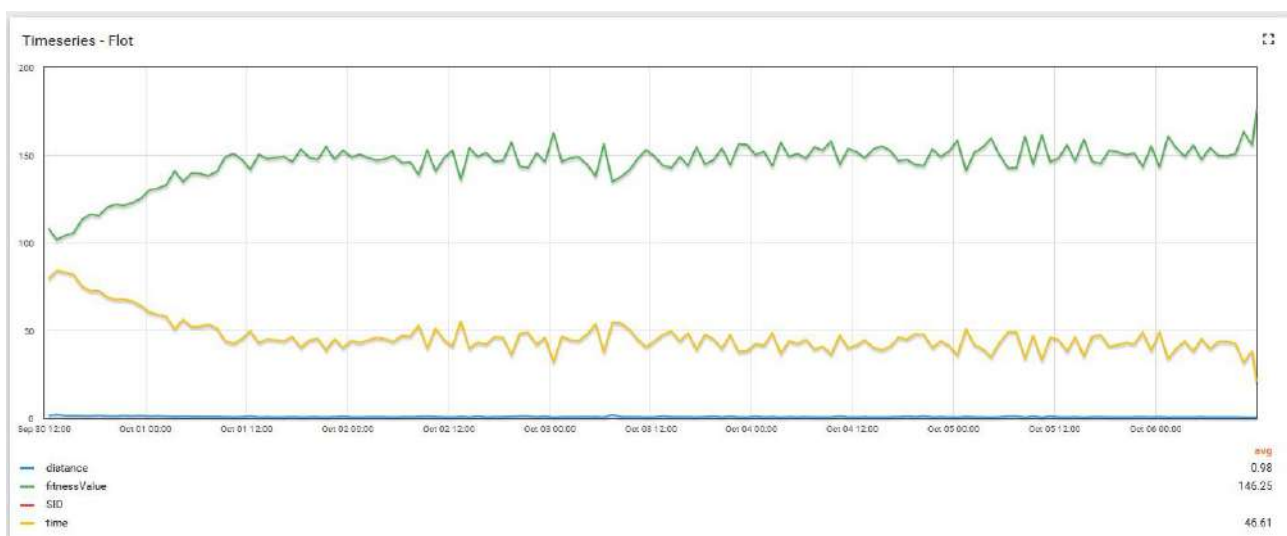


Figure 8. Optimization Monitoring

The output of this optimization is the combination of parameters to be set to obtain the optimized behavior, these values will be then deployed on the CPS by the Deployment Tool. Furthermore, if the SMs are deployed using Kubernetes, through the Launcher, the user can access also to a dashboard⁶ that can be used to monitor the resources on the cluster's nodes (see Figure 9) and the status of the SMs deployed (see Figure 10).

² <https://xmpp.org/>

³ <https://www.igniterealtime.org/projects/openfire/>

⁴ <https://tigase.net/content/tigase-xmpp-server>

⁵ <https://thingsboard.io/>

⁶ <https://github.com/kubernetes/dashboard>

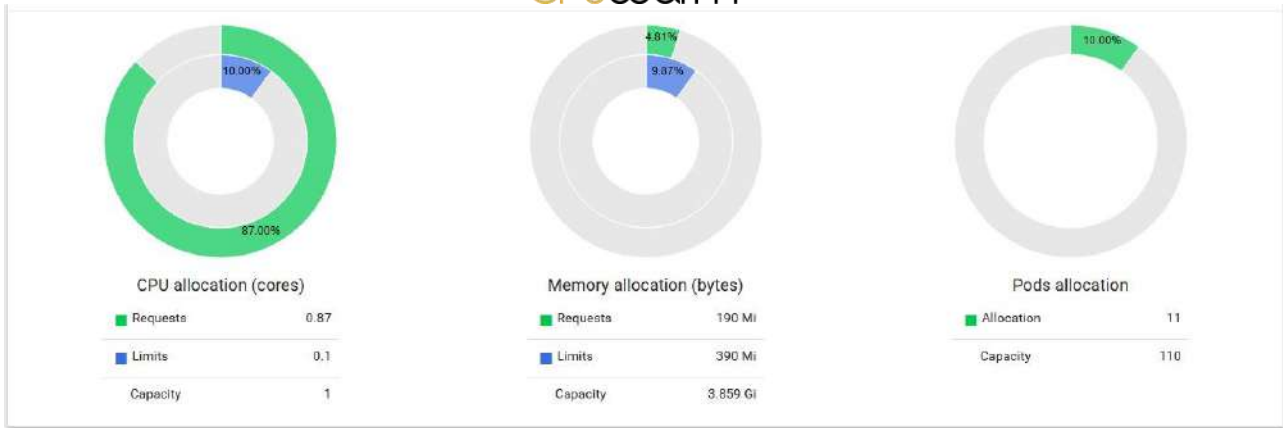


Figure 9. Cluster resources allocation

| Pods | | | | | |
|------------------------|------------------------|---------|----------|---------|--|
| Name | Node | Status | Restarts | Age | |
| stage-8469b6bbc7-kv5rd | sim-mac-fal | Running | 0 | 6 days | |
| frevo-5b4869fd4b-gg6nl | pert-demoenergy-virtus | Running | 0 | 22 days | |

Figure 10. Simulation Managers deployed status

The Simulation and Optimization Environment can be used also to test a controller in a simulation tool, through its GUI. In this case, the SOO configures one available SM to run the simulation and then run it. In this way, the user can evaluate the behavior on the simulator's Graphic User Interface (GUI) (see Figure 11).

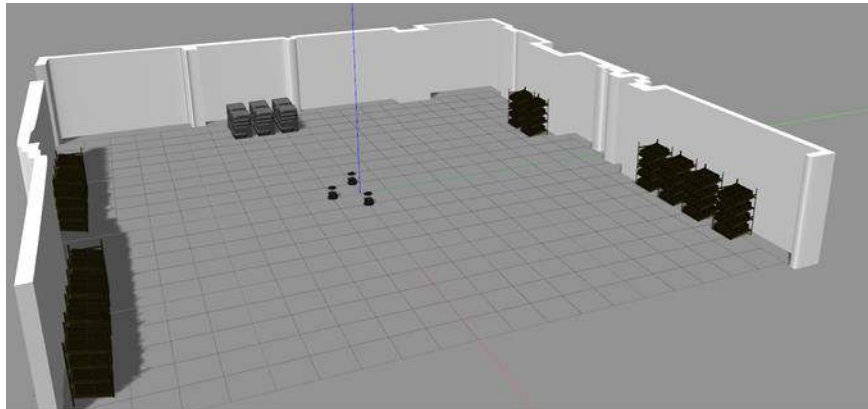


Figure 11. Logistic scenario simulation in Gazebo. (source: D6.6)

2.3 Code Generation

The Code Generator has a central role in driving the flow of the toolchain from the modeling part to the deployment of the code on the actual CPS. Inside the CPSwarm Workbench, the Code Generator is responsible to translate design-level modeled behaviors into concrete and executable code. Such executable code relies on two of set libraries and functionalities so called Behavior Library (described in D4.3 and D7.2).

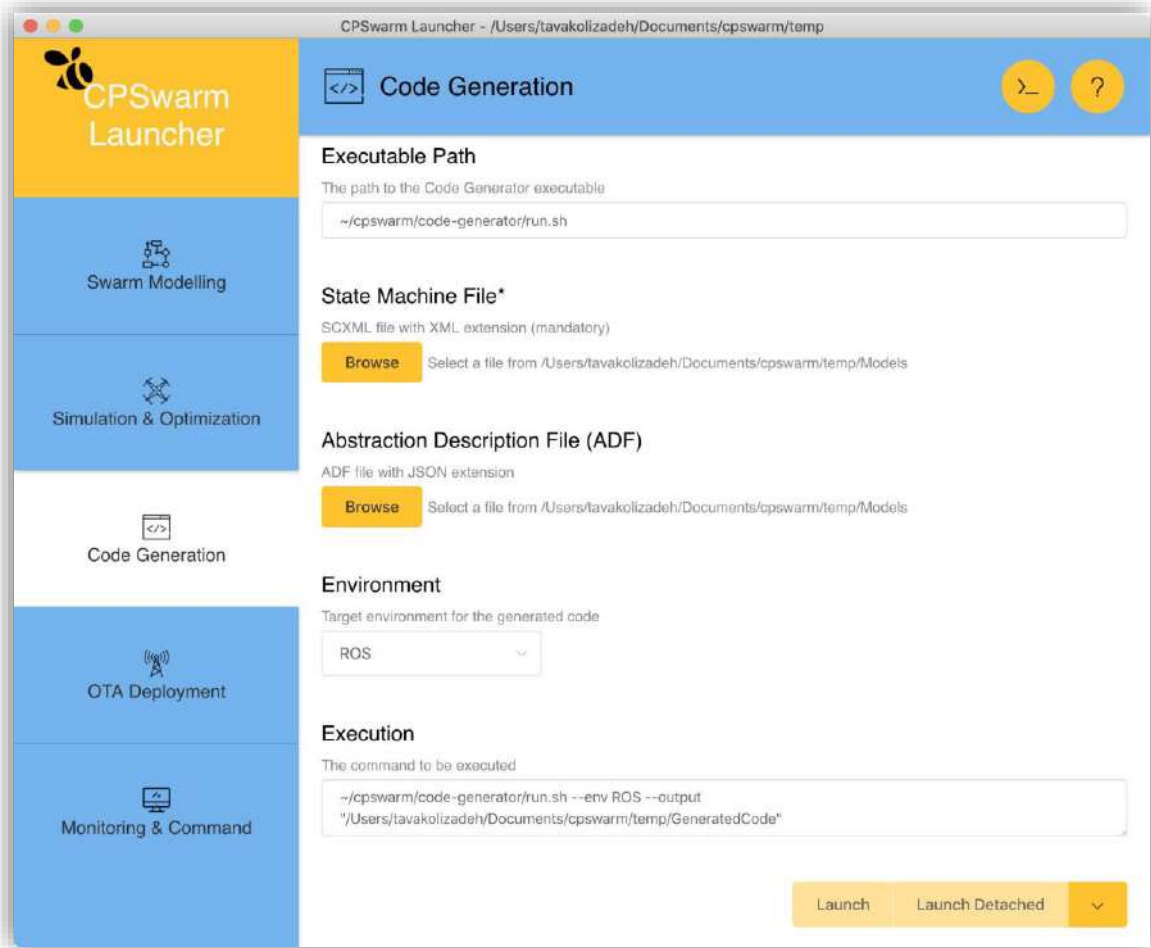


Figure 12. Code Generation tab in the Launcher.

In order to run a code generation process, the user has to specify the following parameters:

- The directory containing the inputs produced by the modeling phase (see Section 2.1). This path to the selected directory can be specified using the Launcher.
- The directory where the outputs of the Code Generator should be collected. Configurable through the Launcher, as before.
- The target Runtime Environment. This parameter is optional and is set to “ROS” by default, as it is the only one supported in the current version of the Code Generator.

The Code Generator supports the generation of code starting from the description of an algorithm in the form of a Finite State Machine (FSM). This state machine can be provided as input to the Code Generator using a standard data format called SCXML. Due to the very schematic and repeatable structure that all algorithms defined using a FSM have, the template-based technique has been selected as the most proper approach to complete the code generation task. Indeed, template-based generation applies whenever it is possible to define a simple set of target-templates to be filled with data extracted from the algorithm specification.

In addition to this, the Code Generator provides the possibility to generate a simple pre-configured template of a new CPS algorithm whose inputs and outputs have been designed into the Modeling Tool. The generated code cannot be directly deployed to the actual CPS and has to be completed and refined by the developer.

2.4 Swarm Deployment

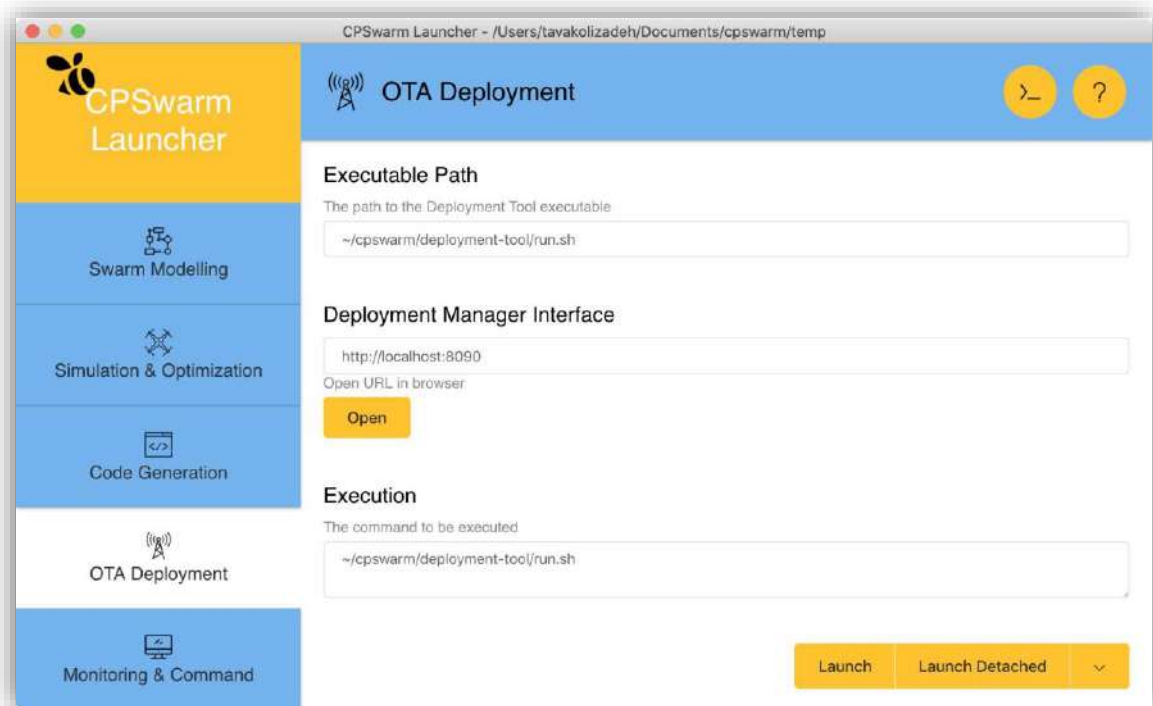


Figure 13. Swarm Deployment tab in the Launcher.

The swarm deployment stage is where the source codes, configuration files, or artifacts are transferred to swarm devices. The source codes are generated codes as well as library files and other handwritten pieces of code. Configuration files are generated after modelling or provided by the software developer or the deployer. Lastly, artifacts are packages which include software or other resources. Once the necessary files are ready for deployment, the user can launch the server component of the Deployment Tool and then open its web interface.

The web interface of the Deployment Tool enables the user to see available swarm devices or initiate the secure registration of new ones. The user may update the meta information about the devices, see the current running applications, and monitor the status of the devices in a list or on the map. Figure 14 is a screenshot of the device management page on the Deployment Tool's web interface.



Figure 14. Deployment Tool's web interface for viewing the devices and their status.

The user may perform the deployment by selecting input files from the local machine, providing deployment configurations, and selecting the appropriate set of targets. The Deployment Tool provides two deployment methods, one for building artifacts, and another for installing and running the software. The left side of Figure 15 is the screenshot of deployment configuration layout. For builds, the user may select the input files, provide necessary build commands, and select a host device to perform it. The host device should be one of the registered devices. It is possible to chain the build to the installation and runtime by also providing the necessary commands and the target group. The target group can be set by entering device names, the tags to pick a whole group, or by selecting one or groups of devices on the map. The build steps can be skipped when a deployment does not require any initial packaging or compilation. After completing the configuration, the user may press the deploy button to trigger the deployment and view the progress. Figure 15 shows the progress three on the right side, when the build is successfully over, one device is still at installation stage, six devices are running the software successfully, and four others has failed to run it. Three of the failed devices are grouped together because the runtime errors have been similar. From there, the user may request runtime logs, terminate the deployment, or repeat it after making modifications.

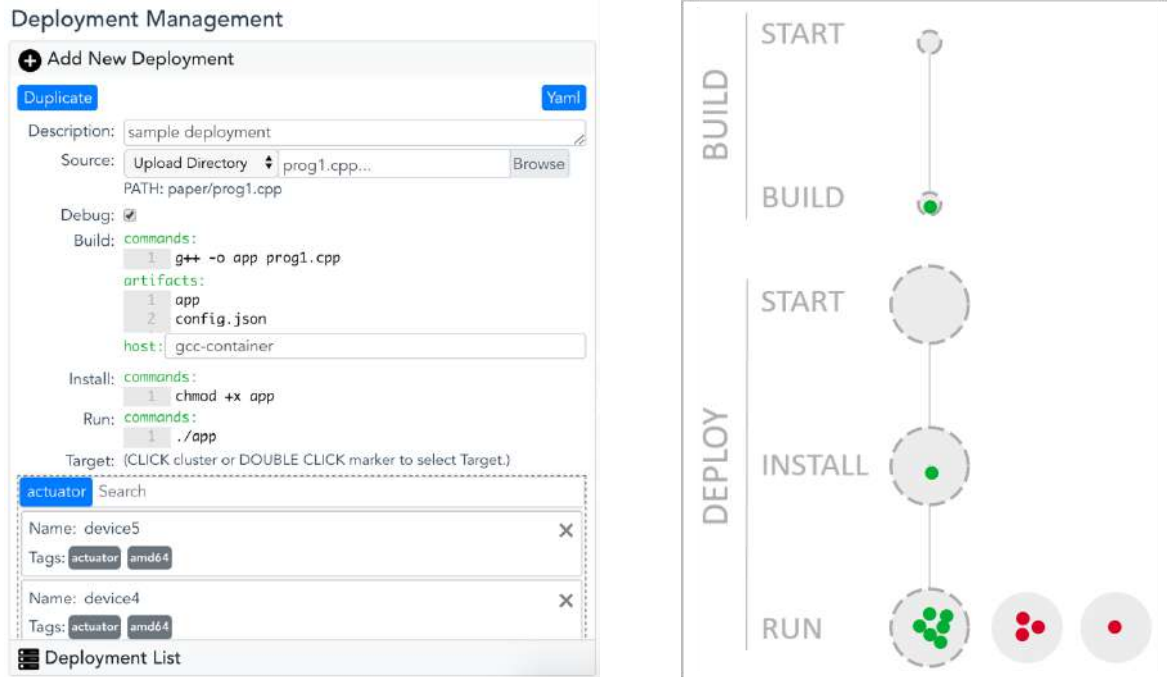


Figure 15. Deployment Tool's web interface for adding deployments (left) and monitoring the progress (right).

More information about the underlying components of the Deployment Tool and its user interface is available in D7.4.

2.5 Monitoring and Command

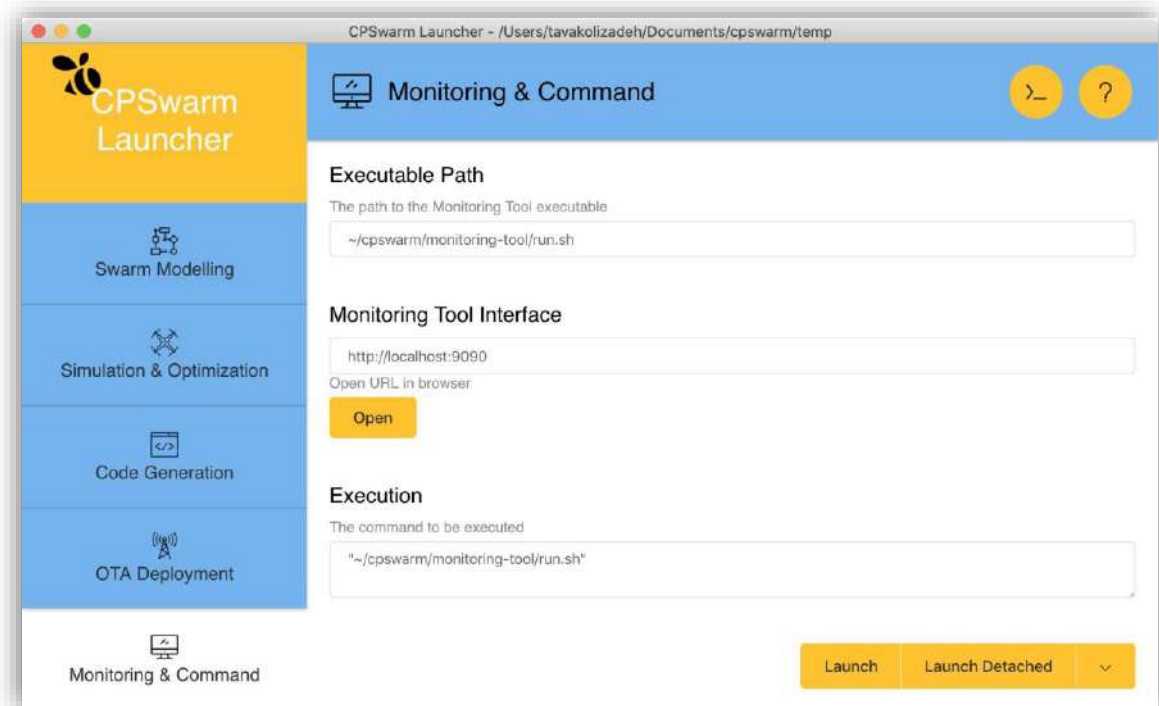


Figure 16. Monitoring & Command tab in the Launcher.

The monitoring and command part of the workflow serves as a mean to support the configuration of the swarm and on the other side to monitor its behavior during operation.

The Monitoring and Command Tool is used as part of the Runtime Environment and after the deployment phase. The user may use the Launcher to start the Monitoring and Command Tool's server and its web-based graphical user interface (see Figure 16). The user interface enables the user to start a mission, monitor the actual status of the swarm, as well as to send configuration/reconfiguration commands to modify/update the swarm behavior, e.g., to abort the mission or to re-purpose part of the swarm members. In general, the Monitoring and Command Tool gathers real-time data from the swarm members and on the other hand, sends out runtime command to the individual swarm members. The information gathered will be presented to the user through the GUI generated at launch time. The basic user interface is depicted in Figure 17.

The Monitoring and Command Tool uses the Communication Library to send and receive events and telemetry, to set and read back parameters and to discover swarm members on the network. Data exchanged between the swarm members and the Monitoring and Command Tool, natively exploits a Publish/Subscribe interaction pattern.

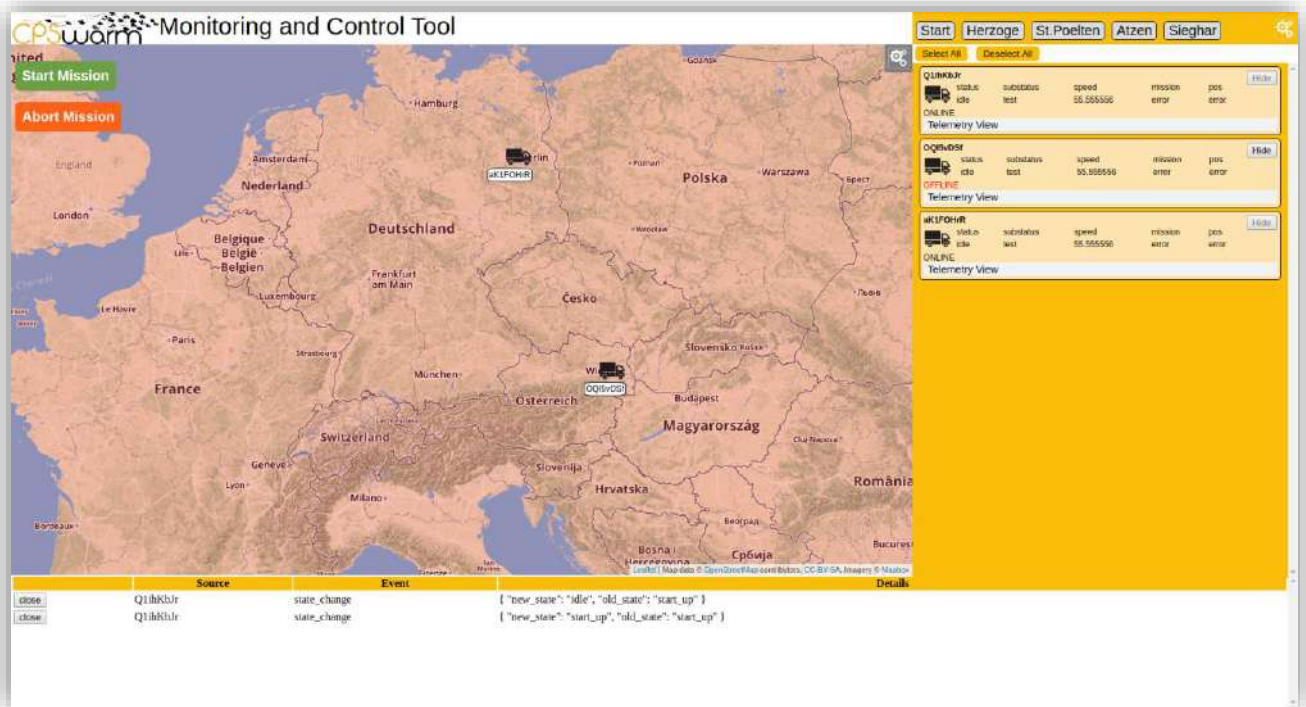


Figure 17: Basic Monitoring and Configuration Tool user interface

The use case specific configuration of the Monitoring and Command Tool is described in detail in the deliverables D7.5 and D7.6.

3 Continuous Integration and Delivery (CICD)

The test and integration plan for the CPSwarm system was documented in D3.7. The plan consists of the methodology for testing of individual components as well as the testing of the integration between different components. This plan was further extended to ease deployments by the addition of a continuous delivery mechanism. The continuous delivery ensures the availability of latest version of every component for use by different stakeholders.

This chapter gives an updated overview of the CICD platform along with high level usage guidelines. This is followed by a summary of implemented build and test plans as well as the continuous delivery setup.

3.1 CICD Platform

The deliverable D3.4 provided detailed description of the CPSwarm CICD platform consisting of build and version control infrastructures. Even though the platform supported various development activities in the project, it was not ideal for project's long-term dissemination goals. Most components of the CPSwarm system reached an exploitable maturity level and were licensed as open source. Thus, there was a need for public platforms to increase visibility and ease contributions from the open source communities. This was achieved by extending the CICD platform to include Github, the most widely used Cloud version control server.

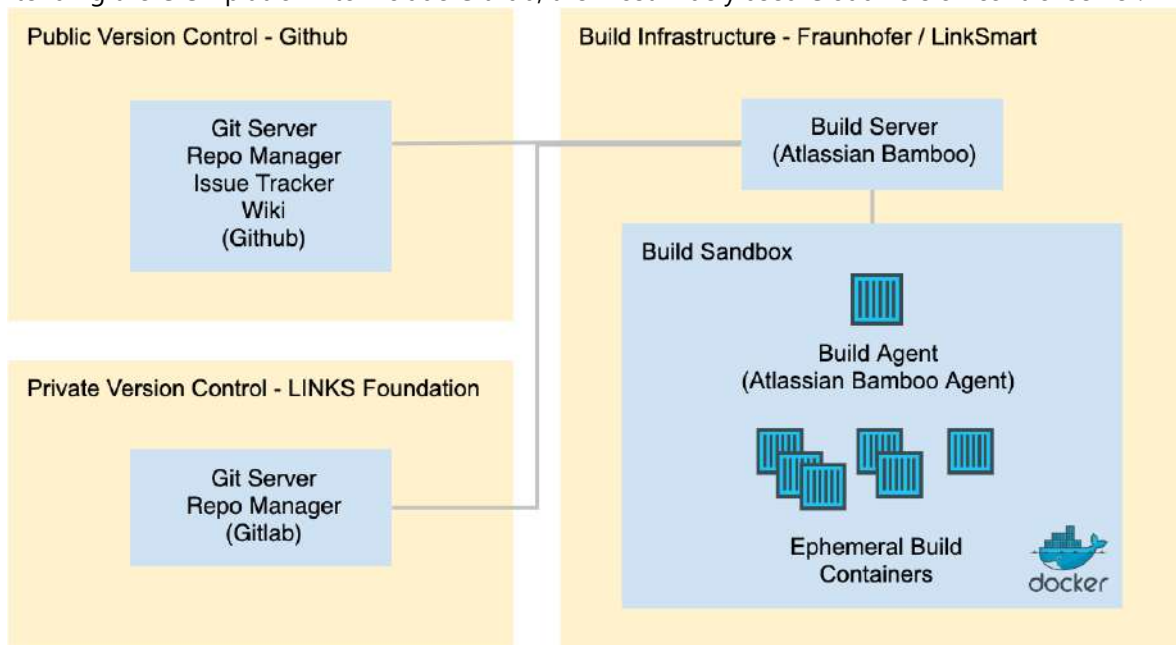


Figure 18. CPSwarm Continuous Integration and Delivery (CICD) Platform.

Figure 18 illustrates the deployment model of the CPSwarm CICD platform including the main software components. The components of the CPSwarm CICD platform are administrated by the consortium members or available as public free services. In particular, Github offers free services for public code management, issue tracking, and documentation, LINKS Foundation provides the infrastructure for code management, and Fraunhofer maintains the build server.

Public Version Control

Github offers free services for version control of publicly accessible projects. The consortium relies on Github for long term storage of the technical project outcomes. The CPSwarm project has an organization in Github with the same name⁷. All open source components are available on Github with corresponding issue trackers

⁷ <https://github.com/cpswarm>

and wiki spaces. All technical consortium members are part of the CPSwarm Github organization and have full access to their component source codes. External contributors may fork any of the repositories and extend or customize them to their needs. External contributions may be submitted to the master repository via Git pull requests. Merging external changes into a master library is subject to the approval of the component owner.

Private Version Control

The private Git server and management interface is provided by an instance of GitLab⁸, a leading open-source software for code management and continuous integration. The instance is deployed and maintained by LINKS Foundation. In the CPSwarm CI platform, we only utilize the code management features of GitLab and realize continuous integration using other tools. The GitLab instance is available to consortium members at ISMB PerT Area Git Repository⁹. Members can perform push/pull git operation on projects created by them and those which they have given write permissions. Pull operations are allowed on all other projects related to CPSwarm. Additionally, members can use the management UI to create and modify projects, view code and branching history, and manage access rights.

Build Infrastructure

The build system is composed of a build server, at least one build agents, and any number of ephemeral build containers. These functional suites consist of one or more components that are containerized with Docker¹⁰. The containerization enables component isolation and portability. Each component is further explained below:

- **Build Server:** The build server is an instance of Atlassian Bamboo¹¹, a professional tool for continuous integration, deployment, and delivery. Atlassian offers free licenses of Bamboo to projects that are open-source and public (Atlassian, 2017). We currently utilize a Bamboo instance deployed as part of the LinkSmart¹² ecosystem. The instance is accessible at LinkSmart Pipelines¹³. Bamboo is connected to the Git servers, listening to changes in the source codes. Currently we perform polling every five minutes. Depending on the configuration, the developers will be notified about the status of successful and/or failed builds by email.
- **Build Agent:** A build agent is an Atlassian Bamboo Agent, responsible for performing builds and different kinds of tests. Each agent can perform one job at a time. Currently, the system has one agent in deployment but it can be easily replicated to allow parallelized builds and tests. The build agents subscribe to a broker exposed by the build server to be informed about build jobs. Once a job is published, agents start picking and executing tasks and publish the resulting logs and artefacts to the build server. The execution of tasks is done in ephemeral build containers. For better isolation, the build agent resides in another virtual machine (VM), which is separated from the VM other services run on. This way, even if errors happen on the build agent, which corrupt the VM, other build services would not be affected.
- **Ephemeral Build Containers:** The Docker containers are created for a specific job and removed upon job completion. A job may create more than one container in order to perform integration tests that require multiple running services. In any case, all containers related to a job are destroyed after job success or failure. The ephemeral container approach helps to isolate build tasks from each other. Furthermore, it ensures that tests are not influenced by each other or the hosting operating system of the build system. The ephemeral build containers reside in the same VM as the agent.

⁸ <https://gitlab.com>

⁹ <https://git.repository-pert.ismb.it>

¹⁰ <https://www.docker.com>

¹¹ <https://www.atlassian.com/software/bamboo>

¹² LinkSmart® is a trademark used by Fraunhofer for IoT software utilities

¹³ <https://pipelines.linksmart.eu>

3.2 Platform Guidelines

As mentioned in the previous section, Bamboo has been used as the continuous integration tool. Bamboo is a powerful and flexible system that allows different kinds of implementation based on different use cases. In order to properly utilize Bamboo for continuous integration, it is essential to get familiar with the Bamboo environment and terminology.

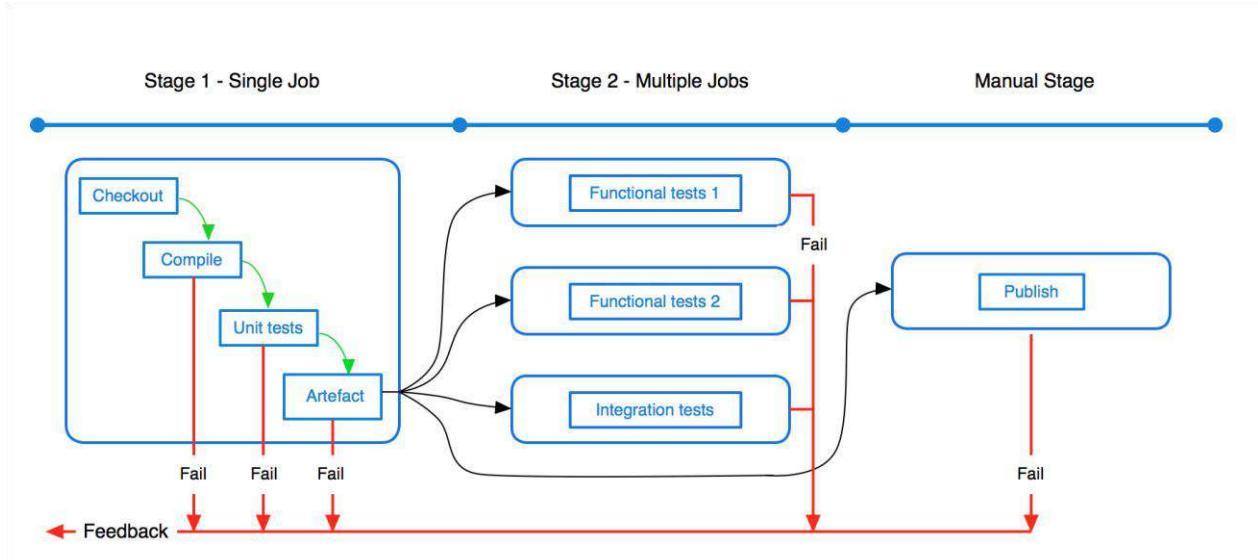


Figure 19. Multi-stage Bamboo plan (Atlassian, 2017)

At the highest level of abstraction, Bamboo divides the CI workflow into build and deployment projects. Build projects contain instructions for building, testing, and publishing software snapshots. Deployment projects include instructions for building and publishing releases as well deploying them on target systems.

A build project is a logical grouping for a set of build plans. We have created one project for CPSwarm. Build projects are structured as below:

- **Build plan:** Plans are independent instructions with separate triggering mechanisms. Each plan consists of one or more stages. Each implemented and testable CPSwarm component has at least one plan. Plans are triggered manually, after changes detected in the source code, or following a predefined schedule.
- **Stage:** Each stage within a plan represents a step within in the build process. A stage may contain one or more jobs which Bamboo can execute in parallel. For example, there can be a stage for compilation jobs, followed by one or more stages for various testing jobs, followed by a stage for deployment jobs.
- **Job:** A group of tasks with shared requirements resulting in one or more artefacts.
- **Task:** A piece of work that is executed as part of a job. Check out source code, the execution of a script, and a shell command are only few examples of tasks.

Figure 19 illustrates a build plan with three stages. Stage 1 consists of a single job with four tasks. Each task leads to the next and a failure at any tasks will break the job and send a feedback. Stage 2 contains three parallel jobs with single tasks all triggered after the successful completion of Stage 1. Failure of each job is reported. Stage 3 is a manual stage that can be triggered after the success of Stage 1. This stage has one job with a single task.

A deployment project allows defining tasks similar to build plans but designated for a specific target environment. In other words, deployment projects consist of one or more environments, each with a single

job. The job includes a set of tasks in order to build and deploy the project to the target environment. Deployment projects can be triggered manually or automatically after a successful build plan.

In CPSwarm, different components are separated into different plans because they are developed by independent parties and managed in separate code repertories. Furthermore, during build-time tests, Docker containers are utilized to contain components for better dependency isolation. A Docker¹⁴ container is a lightweight, stand-alone instance of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings. Available for both Linux and Windows based apps, containerized software always runs the same, regardless of the environment. Containers isolate software from its surroundings, for example, differences in development and staging environments. They help to reduce conflicts between teams running different software on the same infrastructure.

3.3 CICD Practices

A successful CICD workflow relies on a robust and transparent lifecycle. The CPSwarm components follow. This section describes the utilized practices to achieve smooth build, test, and release cycles coupled with flexible monitoring capabilities. The details of the applies practices to individual components are documented in Section 3.4.

3.3.1 Automated Builds and Tests

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 project¹⁵, 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. Based on the architecture, instead of building a monolithic CPSwarm workbench, the consortium has decided to build the CPSwarm system with highly decoupled software components, each of which 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 have 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.

3.3.2 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 continuous delivery integrates with CI and makes it easy to release the code that is added to version control and successfully built and tested. Figure 20 illustrates various cycles of integration and delivery.

¹⁴ <https://www.docker.com/>

¹⁵ <https://pipelines.linksmart.eu/browse/CPSW>

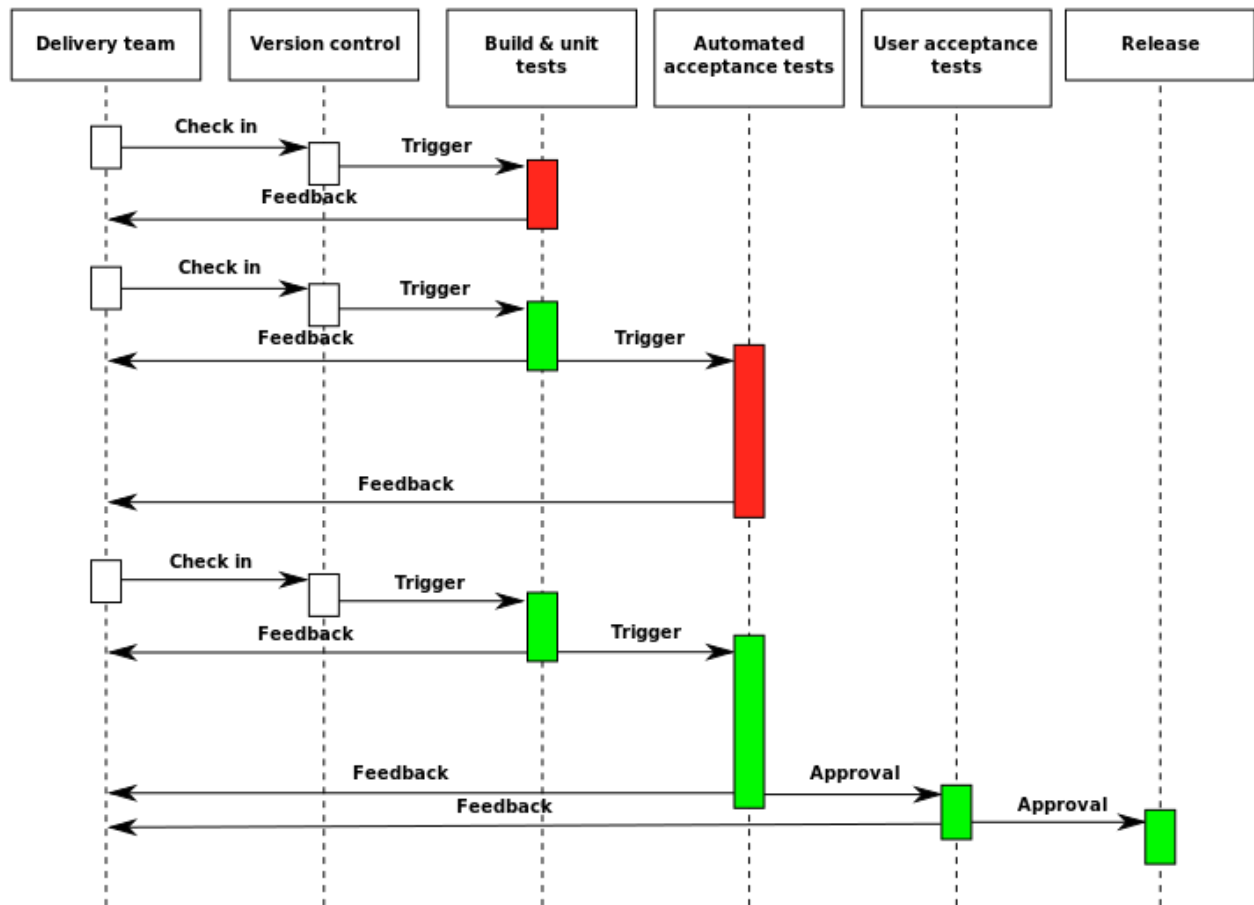


Figure 20. The process of continuous integration and delivery (CICD).¹⁶

In CPSwarm, the continuous delivery 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.

3.3.3 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 as shown in Figure 21. In addition, users can refer to the CI project for CPSwarm¹⁷ to monitor the progress and see the status. Figure 22 shows a screenshot of the CI project home page.

¹⁶ https://en.wikipedia.org/wiki/Continuous_delivery

¹⁷ <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 | Junhong Liang <i>(user)</i> |   |

Figure 21. Sample notification settings for a plan (i.e. Launcher)

















|  Build dashboard CPSwarm Project https://www.cpswarm.eu/ | | |
|--|--|--------------|
| Plan | Build | Completed |
| Code Generator |  #38 | 1 week ago |
| Communication Library (Swarmio) |  #45 | 2 weeks ago |
| Deployment Tool  |  #282 | 18 hours ago |
| Launcher  |  #51 | 1 week ago |
| Modeling Tool (Modelio Attack Tree Module) |  #64 | 1 hour ago |
| Modeling Tool (Modelio CPSwarm Extension) |  #72 | 1 week ago |
| Optimization Tool (Frevo) |  #69 | 1 year ago |
| Optimization Tool (Frevo) - Initialization |  #27 | 1 year ago |
| Optimization Tool XMPP (FREVO-XMPP) |  #5 | 1 week ago |
| Simulation and Optimization Orchestrator |  #146 | 1 week ago |
| Simulation Environment - Minisim Build and Test |  #7 | 1 year ago |
| Simulation Manager - Gazebo |  #9 | 1 week ago |
| Simulation Manager - Stage |  #5 | 1 week ago |

Figure 22. Screenshot of the build overview at the time of writing.

3.4 CICD Plans

The following sub-sections summarize the implemented build and test plans for individual CPSwarm components:

3.4.1 Launcher

A Bamboo plan has been set up for the CPSwarm Launcher which builds the launcher from source and produces executable artifacts. This identifies and notifies the committers about possible compilation issues at every development cycle.

At the time of writing, the launcher provides a framework for testing but implementing specific test procedures requires future work. Testing the launcher is possible by preparing sample Launcher projects as the reference inputs for end-to-end test. The tests may involve the following aspects:

- 1) Testing of UI behavior. For this purpose, sample Launcher projects are loaded into the Launcher. UI behaviors such as whether a tab is displayed in correct state according to the sample projects, and whether the list of files is shown correctly may be tested. Such testing is possible with the help of the integrated Spectron¹⁸ framework.
- 2) Testing of Launcher output. For this purpose, sample Launcher projects with pre-configuration may be loaded into the Launcher. The Launcher would try to launch testing scripts instead of real components from the pre-configuration. Such testing scripts verify whether the Launcher is launching them with correct parameters according to the pre-configuration.

Continuous Delivery

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

Building the project on CI server for macOS is currently not possible because the underlying Electron framework only supports building and signing macOS artifacts natively, on a macOS device. Figure 23 shows the build summary on the CI server.

¹⁸ <https://electronjs.org/spectron>

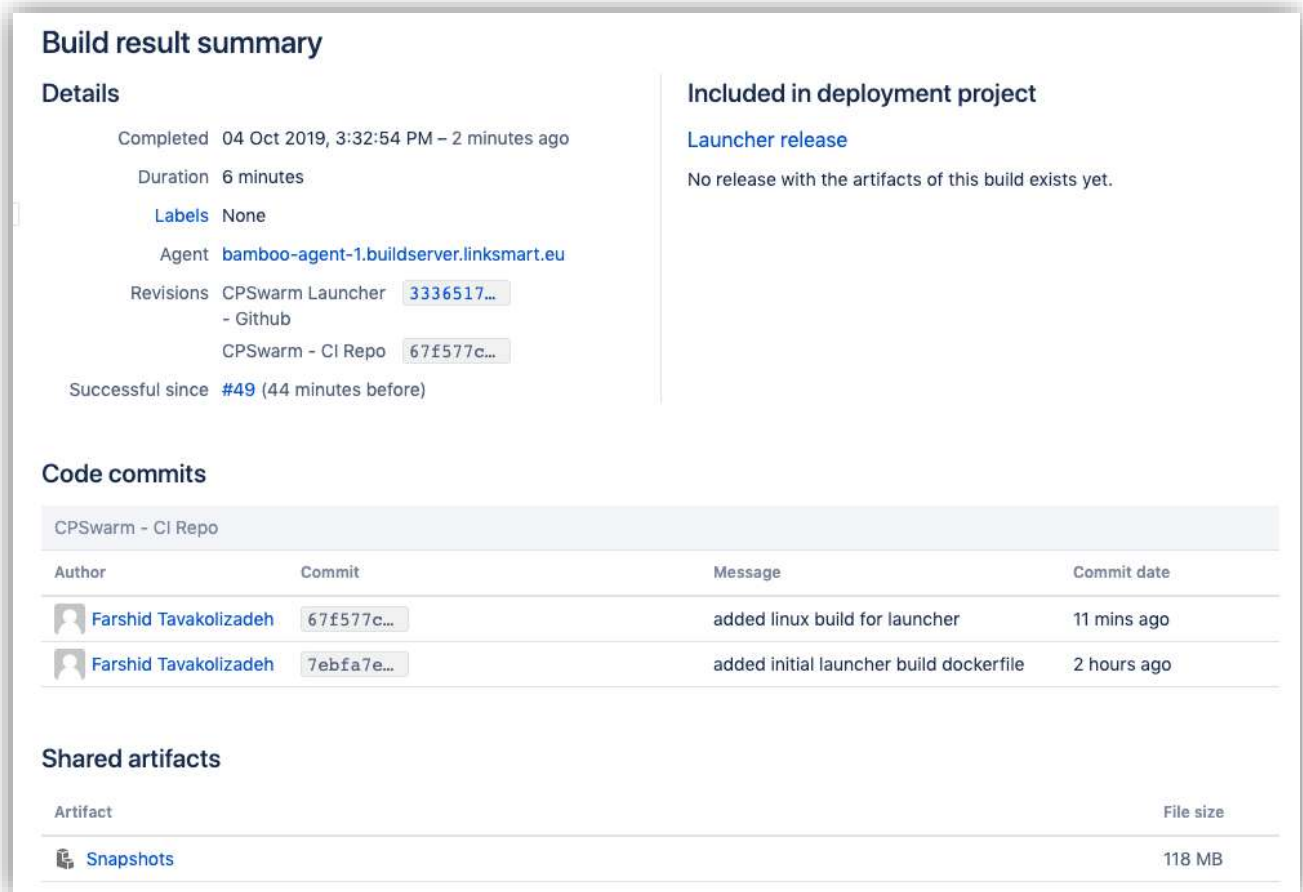


Figure 23. Build result summary for CPSwarm Launcher.

3.4.2 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 is covered by the testing. A Bamboo plan named has been established to test the CPSwarm extension. The plan is composed of the following tasks:

- Build Modelio CPSwarm extension from source
- Deploy the extension into a sample Modelio project
- Open the sample project into Modelio
- Generate output files from the sample project data using the CPSwarm extension dedicated commands
- Verify the generated output files against reference output files

Currently, the plan tests the interfaces between the Modelling Tool and:

- the Code Generator by generating and testing the conformance of SCXML files;
- the Communication library by generating and validating communication configuration files;
- the SOO by generating and validating Swarm configuration (XML file), the list of optimized parameters (JSON file) and the fitness function calculation (Python file).

Continuous Delivery

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 continuous delivery 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 24 shows the Modelio artefact delivery page¹⁹. 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)
- macOS (64-bit)

[Latest release](#)
[Extensions](#)
[Previous versions](#)
[Alternative download links](#)

Download Modelio [Latest version: 3.8.1]

The latest version of Modelio 3.8.1 (Build 201904162130) is now available *(Last update on April 17st, 2019)*.

Please select the right file for your system.

| Platform | Architecture | File |
|-----------------|--------------|---|
| Modelio | | |
| Windows 7/8/10 | 64-bit | Modelio 3.8.1 - Windows 64-bit (309.94 MB) |
| Windows 7/8/10 | 32-bit | Modelio 3.8.1 - Windows 32-bit (300.69 MB) |
| RedHat/CentOS 7 | 64-bit | Modelio 3.8.1 - Red Hat/centOS 7 64-bit (308.41 MB) |
| RedHat/CentOS 7 | 32-bit | Modelio 3.8.1 - Red Hat/centOS 7 32-bit (314.15 MB) |
| RedHat/CentOS 6 | 64-bit | Modelio 3.8.1 - Red Hat/centOS 6 64-bit (308.41 MB) |
| RedHat/CentOS 6 | 32-bit | Modelio 3.8.1 - Red Hat/centOS 6 32-bit (314.15 MB) |
| Debian/Ubuntu | 64-bit | Modelio 3.8.1 - Debian/Ubuntu 64-bit (296.8 MB) |
| Debian/Ubuntu | 32-bit | Modelio 3.8.1 - Debian/Ubuntu 32-bit (302 MB) |
| MacOS X | 64-bit | Modelio 3.8.1 - MacOS X (172.5 MB) |

Figure 24. 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. Figure 25 shows a build summary for the CPSwarm Extension.

¹⁹ <https://www.modelio.org/downloads/download-modelio.html>

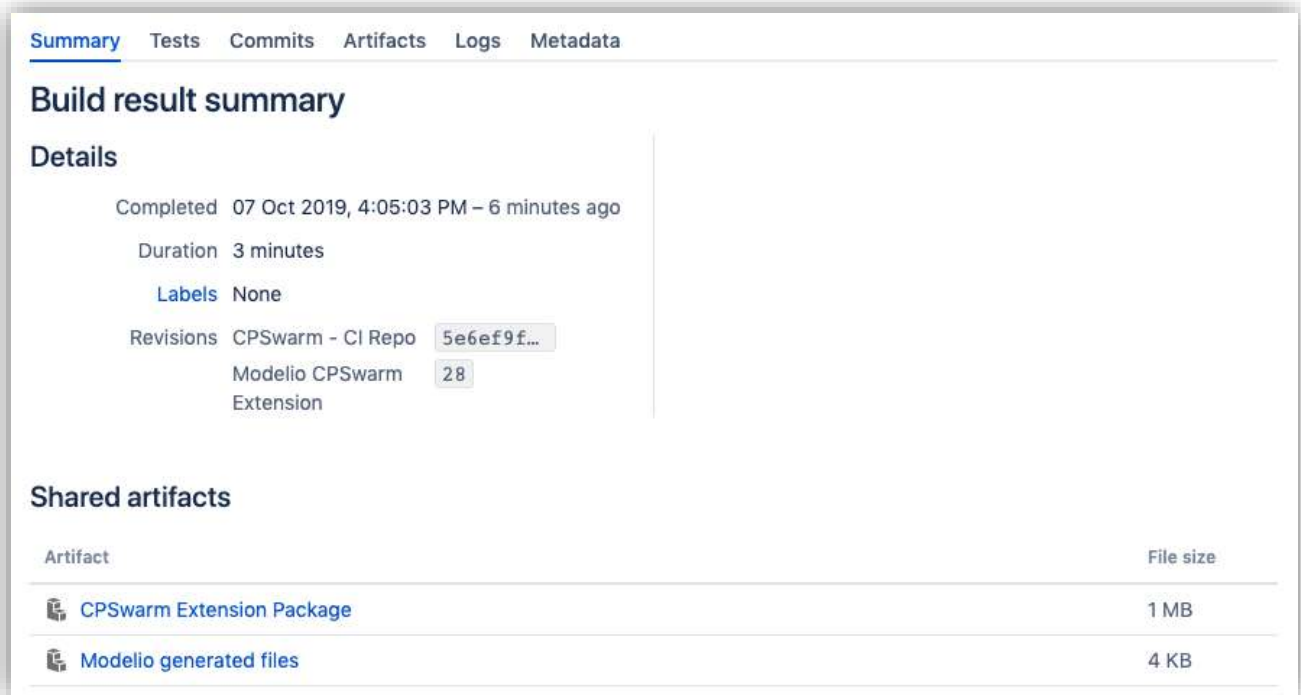


Figure 25. Build result summary for the Modelio CPSwarm extension.

3.4.3 Modelling Library

No specific tests are directly related to the Modelling Library but, several tests performed for the Modelio CPSwarm extension (Section 3.4.2) use models specified on top the Modelling Library. In such case, tests validate the CPSwarm extension but also the fact that the library is well deployed and well formed.

The modelling library is needed by the Modelio CPSwarm extension, so Modelio CPSwarm extension build (Section 3.4.2) includes modelling library release to be deployed and available together with the extension. Figure 26 shows the Modelling Library as part of the Modelling Tool. This model is exported and included during CPS extension build.

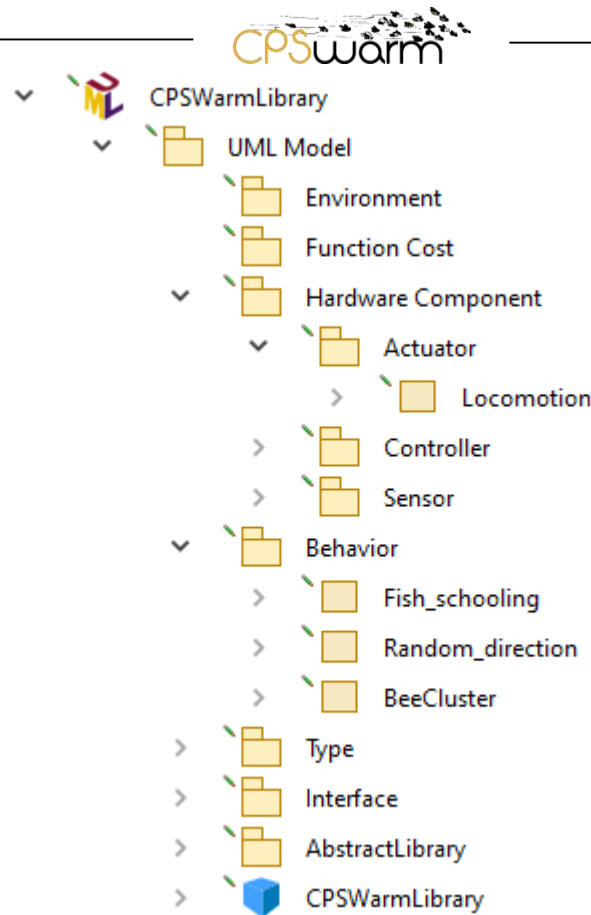


Figure 26. Modelling project of the Modelling library

3.4.4 Behavior Library

The Behavior Library consists of Abstraction Library, Swarm Library, and Communication Library.

3.4.4.1 Abstraction Library

For the Abstraction Library, no comprehensive test suites have been realized. The library is composed of a set of heterogeneous ROS packages and the responsibility to individually test each package was left to the related developers. When available, tests will be reported in the dedicated readme files associated with each GitHub repository.

In general, the main ROS guidelines²⁰ for unit testing are followed in order to design the relevant tests for each package. The testing phase is more focused on ensuring the proper functioning of the ROS independent part of the code while the integration among the different components is mainly conducted through simulations and observations driven by the application requirements.

There are currently no continuously built distributions of the Abstraction Library due to complex build and runtime dependencies. Instead, the library is built directly on targets or remotely using the Deployment Tool.

3.4.4.2 Swarm Library

No specific tests exist for the swarm library. The individual CPS functionality is already verified in the abstraction library (e.g., UAV current position and velocity). The swarm library on the other, uses combinations of these

²⁰ <http://wiki.ros.org/Quality/Tutorials/UnitTesting>

individual functionalities for a swarm behavior. These behaviors operate in a self-organized way. For self-organization, no formal tests are defined yet (Brambilla, Ferrante, Birattari, & Dorigo, 2013). The high dynamics results in a complexity that makes formal testing extremely complicated. Generally, the domain of self-organization lacks defined metrics and test-beds. Thus, researchers work with observations, metrics, and tests that are defined in strong coordination with the specific use case the self-organized system works on. For example, for a coverage algorithm (see D4.5 – Updated Swarm Modeling Library for further information) of the search and rescue use case, we observed coverage time over number of UAVs in a swarm for varying coverage percentage rates during an abstract simulation (see Figure 27).

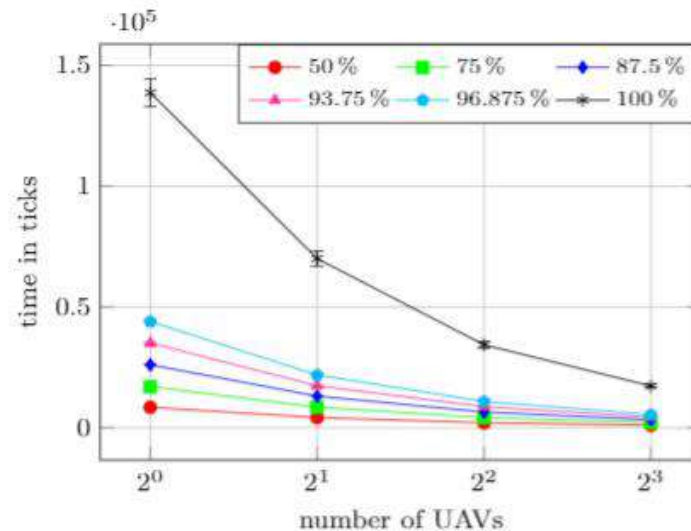


Figure 27. Simulation result of the coverage algorithm.

Distributions of the swarm library are released as software artifacts to the ROS community using the existing infrastructure. Once a new version of the library is available, it is released to the ROS repositories using the bloom release automation tool. It builds binaries of the library for following platforms:

- amd64
- arm64
- armhf
- i386

They are published in the public ROS Debian repositories.

At the same time, a documentation page is generated for the ROS wiki which documents the usage of the swarm library.

3.4.4.3 Communication Library

A Bamboo plan named Communication Library (Swarmio) has been created for the Communication Library. The plan has the task to build the Communication Library from source. This ensures that the source can be successfully compiled into binary distributions. There are currently two test plans for Communication Library.

Program start

The test ensures that the built simulator program of the Communication Library is able to run and exit without errors. In case there were no errors generated by the software while running, this test passes.

Discovery test

The test verifies that the discovery function of ZeroMQ is working as intended, making it possible for swarm members to discover each other in the same network. The test starts two instances (with unique IDs) of the simulator and check if both of the instances are able to discover each other. The test passes in case both of the instances are able to identify other instance with the predefined unique IDs.

Continuous Delivery of the Communication library

The communication library is needed on target devices as well as on the devices that run the monitoring tool. As such, we provide two sets of snapshots, one for desktop machines without ROS support and another for swarm devices with ROS:

- Linux – amd64 without ROS support (this may be used in a virtual machine or Docker container for running on a wide range of platforms)
- Linux – armv7
- Linux – arm64

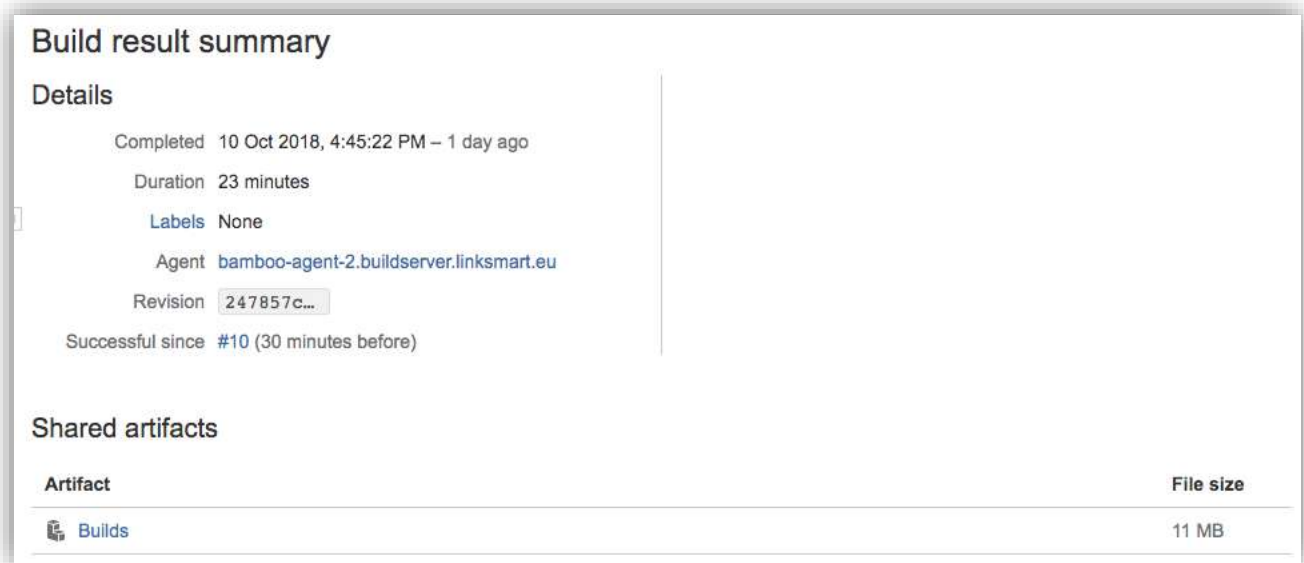


Figure 28. Build result summary for the communication library.

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.

3.4.5 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 four sub-tests for this end-to-end test. Figure 29 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 (XMPP²¹) user of the SOO on the XMPP server.

²¹ <https://xmpp.org/>

- 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.

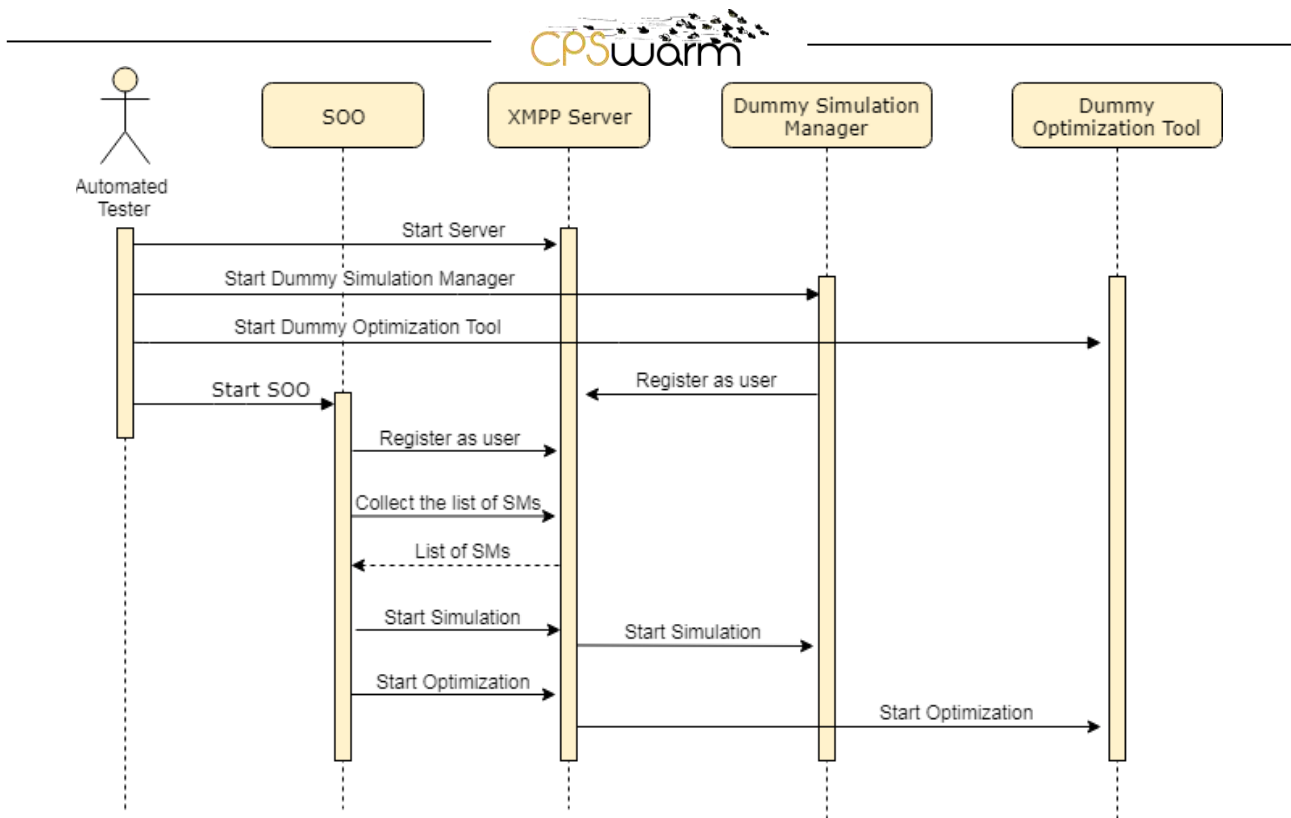


Figure 29. Integration test process, validating the interaction among Simulation Optimization Orchestrator, Simulation Manager and Optimization Tool

Test Kubernetes

This test is used to verify:

- The possibility to use the Kubernetes client embedded in the SOO to connect to a Kubernetes Master. Indeed, the SOO can also be used to automatically deploy the needed Simulation Managers in a Kubernetes cluster, to do this, it must be able to successfully connect to the Master node of an already existing Kubernetes cluster.

The test is passed if the SOO can connect to a Master node (it uses one public cluster instantiated for CPSwarm) and can use the client to query the API provided by the Master.

Besides the above tests, there is also a set of tests designed to test the features implemented to generate the simulation ROS package to actually run the simulation. The tests are as follows:

Test create ROS package

This test is used to verify:

- That the SOO is able to generate the simulation ROS package to be used to run the simulation in one of the supported simulators

The test checks that the package is actually created at the end of the process.

Test create ROS package with existing directory

This test is used to verify:

- That the SOO is able to generate the simulation ROS package updating one package already generated

The test checks that the package is actually updated at the end of the process.

Test generate

This test is used to verify:

- That the SOO is able to generate the code to be inserted in the simulation package

The test checks that the code is actually generated at the end of the process.

Test validate generated files

This test is used to verify:

- That the SOO is able to generate the code to be inserted in the simulation package.
- That the code generated is valid

The test checks that the code generated is valid code to be used in the simulation package.

Continuous Delivery

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 30 is a screenshot of the latest build summary for the artifacts.

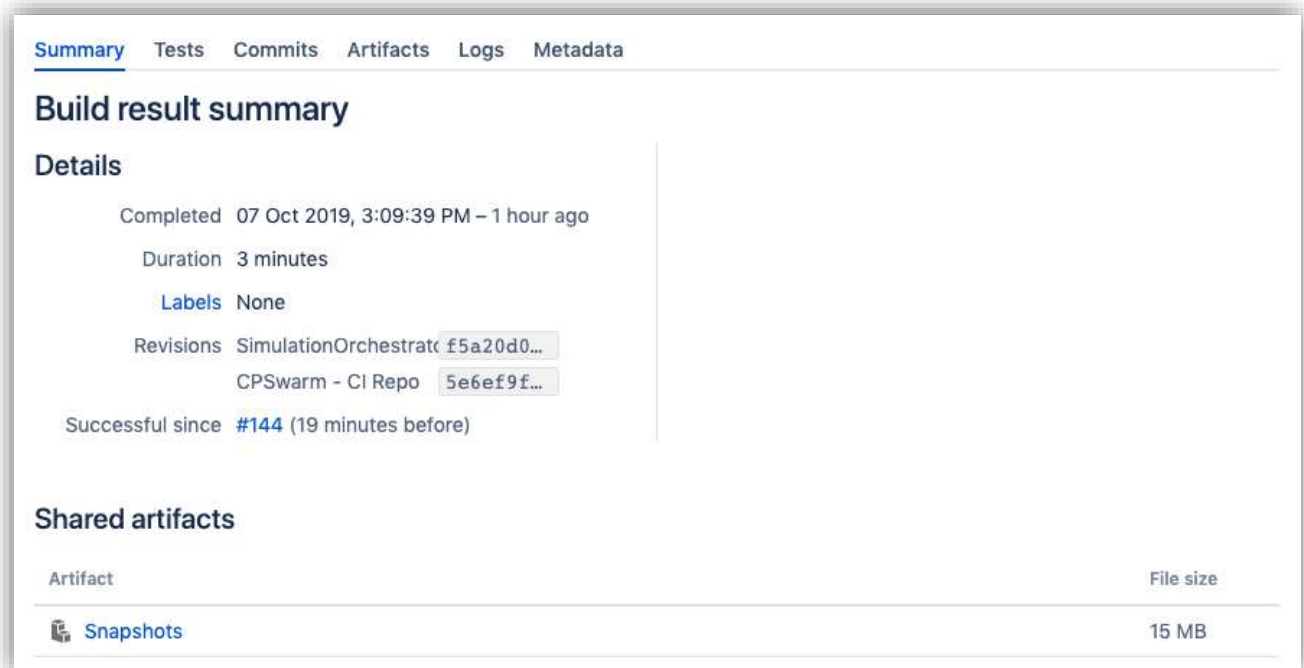


Figure 30. Build result summary for Simulation and Optimization Orchestrator.

3.4.1 Optimization Tool

The Optimization Tool XMPP (FREVO-XMPP) plan builds and tests the components of FREVO used in the CPSwarm project as well as its XMPP wrapper. The plan has the following tasks:

- Build FREVO and the FREVO-XMPP wrapper from source
- Build dummy SOO and SM components
- Test the interaction of FREVO-XMPP with dummy SOO and SM components by running an optimization task and verifying the flow of control between the components.

In more detail, the first two tasks clone the appropriate git repositories and build and install the components using Maven within a Docker image. A test XMPP server is started, followed by containers for FREVO, a dummy SOO and one or more dummy SMs. The test then proceeds to start an optimization task, which in turn requires FREVO to instruct the dummy SMs to perform simulations. Once all simulations have been performed, FREVO returns the optimum controller to the dummy SOO. An outline of the interaction between components is provided in the sequence diagram shown in Figure 31.

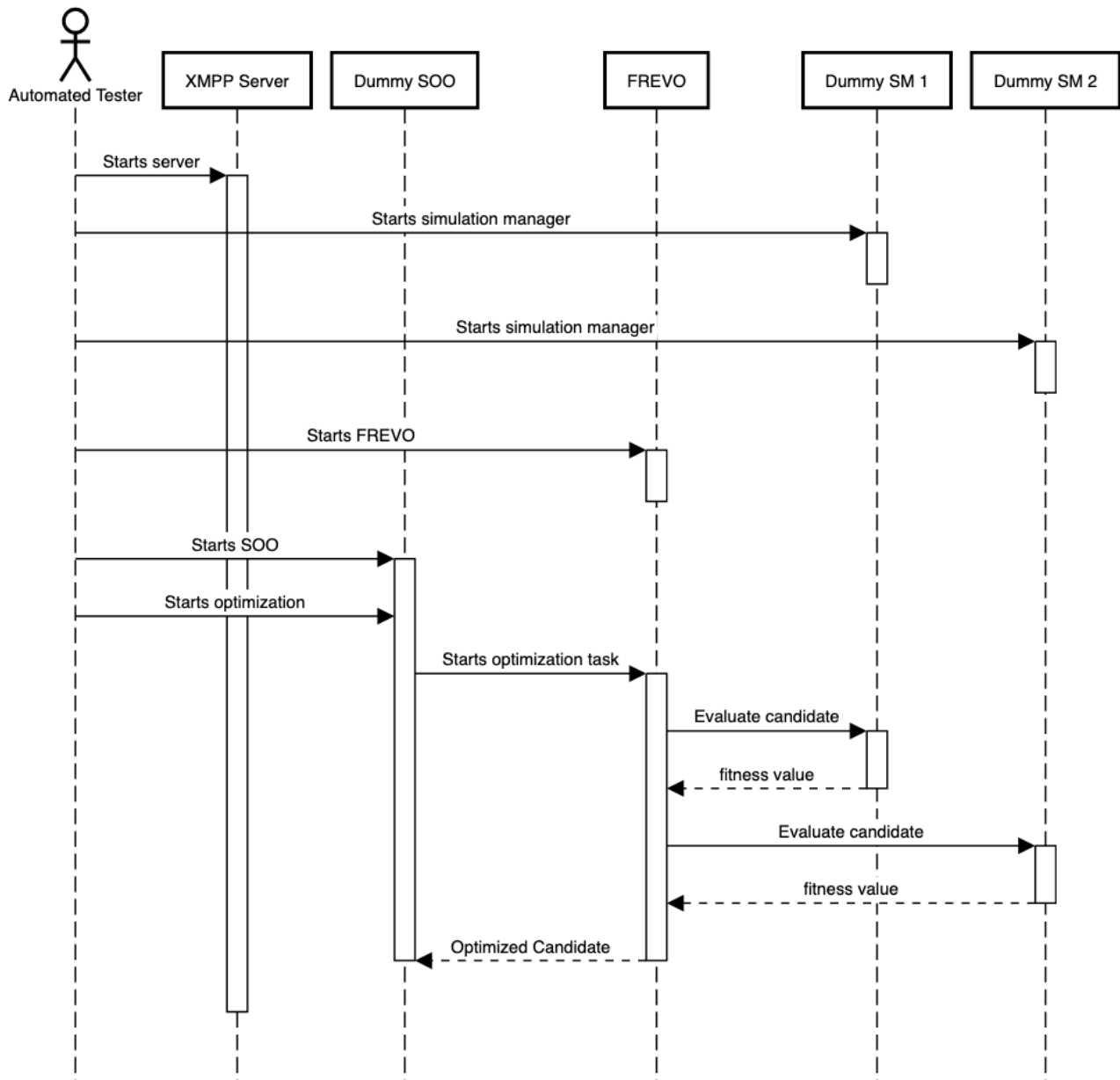


Figure 31. End-to-end test process for FREVO together with dummy SOO and SM components

Continuous Delivery

The build plan also produces artifacts for FREVO, namely a precompiled version (a JAR file) which may be used by other components or deployed as part of the CPSwarm Workbench.

The official branch of FREVO (Sobe, Fehévari, & Elmenreich, 2012) 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 32 shows the FREVO artefact delivery page²². This component is developed in Java and shipped as jar artefacts for all platforms that meet the dependencies.

²² <https://sourceforge.net/projects/frevo/files/>

SOURCEFORGE

Frevo
Frevo is probably the simplest tool for evolutionary design
Brought to you by: [arthurpitman](#), [frodewin](#), [ifeherva](#), [melanieschranz](#), and 2 others

Summary Files Reviews Support Wiki Tickets Discussion Blog

Download Latest Version
Frevo_v1.4.1.zip (18.7 MB) **Get Updates**

Home

| Name | Modified | Size | Downloads / Week |
|---------------------------|------------|---------------|------------------|
| FREVO main package | 2019-07-23 | | 1 |
| Libraries | 2012-08-03 | | 0 |
| FREVO components | 2012-08-03 | | 0 |
| readme.mkd | 2019-07-23 | 2.2 kB | 0 |
| Totals: 4 Items | | 2.2 kB | 1 |

Current latest release: 1.4.1
Release: 1.4.1

- Improved CEA2D optimizer
- Scaling works on 4K screen
- Runs with Java 8

Figure 32. FREVO release page.

3.4.2 Simulation Manager

As explained in the D3.3 - Final System Architecture & Design Specification the distributed architecture of the CPSwarm Simulation and Optimization Environment foresees the presence of several Simulation Managers. The ones released as open source are the one for Stage²³ and the one for Gazebo²⁴. A bamboo plan has been created for each Simulation Manager, which these tasks:

- Build the relative Simulation Manager from source.
- Extract the artifact of the relative Simulation Manager.
- Package the artifact in a docker image.
- Publish the image on a registry (i.e., DockerHub²⁵), to be automatically deployed using the Kubernetes client integrated in the SOO.

The Simulation Managers are part of the CPSwarm Simulation and Optimization Environment, and, for this reason, they are tested using the tests presented in Section 3.4.5. Indeed, those tests use the *Dummy Simulation*

²³ <https://github.com/rtv/Stage>

²⁴ <http://gazebo-sim.org/>

²⁵ <https://hub.docker.com>

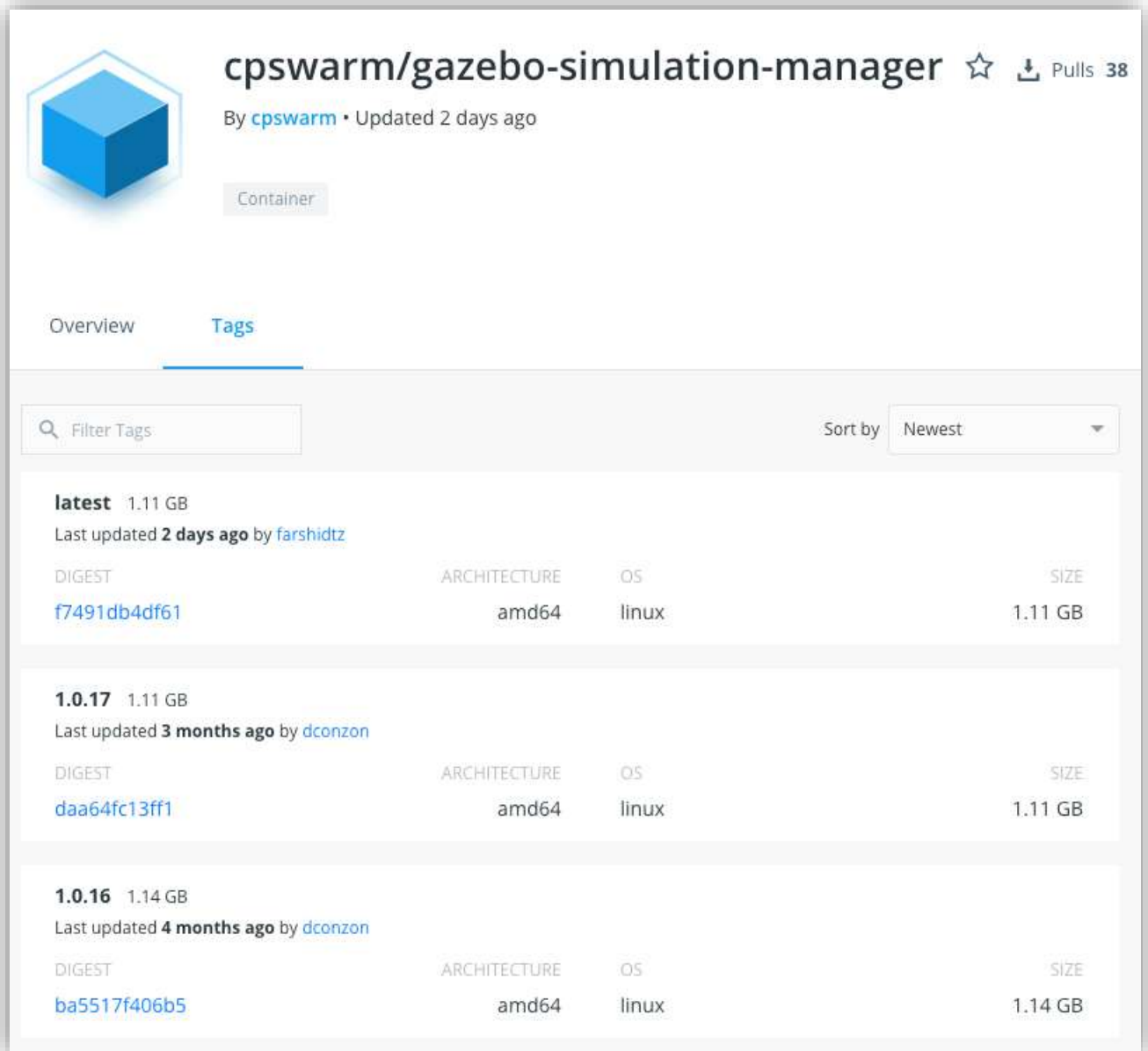
Manager, which is a simplified version of the SMs, which contains all the functionalities that need to be tested to guarantee that the SMs can run a simulation receiving the instructions from the SOO or the OT.

Continuous Delivery

The Simulation Manager builds are released on Dockerhub as Docker Images. The images are built for linux-amd64 architecture for two simulators: Gazebo²⁶ (see Figure 33) and Stage²⁷ (see Figure 34).

²⁶ <https://hub.docker.com/r/cpswarm/gazebo-simulation-manager>

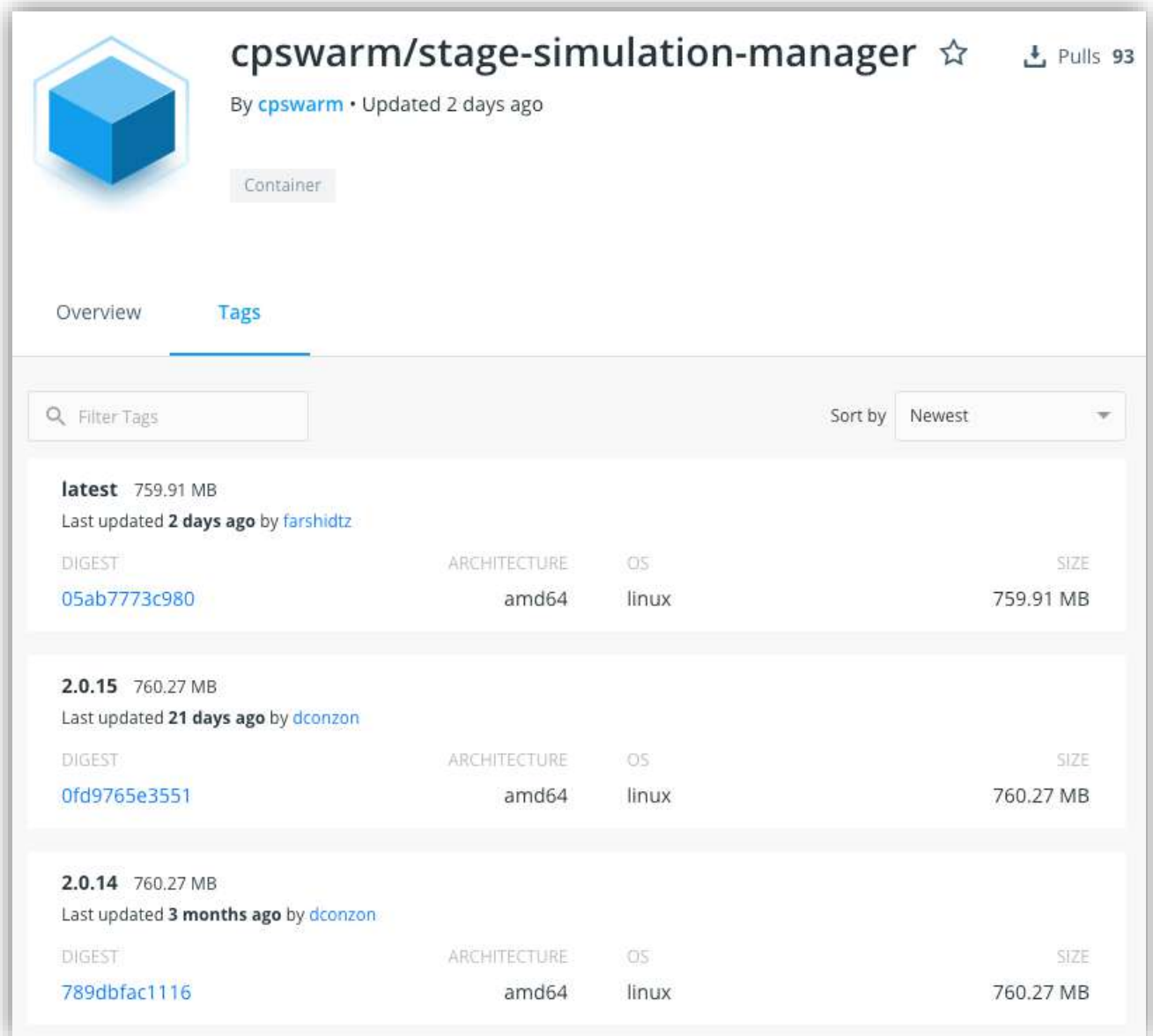
²⁷ <https://hub.docker.com/r/cpswarm/stage-simulation-manager>



The screenshot shows the Docker Hub page for the repository **cpswarm/gazebo-simulation-manager**. The repository is marked as a "Container" and has 38 pulls. It was updated 2 days ago by **cpswarm**. The "Tags" tab is selected, showing a list of published images. The images are sorted by "Newest".

| Tag | Size | Last updated | Updated by | Digest | Architecture | OS |
|---------------|---------|--------------|------------|--------------|--------------|-------|
| latest | 1.11 GB | 2 days ago | farshidtz | f7491db4df61 | amd64 | linux |
| 1.0.17 | 1.11 GB | 3 months ago | dconzon | daa64fc13ff1 | amd64 | linux |
| 1.0.16 | 1.14 GB | 4 months ago | dconzon | ba5517f406b5 | amd64 | linux |

Figure 33. Published Docker images of Gazebo Simulation manager on Dockerhub.



The screenshot shows the Dockerhub page for the repository **cpswarm/stage-simulation-manager**. It features a blue cube icon, a star icon, and a pull count of 93. The page is categorized as a 'Container' and was updated 2 days ago by 'cpswarm'. The 'Tags' tab is selected, showing a list of image versions sorted by 'Newest'. The list includes the 'latest' tag (759.91 MB) and two specific versions: '2.0.15' (760.27 MB) and '2.0.14' (760.27 MB). Each entry shows the last update time, the user who updated it, and a table of metadata including Digest, Architecture (amd64), OS (linux), and Size.

| Tag | Size | Last updated | By | Digest | Architecture | OS | Size |
|--------|-----------|--------------|-----------|--------------|--------------|-------|-----------|
| latest | 759.91 MB | 2 days ago | farshidtz | 05ab7773c980 | amd64 | linux | 759.91 MB |
| 2.0.15 | 760.27 MB | 21 days ago | dconzon | 0fd9765e3551 | amd64 | linux | 760.27 MB |
| 2.0.14 | 760.27 MB | 3 months ago | dconzon | 789dbfac1116 | amd64 | linux | 760.27 MB |

Figure 34. Published Docker images of Stage Simulation manager on Dockerhub.

3.4.3 Code Generator

A Bamboo plan named *Code Generator* has been created for the Code Generator. The plan, as already stated in D3.5, 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.

In addition to the previous points, the plan was extended to test the automatic generation of a complete ROS package (containing the algorithm code) to be deployed on a CPS:

- Run the Code Generator against a given SCXML file as sample input to produce a complete ROS catkin package.
- Verify the correct structure of the generated ROS catkin package against a target reference.

In particular, in order correctly generate a ROS package, a set of requirements must be met:

- The package must have its own folder.

- The package must contain a catkin compliant *package.xml*.
- The package must contain a *CMakeLists.txt* which uses catkin.

The simplest possible package might have a structure which looks like this:

```
my_package/
  CMakeLists.txt
  package.xml
```

Figure 35. Simple ROS catkin package structure

More details can be found in the dedicated ROS wiki page²⁸.

Finally, the test for the generation of algorithm templates (to be manually completed) have been integrated into the plan:

- Run the Code Generator against a given file containing the algorithm formal description (json) as sample input to produce an output file.
- Verify the output file against the reference output.

Continuous Delivery

The latest builds of the Code Generator are made available as JAR artifacts on the CI server. These artifacts may be executed on any platform with a Java Virtual Machine. Figure 36 shows a screenshot of the latest build result summary.

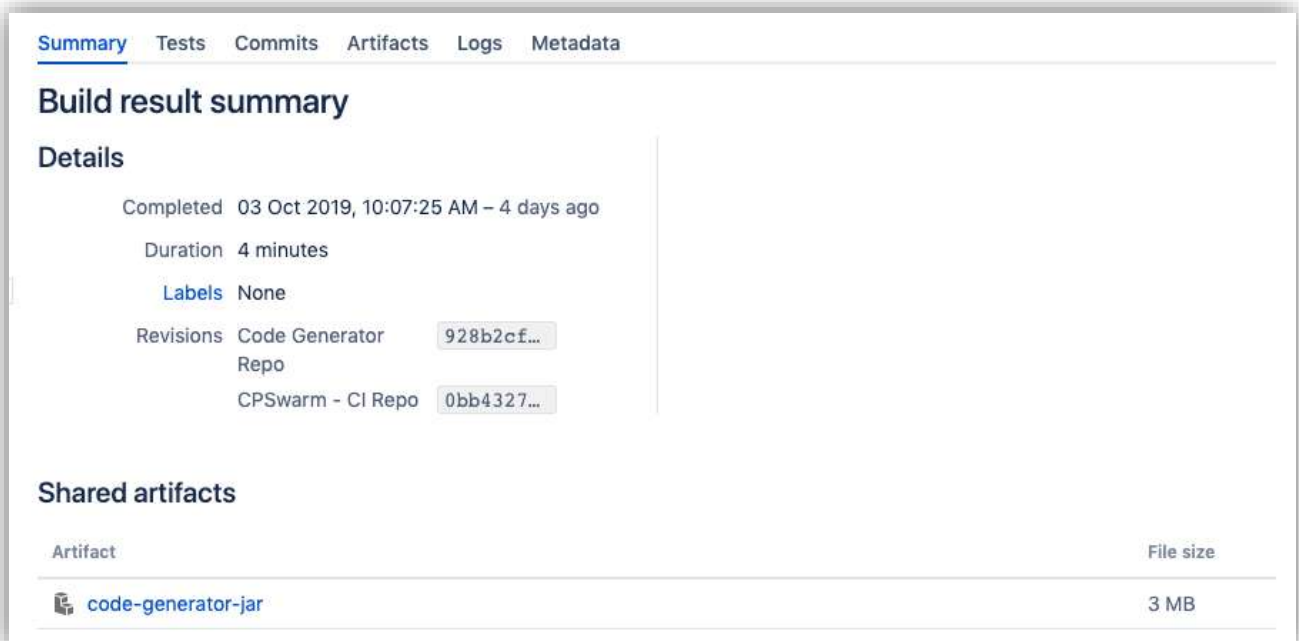


Figure 36. Code Generator build summary on the CI server.

3.4.4 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

²⁸ <http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage>

- 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 37), 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 entire test workflow is containerized with Docker and executed using the Docker SDK.

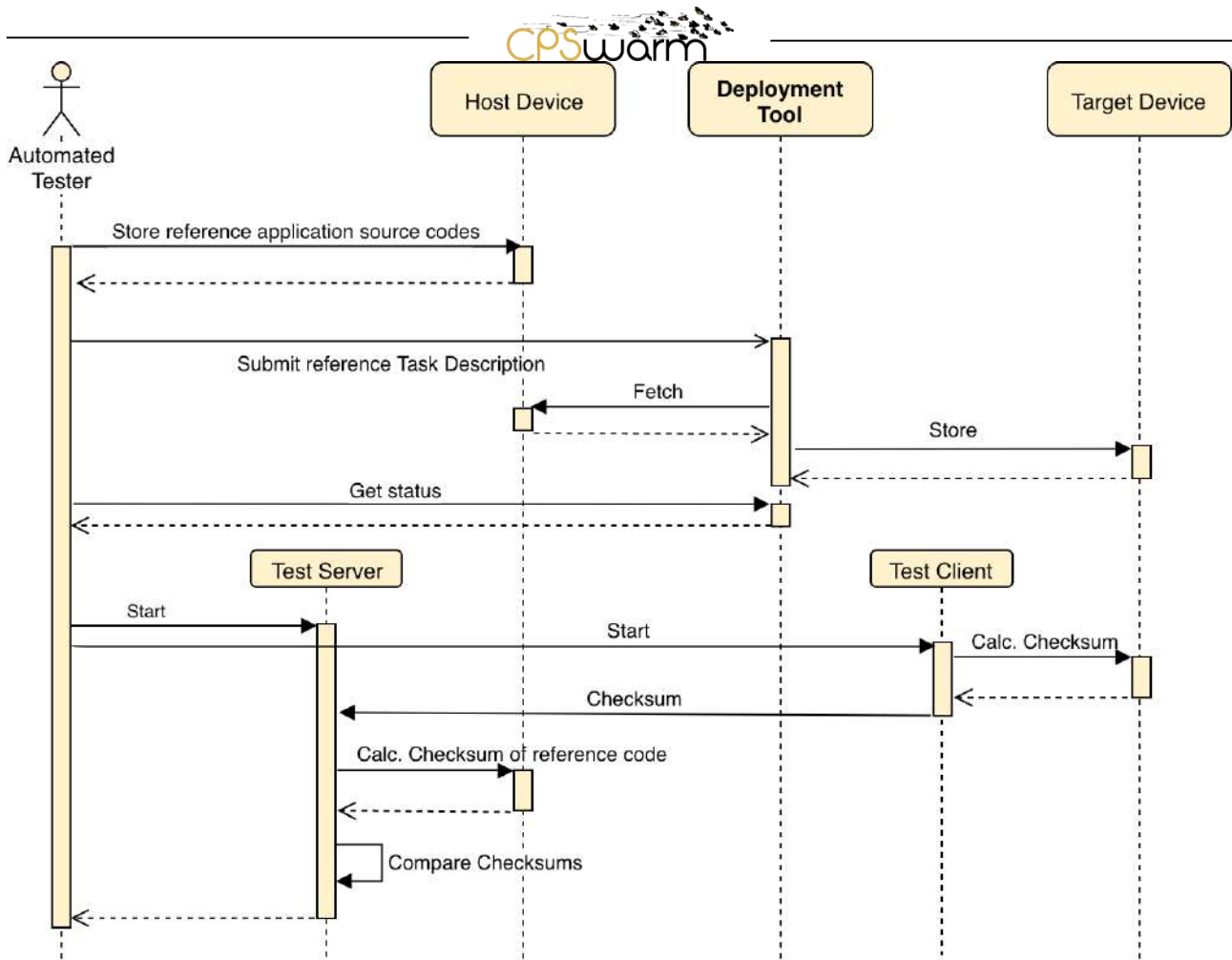


Figure 37. End-to-end test process, validating the behavior and compliance of Deployment Tool to the specification.

Continuous Delivery

The Deployment Tool's delivery involves producing snapshots for Deployment Manager, Deployment GUI, as well as Deployment Agent.

The Deployment Manager and Deployment GUI are desktop or server components and currently as Docker Images for linux/amd64. These images are publicly available on Dockerhub (see Figure 39 and Figure 40).

- 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, there are binary distributions as well as Debian packages for:

- Linux – armhf
- Linux – amd64

Build result summary

Details

Completed 26 Sep 2018, 3:36:51 PM – 2 weeks ago

Duration 1 minute

Labels None

Revision [9fbf7d5...](#)

Successful since [#74](#) (8 minutes before)

Included in deployment project

[Deployment Tool Release](#) › Release: [release-1](#)

| Environment | Status | Actions |
|-------------|----------------|---------|
| Release | SUCCESS | |

Shared artifacts



| Artifact | File size |
|--|-----------|
|  linux-amd64 binary distributions | 26 MB |
|  Frontend UI | 640 KB |
|  linux-arm binary distributions | 20 MB |
|  linux-arm debian package | 3 MB |

Figure 38. Build result summary for CPSwarm Deployment Tool.



linksmart/deployment-manager

By [linksmart](#) • Updated 24 days ago

Container

Pulls **481**

Overview **Tags**

Sort by Latest

latest 49.25 MB

Last updated **24 days ago** by [farshidtz](#)

| DIGEST | ARCHITECTURE | OS | SIZE |
|------------------------------|--------------|-------|----------|
| 5b672d196543 | amd64 | linux | 49.25 MB |

Figure 39. Latest published Docker image of Deployment Manager on Dockerhub.

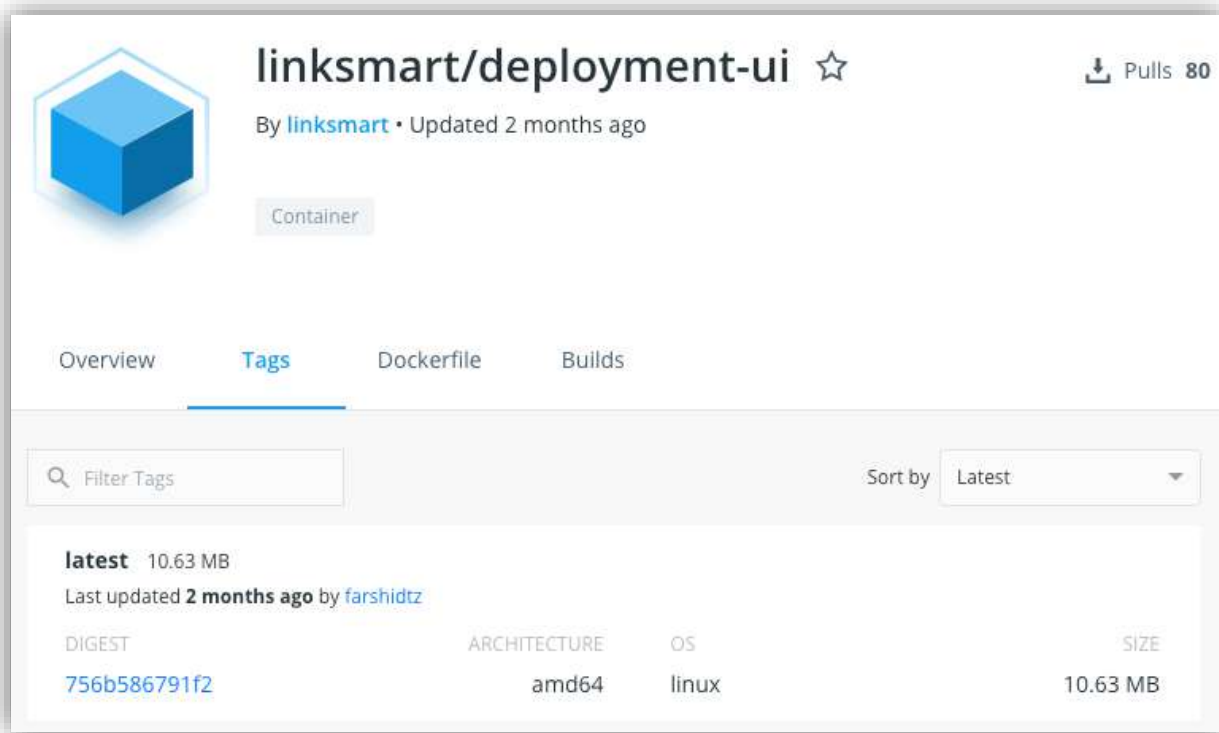


Figure 40. Latest published Docker image of Deployment GUI on Dockerhub.

3.4.5 Monitoring and Command Tool

The plan for testing the Monitoring and Command Tool is work in progress. The following tasks were planned and conducted for the Search and Rescue and the Automotive Scenarios:

- Build the Monitoring and Command Tool from source
- Run a dummy swarm member to communicate with the Monitoring and Command Tool to verify the correctness of communication
- Integrate it step by step into the scenario and check/test for correct function

Consequently, the Monitoring and Command Tool is built and tested manually and according to the test plan introduced above against the requirements originated from the use-cases. We first simulated the swarm by connecting up dummy devices and conduct the verification testing. This assured that no severe errors would endanger the later integrated swarm representatives. A further step conducted testing by using the swarm members as real world objects connected to the Monitoring and Command tool. This worked best with the automotive use case since the swarm members in the lab use case set-up are non-moving objects. The tool was then deployed for the Search and Rescue scenario where also flying drones and moving rovers were integrated step by step (meaning that only one swarm was connected at a time at the beginning and then the other). In a last step, the entire swarm community Rovers and Drones were connected and tested.

Each development cycle is firstly subject to an initial offline test, checking the basic functionalities. The detected insufficiencies are eliminated in this early stage. In the second step, it is integrated into the uses case scenarios to prove that it is correctly functioning by running/monitoring sample use case scenarios. Any findings during these integration and test steps have are subject to correction and upgrade of the Monitoring and Command Tool before the release. After the release, the project partners responsible for the use cases perform another round of tests to validate the integration.

Continuous Delivery

The snapshots of Monitoring and Command Tool are packaged together with the Communication Library as Docker Images. The images are built automatically on Bamboo whenever there is a change in the source code and available online to the consortium. Figure 41 is a screenshot of a latest build summary.

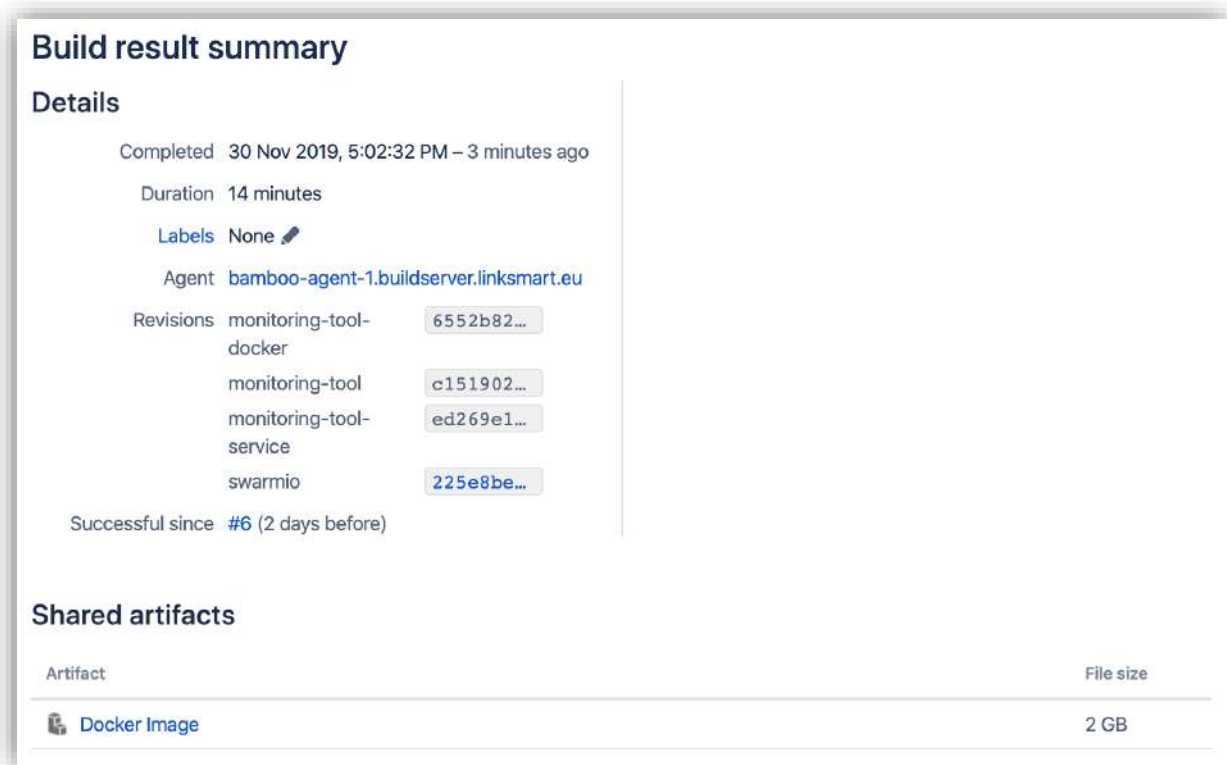


Figure 41. Build result summary for CPSwarm Monitoring & Command Tool.

4 Conclusions

This deliverable reported the implementation as well as the continuous integration and delivery of the CPSwarm Workbench and associated tools by M35 of the project. It briefly summarized the workflow of the CPSwarm system from a user's point of view. The proposed workflow intends to improve the user experience when dealing with heterogenous CPSwarm sub-systems. Furthermore, it presented the current setup of the automated builds, testing, and release of the components in accordance with common continuous integration and delivery (CI/CD) practices. The CI/CD resulted in a more integrated system by providing continuous feedback to developers on every new release over the course of the project.

This document is the last deliverable which reports the status of integration. However, the consortium expects fine-grained improvements in various components in the following weeks. Such changes would not dramatically affect the overall workflow presented here and will be presented in the documentation of individual components.

Acronyms

| Acronym | Explanation |
|---------|---|
| API | Application Programming Interface |
| BPMN | Business Process Model and Notation |
| CD | Continuous Delivery |
| CI | Continuous Integration |
| CICD | Continuous Integration & Continuous Delivery |
| CPDT | CPS Population Design Tool |
| CPS | Cyber-Physical System |
| FREVO | Framework for evolutionary design |
| FSM | Finite State Machine |
| GUI | Graphical User Interface |
| JAR | Java ARchive |
| MARTE | Modeling and Analysis of Real-time and Embedded systems |
| OT | Optimization Tool |
| ROS | Robot Operating System |
| SCXML | State Chart XML |
| SM | Simulation Manager |
| SOO | Simulation & Optimization Orchestrator |
| UAV | Unmanned Aerial Vehicle |
| VM | Virtual Machine |
| XMPP | eXtensible Messaging and Presence Protocol |

List of Figures

| | |
|---|----|
| Figure 1. Final architecture design of the CPSwarm system (Extracted from D3.3)..... | 6 |
| Figure 2. Screenshot of the Swarm Modelling tab in the Launcher. | 7 |
| Figure 3. Screenshot of the Modelling Tool while designing the architecture of a drone. | 8 |
| Figure 4. CPSwarm Behavior example..... | 8 |
| Figure 5. Swarm Composition Modelling | 9 |
| Figure 6. Fitness Function Modelling | 9 |
| Figure 7. Screenshot of the Simulation & Optimization tab in Launcher. | 10 |
| Figure 8. Optimization Monitoring..... | 11 |
| Figure 9. Cluster resources allocation | 12 |
| Figure 10. Simulation Managers deployed status..... | 12 |
| Figure 11. Logistic scenario simulation in Gazebo. (source: D6.6) | 12 |
| Figure 12. Code Generation tab in the Launcher. | 13 |
| Figure 13. Swarm Deployment tab in the Launcher..... | 14 |
| Figure 14. Deployment Tool's web interface for viewing the devices and their status..... | 15 |
| Figure 15. Deployment Tool's web interface for adding deployments (left) and monitoring the progress (right). | 16 |
| Figure 16. Monitoring & Command tab in the Launcher. | 16 |
| Figure 17: Basic Monitoring and Configuration Tool user interface | 17 |
| Figure 18. CPSwarm Continuous Integration and Delivery (CICD) Platform..... | 18 |
| Figure 19. Multi-stage Bamboo plan (Atlassian, 2017) | 20 |
| Figure 20. The process of continuous integration and delivery (CICD). | 22 |
| Figure 21. Sample notification settings for a plan (i.e. Launcher) | 23 |
| Figure 22. Screenshot of the build overview at the time of writing. | 23 |
| Figure 23. Build result summary for CPSwarm Launcher. | 25 |
| Figure 24. Modelio artefact delivery page. | 26 |
| Figure 25. Build result summary for the Modelio CPSwarm extension..... | 27 |
| Figure 26. Modelling project of the Modelling library..... | 28 |
| Figure 27. Simulation result of the coverage algorithm..... | 29 |
| Figure 28. Build result summary for the communication library. | 30 |
| Figure 29. Integration test process, validating the interaction among Simulation Optimization Orchestrator, Simulation Manger and Optimization Tool | 32 |
| Figure 30. Build result summary for Simulation and Optimization Orchestrator. | 33 |
| Figure 31. End-to-end test process for FREVO together with dummy SOO and SM components | 34 |
| Figure 32. FREVO release page. | 35 |
| Figure 33. Published Docker images of Gazebo Simulation manager on Dockerhub. | 37 |
| Figure 34. Published Docker images of Stage Simulation manager on Dockerhub. | 38 |
| Figure 35. Simple ROS catkin package structure..... | 39 |
| Figure 36. Code Generator build summary on the CI server..... | 39 |
| Figure 37. End-to-end test process, validating the behavior and compliance of Deployment Tool to the specification..... | 41 |
| Figure 38. Build result summary for CPSwarm Deployment Tool. | 42 |
| Figure 39. Latest published Docker image of Deployment Manager on Dockerhub. | 42 |
| Figure 40. Latest published Docker image of Deployment GUI on Dockerhub..... | 43 |
| Figure 41. Build result summary for CPSwarm Monitoring & Command Tool. | 44 |

References

- Atlassian. (2017, 11). *Atlassian Bamboo Open Source Project License Request*. Retrieved from Atlassian Bamboo: <https://www.atlassian.com/software/views/open-source-license-request>
- Atlassian. (2017, 11). *Bamboo Best Practice - Using Stages*. Retrieved from Atlassian Bamboo: <https://confluence.atlassian.com/bamboo/bamboo-best-practice-using-stages-388401113.html>
- Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 1--41.
- Sobe, A., Fehévari, I., & Elmenreich, W. (2012). FREVO: A tool for evolving and evaluating self-organizing systems. *Proceedings of the 1st International Workshop on Evaluation for Self-Adaptive and Self-Organizing Systems*. Lyon.
- Thomas Schmickl, H. H. (2011). *Bio-inspired Computing and Networking*. CRC Press.
- Yang, X.-S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.
- Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization. *Stochastic Algorithms: Foundations and Applications*, pp. 169-178.