D7.2 – FINAL CPSWARM ABSTRACTION LIBRARY

Deliverable ID	**D7.2**
Deliverable Title	**Final CPSwarm Abstraction Library**
Work Package	**WP7 Deployment Toolchain**

Dissemination Level	**PUBLIC**

Version	**1.0**
Date	**2020-01-03**
Status	**Final**

Lead Editor	**LINKS**
Main Contributors	**Gianluca Prato (LINKS), Angel Soriano (ROBOTNIK), Ákos Milánkovich (SLAB)**

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.

1 Executive Summary

This deliverable, "D7.2 Final CPSwarm Abstraction Library", gives a complete report of the design phase and implementation of CPSwarm Abstraction Library. Particular attention has been given to describe the strict connection among the actual implemented code – deployed on board of the CPS - and the models available in CPSwarm Modeling Library. Moreover, two sections are dedicated to the presentation of security aspects related to the communication part and to common settings to harden a ROS environment running on a Linux platform.

This deliverable reports on the results of Task 7.1 - CPSwarm Abstraction Library.

Document History

Version	Date	Author(s)	Description
0.1	2019-08-26	Gianluca Prato (LINKS)	First Draft with TOC
0.2	2019-11-15	Gianluca Prato (LINKS) Angel Soriano (ROBOTNIK) Ákos Milánkovich (SLAB)	Integrated contributions from ROBOTNIK and SLAB
0.21	2019-11-29	Gianluca Prato (LINKS)	Initial draft version ready
0.3	2019-12-09	Gianluca Prato (LINKS)	Updated content related to Sections 3
0.4	2019-12-21	Ákos Milánkovich (SLAB)	Updated content of Section 5 according to latest implementation, Section 4 refinement
0.5	2019-12-30	Gianluca Prato (LINKS)	Correction according to Robotnik's review
1.0	2020-01-03	Gianluca Prato (LINKS), Ákos Milánkovich (SLAB)	Correction according to Lake's review, Section 5 fixing, finalization

Internal Review History

Review Date	Reviewer	Summary of Comments
2019-12-26	Angel Soriano (ROBOTNIK)	Approved with minor comments
2019-12-30	Melanie Schranz (LAKE)	Approved with minor comments

Contents

2 Introduction

Autonomous robots are complex systems that require the interaction between numerous heterogeneous components (software and hardware). Because of the general complexity of robotic applications and the diverse range of hardware, many software libraries have been developed to promote the integration of new technologies hiding the complexity of low-level hardware. Furthermore, the development of these libraries has been pushed by the desire to improve software quality, ease the reuse of robotic software infrastructures across multiple research efforts, and to reduce production costs. A survey on some of the most popular robotic libraries have been used to assemble a base knowledge that drove the design and implementation of the Final CPSwarm Abstraction Library.

2.1 Document Organization

The document is organized as follows: firstly, Section 3 presents the software design of the CPSwarm Abstraction Library and describes the link of this component with the modeling part. Section 4 is dedicated to the description of the Communication Library with a focus on the implementation of the security features. Section 5 collects a list of security procedures to harden a ROS system. Finally, Section 6 reports the new implementations realized till M36 for the use case scenarios demonstrations.

2.2 Related documents

ID	Title	Reference	Date
[RD.1]	Final CPS modeling library	D4.3	M33
[RD.2]	Final Swarm Modeling library	D4.6	M34
[RD.3]	Initial CPSwarm Abstraction Library	D7.1	M18
[RD.4]	Final Monitoring and configuration framework	D7.6	M34
[RD.5]	Final Swarm of drones and ground robots demonstration	D8.2	M36
[RD.6]	Final Swarm Logistics demonstration	D8.3	M36

3 CPSwarm Abstraction Library

The CPSwarm Abstraction Library covers two different roles inside the project: First, it has to provide a set of CPS-specific adaptation libraries in order to access platform-specific components of a robotic system in a standard and coherent way. Moreover, in conjunction with the CPSwarm Swarm Library (see D4.6)[1], it provides support for the development of algorithms using a model-driven approach. This approach was promoted by the Consortium not only to ease the design of new CPS's behavior, but also to allow the translation of the modeled algorithm into actual code through the support of automatic code generation tools. In fact, arising the level of abstraction have represented inside the project a key concept to allow the development of a Code Generator (see D4.3) not dealing with the specific low-level details that characterize each specific robotic system.

The CPSwarm Abstraction Library allows to access the hardware provided by the CPS. It guarantees the support of controlling several types of sensors such as ultrasonic range sensors, cameras, or GPS, and driving actuators such as grippers, motors and servos. It raises the level of abstraction from a platform-dependent point of view to an application-oriented perspective. Furthermore, the Abstraction Library provides facilities to easily develop high-level routines. It shifts the focus of the developer from coding CPS specific implementations to swarm behavioral executions. This allows to concentrate on describing how the CPSs should behave in order to complete a high-level task or reach an application-specific goal.

3.1 Library Structure

As already presented in D7.1, the Abstraction Library has been organized as a composition of three layers (see Figure 1) where each layer adds a new level of abstraction on top of the previous one. First, the bottom most layer of *Hardware Drivers* gathers the software libraries that are responsible to enable the other layers to access the hardware functionalities. This layer constitutes the foundation of the Abstraction Library and includes all the drivers for sensors and actuators that are mounted on the CPSs.

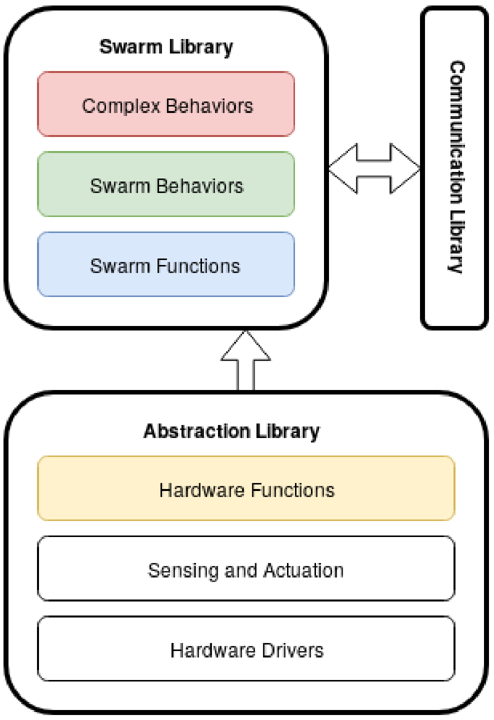

Figure 1 - The CPSwarm Abstraction Library structure and link with CPSwarm Swarm Library

[1] The 2 libraries compose the so-called Behavior Library.

Second, the *Sensing and Actuation* layer is responsible for providing sensor information and for controlling the CPSs using their actuators. While the Hardware Drivers layer has a direct connection with the hardware, this layer purely consists of software that contributes to supply a first degree of abstraction by realizing complex functionalities required by the overlying layer. Depending on the available computational power, memory, and hardware resources, this layer can be presented as an independent component or incorporated inside the Hardware Drivers layer.

Finally, the topmost layer of *Hardware Functions* exhibits a set of high-level functions corresponding to complex routines that a CPS can execute involving a set of sensors and actuators. Each function interacts with the lower layers for sending actuator commands and requesting sensor information. Each function of this layer constitutes a base building block to define a state of the FSMs. A single state can be associated to a specific function of the Abstraction Library that will be executed while the state is active. Therefore, the mapping of robot's functionalities to specific software modules can guarantee the reusability of those functionalities. In this way, the effort to develop new CPS applications will be significantly reduced letting the developers re-use some existing solution and focusing their attention to more application-specific problems.

The components of the CPSwarm Abstraction Library developed among the project and released as open source are available on GitHub[2]. As explained in previous version of this deliverable D7.1, the implementation of all the three layers of the library has been realized using ROS as the base framework and common ROS conventions have been followed.

All data coming from the sensors are continuously streamed through the ROS topics[3] facilities. Following a single sender multiple receiver arrangement, each component in the stack that is interested in processing a particular type of data (e.g. the current position, the speed, …) can subscribe to the dedicated topic to receive periodic updates. This method can also be applied to implement virtual sensor: a processing module can collect information coming from different sources, elaborate them and finally re-publish the result on a new topic. The publish and subscribe mechanism provided by ROS is used also by the Communication Library (more details are available in the dedicated section 4.3.3) in order to gather the telemetry information and provide them to the CPSwarm Monitoring Tool.

The dispatch of direct commands and requests for specific information are managed through the ROS Services[4]. Indeed, the client-server scheme fits well to manage this request/reply interaction between two specific components.

Finally, for components that belong to the Hardware Functions a mixed approach has been preferred:

- Short running tasks, such as moving up and an elevator or letting a drone take off, can be activated using Services.
- For long running and computational expensive operations (e.g. path following or a target research), the ROS actionlib[5] package has been used. In fact, ROS Actions provide a feedback mechanism that is very useful to check the current status of a long activity during the execution.

[2] https://github.com/cpswarm
[3] http://wiki.ros.org/Topics
[4] http://wiki.ros.org/Services
[5] http://wiki.ros.org/actionlib

3.2 Connection with CPSwarm Modeling Library

In strong collaboration with tasks "T4.3 - Swarm modeling" and "T5.4 - Code generation for CPS systems", a comprehensive analysis of how the models in the Modeling Tool and the code in the Behavior Library (which includes the Abstraction Library itself) was realized.

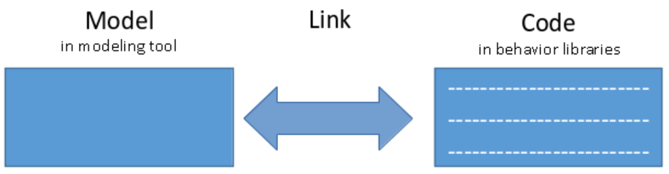

Figure 2 - Correspondence between the models and the behaviors implemented as code

A detailed description of this activity has been presented in D5.4. The main objective of such work was to identify the necessary set of files to guarantee the correct synchronization among the components available at the modeling level with the actual code deployed on the CPS. In fact, without this synchronization, a coherent flow from the application design, realized in the Modeling Tool, to the final code deployment on board of the CPS cannot be provided. As a result of these research activities, the needs for a specific file format to describe the components of the Abstraction Library was identified. Such format was defined and here presented as an achievement of T7.1 and so-called Abstraction Description File.

3.2.1 CPSwarm Abstraction Description File

The Abstraction Description File is a new addition to the Abstraction Library. It describes the available APIs on the Abstraction Library, e.g. what sensors/actuators are available or what basic functionalities the device can perform.

In order to identify the best solution for this description file, an evaluation of already existing formats was performed. A promising research trend attempted to model CPS capabilities as Web services [1][2]. Web services present an effective approach for providing abstractions to ROS resources[6]. Web services can be used to provide an abstract specification of ROS-enabled robot services by taking advantage of the WSDL[7] standard for SOAP Web services and the WADL[8] standard for REST Web services. Another interesting solution was proposed in [3] to model ROS nodes, and robotic architectures in general, using the Architecture Analysis and Design Language[9] (AADL) and by deriving, from these models, reusable templates to streamline the design of robotic systems. Nevertheless, none of those proposed formats seemed able to fit the descriptive requirement needed for the project. A first try to describe ROS APIs through WADL and WSDL format was done but, because of the high flexibility of ROS environment, both were found to be too restrictive. Therefore, a specific format has been defined using the JSON Data Interchange Standard[10].

The CPSwarm Abstraction Description File is composed by 3 basic properties:

- *runtime-env*: this property identifies the runtime software environment whose APIs are described using the ADF format. The current version has, by default, "ROS" as constant value. How to correctly parse the content of the ADF is strictly dependent on the value of this property as every runtime environment will adopt specific convention in the APIs definition.
- *sensors/actuators*: this property contains a list of all the sensors and actuators on board of the described CPS. In relation to the different layer of the Abstraction Library, the list will contain

[6] ROS was considered because of his adoption in the relevant use case scenarios.
[7] https://www.w3.org/TR/wsdl.html
[8] https://www.w3.org/Submission/wadl
[9] https://en.wikipedia.org/wiki/Architecture_Analysis_%26_Design_Language
[10] https://www.json.org

information corresponding to the Hardware Drivers and Sensing/Actuation layers. In the particular case of the ROS environment, each sensor/actuator is described by the following properties:

- *name*: name of the described sensor/actuator.
- *description*: brief description of the sensor/actuator
- *category*: category of the described hardware component. Possible values are: "Sensor", "Actuator" or "Virtual" (to identify virtual sensors).
- *api*: this property contains the description of the API related to the described component and specific for the selected runtime environment. In the case of ROS, the APIs correspond to the topics where the sensor/actuator will publish specific information and the services provided by the component.
- *functions*: this property lists the set of high-level functionalities executable by the CPS. As presented in section 3.1, for ROS systems this is provided through ROS services or actions. Each functionality will present a list of its inputs and outputs.

The complete structure of the Abstraction Description File can be analyzed consulting the full schema[11] in ANNEX A. Moreover, a simple example of an ADF describing a drone is available in ANNEX B.

[11] https://json-schema.org

4 Communication Library

The Communication Library provides a unified interface tool that swarm members can use to interact with each other. It is the duty of the library to ensure that the entire communication happens with the desired reliability, security level and latency.

4.1 Introduction

After evaluating the requirements established by the project's use cases and the design goal of swarms in general, we concluded that interactions have a well-defined set of primitives and actions:

- Swarm members need to be discoverable on the network.
- Events and commands need to be sent and received.
- Parameters need to be remotely adjustable.
- Telemetry needs to be sent back to operators and other subscribers.

The Consortium aimed to provide a stable API for all tools that abstracts away the physical layer and the authentication scheme used. To do this, a pluggable architecture was designed for the Communication Library, which separates the logical layer responsible for implementing these primitives and the endpoint implementation capable of sending individual messages over the network. This extensible infrastructure makes it possible to add support for new low-level protocols, physical layers and security schemes without affecting the rest of the system. As a first step, the Zyre[12] protocol was integrated with the library, but as the project progressed, a secure endpoint was added as well.

Tools and any custom software developed by users can link against the public C++ API of the library and can participate in the swarm on equal terms with other swarm members. While swarm management tools such as those integrated with the CPSwarm Workbench seem to be the primary target for this API, IoT devices can also run custom software capable of participating in the swarm in order to realize IoT2Swarm interactions and to offer a physical interface for operators.

Since the Search and Rescue and Logistics use cases involve ROS, special care was taken to perform integration into a ROS environment, which manifests in a ROS node capable of bridging native ROS IPC facilities with the primitives supported by the Communications Library. Among other things, this makes it possible to adjust the parameters of ROS packages unrelated to the CPSwarm project and to observe through telemetry the topics published by any of the ROS components. Since a typical ROS-based use case will involve many third party packages (e.g., in order to support hardware components), this will, in many cases, spare the user the effort of having to use a separate facility to interact with their topics and parameters or to write such a bridge manually.

End users may also implement any scheme they think would better fit their use case without disrupting any of the other components - thanks to the pluggable architecture.

While it is not the primary goal of the library to provide real-time or near real-time performance, such an endpoint type can be developed in theory. We aim to benchmark the performance of the implementations that will be provided with the final release to help end users evaluate whether they fit their requirements. The current Zyre based implementation suggests a 10ms overhead on a request-reply type exchange over standard IP based networks.

[12] https://github.com/zeromq/zyre

4.2 Key concepts

High level C++ API

- Abstracts away the physical and transport layers
- Responsible for reliable delivery and fault detection
- Exposes functionality through services

Protobuf[13]-based serialization

- The API works with Protobuf objects directly
- Objects can be re-serialized at any point in the communication flow
- Complex data types can be defined (area, route, etc.)

Cross-platform

- Uses only C++ standard library primitives and other cross-platform libraries
- Compatible with ROS[14]

4.3 Library

The Library contains the required source code for integration into your project[15].

4.3.1 Endpoints

- Used to abstract away the transport and physical layer
- Endpoint implementations based on BasicEndpoint only need to implement:
  - Starting and stopping the endpoint
  - Sending binary messages
  - Receiving binary messages
  - Tracking the presence of nodes
- While we are targeting IP networks (including mesh networks), the library doesn't care about the medium
- Zyre-based endpoint implementation without security features
- The secure endpoint is an extension of ZyreEndpoint based on libsodium[16]

4.3.2 Services

- A combination of well-defined functionality and related data types
- Asynchronous, thread-safe interfaces
- Currently available services:
  - Discovery: the Discovery Service is responsible for detecting the supported features of participating swarm members. In order to make the Monitoring and Configuration Tool a universal tool for the management of compatible swarms, regardless of specific behavior or target hardware, the Communication Library provides a way to obtain a description of the

[13] https://developers.google.com/protocol-buffers
[14] https://www.ros.org/about-ros
[15] https://github.com/cpswarm/swarmio
[16] https://libsodium.gitbook.io/doc

events supported by each member and of the different telemetry and parameter values and their underlying data types. The Discovery Service works on two layers: the lower layer, provided by the specific endpoint implementation, is purely responsible for detecting the presence of swarm members and tracking their online-offline states, while the higher layer can request and answer, as well as cache and invalidate information about the supported facilities.

- Event (commands and global events): swarm members send and receive events during behavior execution, informing other members of important events and reacting to external and internal stimuli in order to change or modify the current state of execution. An event, on its own, has only a name and a list of parameters, it is only how the behavior reacts that makes the event meaningful. As such, events can represent commands issued by the operator, real events happening on a local or remote node or other simple messages that aid coordination. The Monitoring and Configuration Tool can use the Event Service to send arbitrary events to swarm members (in order to issue commands) and can monitor events as they are happening on swarm members.
- Key-Value (parameters and other mostly static data): parameters such as the operational area or the location of known obstacles are subject to change during deployment, and as such, need a way to be set during the mission. The Key-Value Service provides a way to write (and read) complex named values on swarm members. The behavior can use these values to perform calculations and to make decisions. The Monitoring and Configuration Tool uses the Key-Value Service to retrieve and set the parameters that govern swarm member behavior.
- Telemetry (streaming data to subscribers): for the operator to receive meaningful information about the state of each swarm member, a continuous stream of information needs to be sent by the swarm members being monitored by the Monitoring and Configuration Tool, and eventually, by the operator. The Telemetry Service can be used to subscribe to such information on-demand, specifying the required resolution and scope of the information. All data sent back is strongly typed and can have complex schema. Each telemetry value (however complex) is treated as an atomic value relevant to a single point in time. The Monitoring and Configuration Tool uses the Telemetry Service to display and visualize the key elements describing the state of individual swarm members.
- Ping (measuring latency): for debug purposes the ping service is able to test if a connection is successfully established among swarm members. Additionally, the measurement of end-to-end communication latency is performed to gain insights.

4.3.3 **ROS bridge**

Since our primary targets are ROS-based devices, support for ROS native facilities is needed. Communication within ROS uses a proprietary messaging format not available on non-ROS systems. A bridge node was developed, which is capable of translating between a ROS-based system and other software entities:

- Publishing any ROS topic as telemetry
- Forwarding events to and from the behavior
- Setting parameters on the ROS Parameter Server

Using the communication node, developed applications or behavior generated for ROS-based devices can use native ROS facilities and do not need to take care about the presence of the library. The bridge is just another application using the library. It receives no special treatment.

- Using standard ROS facilities to communicate
  - Bridge the Key-Value Service to ROS Parameter Server
  - Transfer events and telemetry through ROS publish-subscribe

- Bind to ROS resources as defined in a configuration file
    - Should be part of the deployment package
    - Reloadable without interruption
- Should be one of the first things to install on a node during provisioning
    - Cryptographic proof of swarm membership will need to be established
    - Network interfaces need to be configured

An example configuration can be seen here[17].

4.3.4 Simulator

The Swarmio-Simulator is an example of using the Library with ZyreEndpoint capable of discovering and sending a predefined telemetry message simulating a linear-path movement.

4.3.5 Tool

The Swarmio-Tool is a management component with the following available commands:

- members: lists the members of the swarm
- rediscover: sends a discover message to trigger rediscover process
- info: lists UUID, device class, available parameters for subscription about a selected node
- select [MID]: selects a member by ID for further commands
- event NAME [KEY=VALUE]: sends an event with key, value pair
- get KEY: requests a key
- set KEY=VALUE: sets the corresponding value of a key to VALUE
- subscriptions: lists the current subscriptions
- subscribe [key=KEY] [interval=N]: subscribes the tool to a specific key to receive updates on that topic
- unsubscribe [SID]: unsubscribe from a previous subscription by subscription id
- ping [SIZE]: sends a ping message with SIZE size
- help: displays this list
- log: lists the log entries of communication
- exit: stops the program

4.4 Secure Communication Library

During the second half of the project the library has been extended with new security features.

4.4.1 Requirements for message types:

Four different criterions have been evaluated for each different type of message managed by the Communication Library:

- **Reliable**: if the message has been delivered to the recipient (trying multiple times as necessary), an acknowledgement is sent back to the sender.

[17] https://github.com/cpswarm/swarmio/blob/master/swarmros/resources/swarmros.cfg.example

- **Multicast**: A message is sent to multiple parties at once (based on physical location/address group).
- **Confidential**: the message is encrypted; the content is only retrievable via the knowledge of the key.
- **Authenticated**: the sender of the message is verified.

Message types and their requirements	Reliable	Multicast	Confidential	Authenticated
Event *an event has occurred on one of the swarm members that needs to be propagated*	Yes	Yes	Yes	Yes
Command *the Monitoring and Configuration Tool has raised a remote event on a specific swarm member*	Yes	No	Yes	Yes
Artefact *the Deployment Tool has sent a software artefact that needs to be deployed on the swarm member*	Yes	No	Yes	Yes
Status *the swarm member has made progress deploying the software artefact*	Yes	No	Yes	Yes
Set / Get *the Monitoring and Configuration Tool has sent a request to get or set the value for a global parameter of the behaviour*	Yes	No	Yes	Yes
Subscribe / Unsubscribe *the Monitoring and Configuration Tool wants to subscribe to or unsubscribe from updates on a property*	Yes	No	Yes	Yes
Telemetry *the swarm member has sent an update for the value of a property to a subscriber*	No	No	Yes	Yes

Table 1 - Communication Library message types and related requirements

Please note that response messages, which only include a confirmation that the operation has completed successfully, are not included and that the descriptions in italic are only examples for how such a message might be used. According to the requirements above, most of the messages are sent unicast (only the events are multicast). Acknowledgments are also required for almost every message type (except Telemetry). If authentication is required, encryption is also necessary.

4.4.2 **Implementation**

Security features can be implemented at various layers of the OSI model18. We propose to use the presentation layer (layer 6) to enable confidentiality, integrity protection and authentication.

On a lower level, in order to facilitate discovery and to provide a way for swarm members and workbench tools to keep track of the current composition of the swarm, a discovery mechanism is implemented by Zyre. Additional security functionality (like initial authentication and key exchange) can also be implemented as an extension of the discovery process. Initial discovery is currently multicast over UDP, while responses arrive as unicast messages over TCP according to ZRE[19] specification.

The security functionalities are to be provided by libsodium (based on NaCl[20]). Libsodium is a popular solution for crypto library used by e.g.: WordPress, Discord, Secrets, Remembear. All cryptographic functions are based on:

- Edwards-Curve Digital Signature Algorithm (EdDSA)[21]
- Encryption: XSalsa20[22] stream cipher
- Authentication: Poly1305[23] MAC

The following security dimensions are addressed:

- The Deployment tool is able to securely provision new node members (by generating their keys and signing their certificates).
- Access control is provided for provisioned nodes by certificate checking, using a pre-shared signing key.
- Authentication is provided by signature checking.
- Non-repudiation is provided by signature and timestamp checking for each packet.
- Confidentiality is provided end-to-end by payload encryption.
- Integrity checking is provided by using a tag for packet integrity.
- Availability is maintained using each nodes security table, which stores valid authentication credentials.

Communication overhead of security features has a constant size of 36 bytes (16 bytes nonce, 20 bytes authentication tag) for every message (except for discovery and ACK).

4.4.2.1 ***Secrets***

The following secrets are used as building blocks to provide secure communication.

4.4.2.2 ***Signing key of deployment tool***

Public-private key pair (SK_pri, SK_pub) is generated by the Deployment tool at initialization of a swarm (and never changed during its lifetime). The private key is used in Deployment tool only, creating trusted certificates for swarm members. The public key is shared by swarm members for checking signatures.

[18] https://en.wikipedia.org/wiki/OSI_model
[19] https://rfc.zeromq.org/spec:36/ZRE/
[20] http://nacl.cr.yp.to/
[21] https://tools.ietf.org/html/rfc8032
[22] https://en.wikipedia.org/wiki/Salsa20
[23] https://en.wikipedia.org/wiki/Poly1305

4.4.2.3 *Communication key pair of swarm member*

The public-private communication key pair (CK_pri, CK_pub) is generated for each new member at the phase of adding the new member to the swarm. They are used for creating unique session keys during key exchange. The certificate of each swarm member contains CK_pub.

4.4.2.4 *Certificate of swarm member*

The certificate is member-unique, generated by the Deployment tool at the phase of adding a new member to the swarm. It encapsulates the public communication key, validity timestamp, roles, memberships, etc. It is encrypted / signed by using the SK_pri key and can be decrypted/validated with SK_pub.

4.4.2.5 *Unicast session key pair*

The session-unique unicast key pair (UK_rx, UK_tx) is used by each pair of communicating swarm members to encrypt traffic sent to each other (one's rx key is the others tx key), the keys are generated by the unicast key exchange process during joining the swarm.

4.4.2.6 *Multicast key*

Device-unique multicast key MK is used only for transmitting multicast messages. The multicast key, which is used for symmetric key encryption, is sent to the connected partners encrypted by their session key after the unicast key exchange process. Only devices that have previously received the multicast key from the sender will be able to decrypt the multicast messages. The device can change the multicast key after a set time period or number of uses, then send the new key to the connected swarm members.

4.4.3 **Secrets definition process**

The following processes are built on the secrets defined above to create a secure communication environment.

4.4.3.1 *Deployment of a swarm member*

On deployment, the Deployment tool generates the unique communication key pair, and a member certificate. The device should store its SK_pub, certificate, and its communication key pair, and the multicast key.

4.4.3.2 *Unicast key exchange/update*

The swarm members are required to exchange keys before they are able to securely communicate unicast. Initial key exchange can be done during the discovery phase. Using libhydrogen's KK key exchange API[24], a client and a server can securely generate and exchange session keys (see Figure 3):

1. The initiator (client) sends its certificate.
2. The listener (server) receives the certificate, validates it by using SK_pub, and sends its own certificate back. (Now the server knows the client's public key.)
3. The client receives the server's certificate and validates it. (Now the client knows the server's public key.) Then the client sends the unique session key request in packet_3.
4. The server validates the packet using the keys exchanged, computes the session keys and sends a response to the client in packet_4.
5. The client uses this packet and previous data in order to compute the same session key pair.

Two session keys are computed on both sides. One can be used to encrypt data sent from the client to the server; the latter can be used in the other direction. Keys can be refreshed at every start-up or after a certain

[24] https://github.com/jedisct1/libhydrogen/wiki/KK-variant

number of messages sent. Key storage is out of scope in communication library: hardening will deal with it. The first two steps are only necessary in case the swarm members don't know each other's public keys.

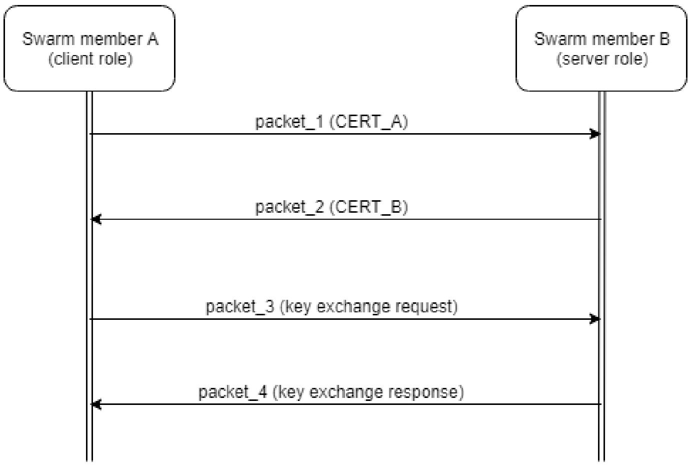

Figure 3 - Key exchange among swarm members

4.4.4 Configuration

All the security features are activated by adding a config.json file next to the executable with the following contents (example):

```
{
        "privateKey": "cSKuVIqy6iR6bW9F1xLgv/B0e5cqyge/OBLIHd9PS2M=",
        "publicKey": "Hh8yd4dIUbeNM90xEhtdmjP12hcQPGepfcIeyEq8zfE=",
        "signature":
"kgJnRa/7/KrErXZ1lobmV/XVcacE94A1+KwWfxF4zjbroafwrU0PYkEZEC0C5afT6XJDNS/q3TjZun3F6gxXBA
==",
        "ca": "utGq0AIAV+c3JmwOsS5/h2T6mp331GD8WQhMzcPyzGs="
}
```

In case the security features are activated, the nodes are only able to communicate in a secure way (there is no selective approach to special message types).

5 ROS Hardening

This section describes some general, UNIX and ROS-specific recommendations to provide a more secure ROS environment for the CPSwarm project.

The Monitoring and Command Tool and the Deployment Tool rely on services provided by the Communication Library and most of the swarm devices are based on ROS. The secure version of the Communication Library is described by the previous chapter. To provide a more secure ROS environment we hereby include a list of hardening methods for ROS. These measures ensure that no unnecessary and potentially vulnerable attack surfaces are present on the devices and thus the probability of manipulated communication is significantly lowered. The "Value" in the sections below describes the configuration/setting in vanilla ROS Melodic Morena.

5.1 General

This section describes general Linux hardening suggestions.

5.1.1 Use a suitable SCAP profile as a baseline configuration

The Security Content Automation Protocol (SCAP) is a synthesis of interoperable specifications derived from community ideas. SCAP enables security administrators to define security policies and provides a way to express, organize, and manage security guidance while scanning the computers and software to determine if the configuration and software patches are implemented to the specified security policies that they are implemented to.

Implementation:

For implementation see the reference.

Reference:

http://www.open-scap.org/

https://www.techrepublic.com/article/how-to-perform-security-audits-on-ubuntu-server-with-openscap/

5.1.2 Use a hardening script or a security auditing tool such Lynis

The systems should be scanned for common security configuration issues, for a consistent approach to some base-level security.

Implementation:

https://cisofy.com/documentation/lynis/get-started/

5.2 Network Configuration and Firewalls

This section describes hardening suggestions related to network configuration and firewall settings.

5.2.1 Configure Kernel Parameters Which Affect Networking

5.2.1.1 *Disable Kernel Parameter for Sending ICMP Redirects by Default*

ICMP (Internet Control Message Protocol) redirect is a mechanism for routers to convey routing information to hosts that a more direct route exists for a particular destination. These messages contain information from the system's route table possibly revealing portions of the network topology.

The capability to send ICMP redirects is only applicable for systems acting as routers.

Implementation:

To set the runtime status of the net.ipv4.conf.default.send_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.send_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.send_redirects = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 0.

Reference:

https://askubuntu.com/questions/118273/what-are-icmp-redirects-and-should-they-be-blocked

https://wiki.ubuntu.com/ImprovedNetworking/KernelSecuritySettings

5.2.1.2 *Disable Kernel Parameter for Sending ICMP Redirects for All Interfaces*

ICMP redirect is a mechanism for routers to convey routing information to hosts that a more direct route exists for a particular destination. These messages contain information from the system's route table possibly revealing portions of the network topology.

The capability to send ICMP redirects is only applicable for systems acting as routers.

Implementation:

To set the runtime status of the net.ipv4.conf.all.send_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.send_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

```
net.ipv4.conf.all.send_redirects = 0
```

Then run the following command to apply kernel parameter modifications:

```
$ sudo sysctl –p
```

Value:

Its value was 0.

Reference:

https://askubuntu.com/questions/118273/what-are-icmp-redirects-and-should-they-be-blocked

https://wiki.ubuntu.com/ImprovedNetworking/KernelSecuritySettings

5.2.1.3 ***Disable Kernel Parameter for IP Forwarding***

Routers generally use routing protocol daemons that are used for exchanging network routing information with other routers. This ability should be used only when it is required, otherwise system network information may be unnecessarily transmitted across the network.

Implementation:

To set the runtime status of the net.ipv4.ip_forward kernel parameter, run the following command:

```
$ sudo sysctl -w net.ipv4.ip_forward=0
```

If this is not the system's default value, add the following line to /etc/sysctl.conf:

```
net.ipv4.ip_forward = 0
```

Then run the following command to apply kernel parameter modifications:

```
$ sudo sysctl –p
```

Value:

Its value was 0.

Reference:

https://askubuntu.com/questions/311053/how-to-make-ip-forwarding-permanent

https://openvpn.net/faq/what-is-and-how-do-i-enable-ip-forwarding-on-linux/

5.2.1.4 ***Configure Kernel Parameter for Accepting Source-Routed Packets for All Interfaces***

With using source-routed packets, it becomes possible for the source of the packet to advise different path for the routers for forwarding the packet, than it is preconfigured on the router. With this, it is feasible to bypass network security measures. This requirement applies only to the forwarding of source-routed traffic, such as when IPv4 forwarding is enabled and the system is functioning as a router.

In the IPv4 protocol accepting source-routed packets has some legitimate uses. If it is not totally required, it should be disabled.

Implementation:

To set the runtime status of the net.ipv4.conf.all.accept_source_route kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.accept_source_route=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.all.accept_source_route = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-server_security-disable-source-routing

https://blog.scottlowe.org/2013/05/29/a-quick-introduction-to-linux-policy-routing/

5.2.1.5 ***Configure Kernel Parameter for Accepting ICMP Redirects for All Interfaces***

ICMP redirect is a mechanism for routers to convey routing information to hosts that a more direct route exists for a particular destination. These kinds of messages are unauthenticated and modifies the host's routing table. With an illegitimate ICMP redirect message is possible to cause a man-in-the-middle attack.

In the IPv4 protocol this feature has some legitimate uses. If it is not totally required, it should be disabled.

Implementation:

To set the runtime status of the net.ipv4.conf.all.accept_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.accept_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.all.accept_redirects = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://askubuntu.com/questions/118273/what-are-icmp-redirects-and-should-they-be-blocked

https://wiki.ubuntu.com/ImprovedNetworking/KernelSecuritySettings

5.2.1.6 ***Configure Kernel Parameter for Accepting Secure Redirects for All Interfaces***

In the IPv4 protocol accepting "secure" ICMP redirects (from those gateways listed as default gateways) has few legitimate uses. If it is not totally required, it should be disabled.

Implementation:

To set the runtime status of the net.ipv4.conf.all.secure_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.secure_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.all.secure_redirects = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-server_security-disable-source-routing

https://blog.scottlowe.org/2013/05/29/a-quick-introduction-to-linux-policy-routing/

5.2.1.7 ***Configure Kernel Parameter to Log Martian Packets***

The sign of a villainous network activity could be the existence of "martian" packets () just like spoofed packets, source routed packets and redirects. With logging these kinds of packets can help in detecting these activities.

Implementation:

To set the runtime status of the net.ipv4.conf.all.log_martians kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.log_martians=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.all.log_martians = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 0.

Reference:

https://www.cyberciti.biz/faq/linux-log-suspicious-martian-packets-un-routable-source-addresses/

5.2.1.8 ***Configure Kernel Parameter to Log Martian Packets By Default***

The sign of a villainous network activity could be the existence of "martian" packets () just like spoofed packets, source routed packets and redirects. With logging these kinds of packets can help in detecting these activities.

Implementation:

To set the runtime status of the net.ipv4.conf.default.log_martians kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.log_martians=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.log_martians = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 0.

Reference:

https://www.cyberciti.biz/faq/linux-log-suspicious-martian-packets-un-routable-source-addresses/

5.2.1.9 ***Configure Kernel Parameter for Accepting Source-Routed Packets By Default***

With using source-routed packets, it becomes possible for the source of the packet to advise different path for the routers for forwarding the packet, than it is preconfigured on the router. With this, it is feasible to bypass network security measures.

In the IPv4 protocol accepting source-routed packets has some legitimate uses. If it is not totally required, it should be disabled, such as when IPv4 forwarding is enabled and the system is functioning as a router.

Implementation:

To set the runtime status of the net.ipv4.conf.default.accept_source_route kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.accept_source_route=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.accept_source_route = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_network_access

5.2.1.10 ***Configure Kernel Parameter for Accepting ICMP Redirects By Default***

ICMP redirect is a mechanism for routers to convey routing information to hosts that a more direct route exists for a particular destination. These kinds of messages are unauthenticated and modify the host's routing table. With an illegitimate ICMP redirect message is possible to cause a man-in-the-middle attack.

In the IPv4 protocol this feature has some legitimate uses. If it is not totally required, it should be disabled.

Implementation:

To set the runtime status of the net.ipv4.conf.default.accept_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.accept_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.accept_redirects = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_network_access

5.2.1.11 ***Configure Kernel Parameter for Accepting Secure Redirects By Default***

In the IPv4 protocol accepting "secure" ICMP redirects (from those gateways listed as default gateways) has few legitimate uses. If it is not totally required, it should be disabled.

Implementation:

To set the runtime status of the net.ipv4.conf.default.secure_redirects kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.secure_redirects=0

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.secure_redirects = 0

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_network_access

5.2.1.12 ***Configure Kernel Parameter to Ignore ICMP Broadcast Echo Requests***

Responding to broadcast (ICMP) echo can help in network mapping and provides a vector for amplification attacks. It makes the system somewhat more difficult to enumerate on the network if ICMP echo requests (pings) are ignored.

Implementation:

To set the runtime status of the net.ipv4.icmp_echo_ignore_broadcasts kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.icmp_echo_ignore_broadcasts = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://www.theurbanpenguin.com/broadcast-icmp-in-linux-and-how-to-initiate-and-protect/

5.2.1.13 ***Configure Kernel Parameter to Ignore Bogus ICMP Error Responses***

To reduce log size, it is possible to ignore bogus ICMP error responses, but with this some activity would not be logged.

Implementation:

To set the runtime status of the net.ipv4.icmp_ignore_bogus_error_responses kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.icmp_ignore_bogus_error_responses=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.icmp_ignore_bogus_error_responses = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://www.stigviewer.com/stig/oracle_linux_6/2016-12-20/finding/V-50663

5.2.1.14 ***Configure Kernel Parameter to Use TCP Syncookies***

TCP SYN flood attack with filling a system's TCP connection table with connections in the SYN_RCVD state can cause a denial of service. When a subsequent ACK is received syncookies can be used to track a connection. Verifying the initiator is attempting a valid connection and is not a flood source. This feature enables the system to continue servicing valid connection requests and it is activated when a flood condition is detected.

Implementation:

To set the runtime status of the net.ipv4.tcp_syncookies kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.tcp_syncookies=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.tcp_syncookies = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

<u>*Reference:*</u>

https://www.cyberciti.biz/faq/enable-tcp-syn-cookie-protection/

https://geode.apache.org/docs/guide/19/managing/monitor_tune/disabling_tcp_syn_cookies.html

5.2.1.15 *Configure Kernel Parameter to Use Reverse Path Filtering for All Interfaces*

Packets with source addresses that should not have been able to receive on the interface they were received on can be dropped by enabling reverse path filtering. This is helpful for end hosts and routers that are serving in small networks, but it shall not be used on systems that are routers for complex networks.

<u>*Implementation:*</u>

To set the runtime status of the net.ipv4.conf.all.rp_filter kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.all.rp_filter=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.all.rp_filter = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

<u>*Value:*</u>

Its value was 1.

<u>Reference:</u>

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_network_access

5.2.1.16 *Configure Kernel Parameter to Use Reverse Path Filtering by Default*

Packets with source addresses that should not have been able to receive on the interface they were received on can be dropped by enabling reverse path filtering. This is helpful for end hosts and routers that serving in small networks, but it shall not be used on systems that are routers for complex networks.

<u>*Implementation:*</u>

To set the runtime status of the net.ipv4.conf.default.rp_filter kernel parameter, run the following command:

$ sudo sysctl -w net.ipv4.conf.default.rp_filter=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv4.conf.default.rp_filter = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 1.

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_network_access

5.2.2 IPv6

5.2.2.1 *Disable Support for IPv6 Unless Needed*

Enable only that kind of IP protocol versions (e.g. only IPv4) that are specified in the system communication matrix. It can highly increase the attack surface if it has undocumented running and accessible protocols.

Implementation:

To disable support for (ipv6) run the following command:

$ sudo sysctl -w net.ipv6.conf.all.disable_ipv6=1

If this is not the system's default value, add the following line to /etc/sysctl.conf:

net.ipv6.conf.all.disable_ipv6 = 1

Then run the following command to apply kernel parameter modifications:

$ sudo sysctl –p

Value:

Its value was 0.

Reference:

https://linuxconfig.org/how-to-disable-ipv6-address-on-ubuntu-18-04-bionic-beaver-linux

5.2.2.2 *Disable IPv6 Networking Support Automatic Loading*

To reduce the vulnerability to exploitation all unnecessary network stacks even IPv6 should be disabled.

Implementation:

For implementation see the reference.

Reference:

https://linuxconfig.org/how-to-disable-ipv6-address-on-ubuntu-18-04-bionic-beaver-linux

5.2.2.3 ***Disable Automatic Configuration***

5.2.2.3.1 Configure Accepting IPv6 Router Advertisements

An illegitimate router advertisement message could result in a man-in-the-middle attack.

Implementation:

To set the runtime status of the net.ipv6.conf.all.accept_ra kernel parameter, run the following command:

```
$ sudo sysctl -w net.ipv6.conf.all.accept_ra=0
```

If this is not the system's default value, add the following line to /etc/sysctl.conf:

```
net.ipv6.conf.all.accept_ra = 0
```

Then run the following command to apply kernel parameter modifications:

```
$ sudo sysctl –p
```

Value:

Its value was 1.

Reference:

https://help.ubuntu.com/community/NetworkConfigurationCommandLine/Automatic

5.2.2.3.2 Configure Accepting IPv6 Redirects By Default

An illegitimate ICMP redirect message could result in a man-in-the-middle attack.

Implementation:

To set the runtime status of the net.ipv6.conf.all.accept_redirects kernel parameter, run the following command:

```
$ sudo sysctl -w net.ipv6.conf.all.accept_redirects=0
```

If this is not the system's default value, add the following line to /etc/sysctl.conf:

```
net.ipv6.conf.all.accept_redirects = 0
```

Then run the following command to apply kernel parameter modifications:

```
$ sudo sysctl –p
```

Value:

Its value was 1.

Reference:

https://help.ubuntu.com/community/NetworkConfigurationCommandLine/Automatic

5.2.3 Firewall

5.2.3.1 *Verify ufw Enabled*

Access control methods has the ability to enhance system security posture to prevent connections from unknown hosts and protocols, by restricting services and known good IP addresses and address ranges.

Implementation:

The ufw service can be enabled with the following command:

$ sudo ufw enable

Value:

It was active.

Reference:

https://help.ubuntu.com/community/UFW

5.2.3.2 *Use packet filters (e.g. iptables) as host firewall*

Using a packet filter will allow restricting access to a system by IP address, TCP/UDP port, or bad TCP/UDP packets, therefore it is highly recommended to protect a system using a packet filter on the host.

Implementation:

On Ubuntu you can also use iptables as the packet filter. See the referenced guide.

Reference:

https://help.ubuntu.com/community/IptablesHowTo

5.2.3.3 *Strengthen the Default Ruleset*

The ruleset always has to be hardened to allow only the necessary traffic, preferably in a stateful manner whenever applicable depending on the configuration and communication matrix.

5.3 Services

This section describes hardening suggestions related to standard linux services.

5.3.1 APT service configuration

5.3.1.1 *Disable unauthenticated repositories in APT configuration*

Unauthenticated repositories should not be used for updates. Repositories host all packages that will be installed on the system during update. If a repository is not authenticated, the associated packages can't be trusted, and then should not be installed locally.

Implementation:

The default behaviour is that it does not allow installing packages from untrusted sources, although there are some ways and options to change this behaviour, but all needs root privileges.

Reference:

https://static.open-scap.org/ssg-guides/ssg-ubuntu1604-guide-anssi_np_nt28_minimal.html

5.3.2 Deprecated services

5.3.2.1 *Uninstall the nis package*

The support for Yellowpages should not be installed unless it is required. NIS is the historical SUN service for central account management, more and more replaced by LDAP. NIS does not support efficiently security constraints, ACL, etc. and should not be used.

(It was not installed)

Implementation:

sudo apt-get remove --auto-remove nis

Value:

It was not installed.

Reference:

https://static.open-scap.org/ssg-guides/ssg-ubuntu1604-guide-anssi_np_nt28_minimal.html

https://www.howtoinstall.co/en/ubuntu/trusty/nis?action=remove

5.3.2.2 *Uninstall the telnet server*

The telnet daemon should be uninstalled unless it is required. The telnet allows clear text communications and does not protect with encryption any data transmission between client and server. Any confidential data can be listened, and no integrity checking is made.

Implementation:

sudo apt-get remove --auto-remove telnetd

Value:

It was not installed.

Reference:

https://static.open-scap.org/ssg-guides/ssg-ubuntu1604-guide-anssi_np_nt28_minimal.html

https://www.howtoinstall.co/en/ubuntu/xenial/telnetd?action=remove

5.3.2.3 ***Uninstall the SSL compliant telnet server***

The telnet daemon, even with SSL[25] support, should be uninstalled unless it is required. The telnet, even with SSL support, should not be installed. When remote shell is required, up to date SSH[26] daemon can be used.

Implementation:

sudo apt-get remove --auto-remove telnetd-ssl

Value:

It was not installed.

Reference:

https://static.open-scap.org/ssg-guides/ssg-ubuntu1604-guide-anssi_np_nt28_minimal.html

https://www.howtoinstall.co/en/ubuntu/trusty/telnetd-ssl?action=remove

5.3.2.4 ***Uninstall the inet-based telnet server***

The inet-based telnet daemon should be uninstalled unless it is required. The telnet allows clear text communications and does not protect with encryption any data transmission between client and server. Any confidential data can be listened, and no integrity checking is made.

Implementation:

sudo apt-get remove --auto-remove inetutils-telnetd

Value:

It was not installed.

Reference:

https://static.open-scap.org/ssg-guides/ssg-ubuntu1604-guide-anssi_np_nt28_minimal.html

[25] https://it.wikipedia.org/wiki/Transport_Layer_Security
[26] https://en.wikipedia.org/wiki/Secure_Shell

https://www.howtoinstall.co/en/ubuntu/precise/inetutils-telnetd?action=remove

5.4 Privacy

This section describes general privacy-related hardening suggestions.

5.4.1 Disable Apport Error reporting service

Apport is a system that is gathering information about crashes and the OS environment then sends it back to an Ubuntu server. It has several other similar features. It should be disabled, unless it is required.

Implementation:

It is disabled by default.

Stop and disable the service by running:

sudo systemctl stop apport.service

sudo systemctl disable apport.service

sudo systemctl mask apport.service

In addition, the following commands need to be executed as the primary device user (not the administrator):

gsettings set com.ubuntu.update-notifier show-apport-crashes false

ubuntu-report -f send no

Value:

It was running.

Reference:

https://www.ncsc.gov.uk/collection/end-user-device-security/platform-specific-guidance/ubuntu-18-04-lts#Policies

https://wiki.ubuntu.com/Apport

5.4.2 Disable the Whoopsie service

This service is the crash database submission daemon, this is responsible for sending the crash reports back to an Ubuntu server. It should be disabled unless it is required.

Implementation:

Stop and disable the service by running:

sudo systemctl stop whoopsie.service

sudo systemctl disable whoopsie.service

sudo systemctl mask whoopsie.service

Value:

It was running.

Reference:

https://www.ncsc.gov.uk/collection/end-user-device-security/platform-specific-guidance/ubuntu-18-04-lts#Policies

https://askubuntu.com/questions/tagged/whoopsie

5.4.3 **Remove Popularity Contest service**

The popularity contest service gathers information about Debian packages installed on the system and sends every week the list of packages installed and the access time of relevant files to the server via email. It should be disabled, unless it is required.

Implementation:

Uninstall the service by running:

sudo apt-get remove -y popularity-contest

So, users cannot unset them, these settings should be locked using the following steps:

Create a /etc/dconf/profile/userfile containing:

user-db:user

system-db:local

Then run:

dconf update

Value:

It was not removed.

Reference:

https://www.ncsc.gov.uk/collection/end-user-device-security/platform-specific-guidance/ubuntu-18-04-lts#Policies

https://popcon.debian.org/

5.4.4 **Disable Connectivity Checker**

The "Connectivity Checker" sends pings to a Canonical server to check if it's online. This is enabled by default and can be turned off in the privacy section of the settings menu. It should be disabled, unless it is required.

Implementation:

Add the following lines to the /var/lib/NetworkManager/NetworkManager-intern.conf file:

[connectivity]

.set.enabled=false

Then:

sudo systemctl restart NetworkManager.service

sudo find /var -newermt "-1 minute" -ls

Value:

It was enabled.

Reference:

https://www.ncsc.gov.uk/collection/end-user-device-security/platform-specific-guidance/ubuntu-18-04-lts#Policies

https://askubuntu.com/questions/1029108/how-do-i-programmatically-disable-connectivity-checking

5.5 Interfaces

This section describes hardening suggestions related to interfaces.

5.5.1 Disable USB usage

Depending on how critical your system is, sometimes it's necessary to disable the USB sticks usage.

Implementation:

Add the following line into the /etc/modprobe.d/blacklist.conf file:

blacklist usb_storage

Value:

It was enabled.

Reference:

https://www.computerworld.com/article/3144985/linux-hardening-a-15-step-checklist-for-a-secure-linux-server.html

https://itsfoss.com/how-to-disable-usb-ports-in-ubuntu/

5.6 Others

This section describes linux hardening suggestions that did not fit into previous categories.

5.6.1 Account and Access Login

5.6.1.1 *Disable root login*

The root user has superior power and can execute any command in the operating system. If your server is accessible by anyone on the internet, then it is better to disable root login directly from the SSH. Generally, intruders will try to crack or guess root user password first to penetrate your system.

Implementation:

Disable root login in the system by uncommenting "PermitRootLogin" parameter and changing the value of it to "no" in the /etc/ssh/sshd_config file:

PermitRootLogin no

Then restart the SSH service to reflect the changes.

Value:

It was enabled.

Reference:

https://linuxacademy.com/guide/19700-linux-security-and-server-hardening-part1/

5.6.2 SSH

5.6.2.1 *Enable Detailed Logging for SSH*

SSH (Secure Shell) is a cryptographic network protocol used for a secure connection between a client and a server. If you want more detailed logging, then you can use the value DEBUG, DEBUG1, DEBUG2, and DEBUG3 instead of VERBOSE. Logging with a DEBUG level violates the privacy of users and is not recommended.

Implementation

You can enable detailed logging for SSH through /etc/ssh/sshd_config file by changing the value of "LogLevel" parameter to verbose:

LogLevel VERBOSE

Value:

It was disabled.

Reference:

https://linuxacademy.com/guide/19700-linux-security-and-server-hardening-part1/

https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server-in-ubuntu

6 Developments for Use Case Scenarios

Within CPSwarm project, a prototypical Abstraction Library for rovers and drones used in Search&Rescue and Logistic use cases was developed as a proof of concept. Generally, the development of the Abstraction Library for different robots is expected to be carried out by manufacturers or the open source community. For swarm designing, the assumption is made that abstraction layer's implementation for the different CPSs are already available and follow the convention defined by CPSwarm.

As already done for the previous version of this deliverable, the following section will collect the main activities carried out during this second part of the project to integrate new hardware for our scenarios. For a deeper description related to each scenario, dedicated deliverables can be consulted (D8.2 and D8.3).

6.1 Search and Rescue Scenario

With respect to work already presented in D7.1, specific additions to the CPSwarm Abstraction Library have been realized in the SAR scenario to allow the execution of the demonstrator in an indoor environment. These two implementations were necessary because of the impossibility to use the magnetometer and the barometer in some closed area, where both components are affected by possible interferences and cannot guarantee an appropriate level of reliability.

In order to substitute the magnetometer, a new solution was implemented using the video camera looking at a particular carpet filled with many AprilTags[27], as shown in the figure below. Computing his displacement from the orientation of the framed AprilTag the drone could infer its actual yaw.

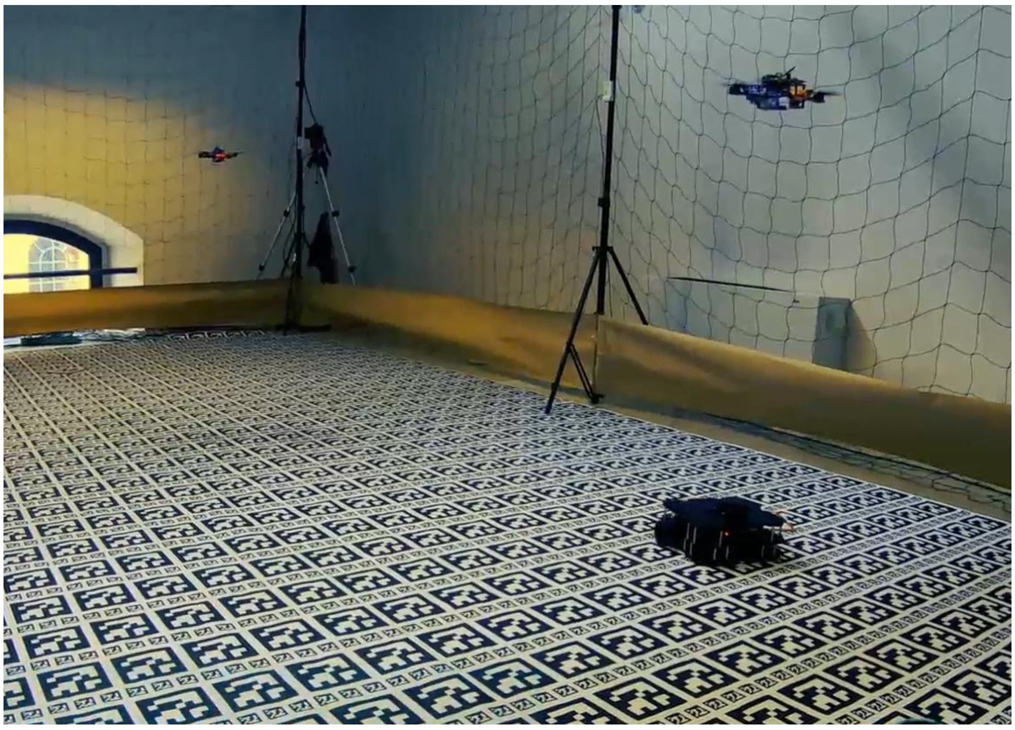

Figure 4 - Drone yaw estimation using AprilTags and video-camera

The camera used for such implementation is the OpenMV Cam H7[28] and was integrated into the system by updating the related ROS component (see Table 2) that was already in charge to detect single targets simulating the presence of a casualty to be rescued.

[27] https://april.eecs.umich.edu/software/apriltag.html

[28] https://openmv.io/products/openmv-cam-h7

For the substitution of the barometer that was used to compute the actual altitude position of the UAV, a small lidar[29] component (see Figure 5) has been added on board of the drone. Also, in this case, the new hardware component has been integrated into the system implementing a dedicated ROS package.

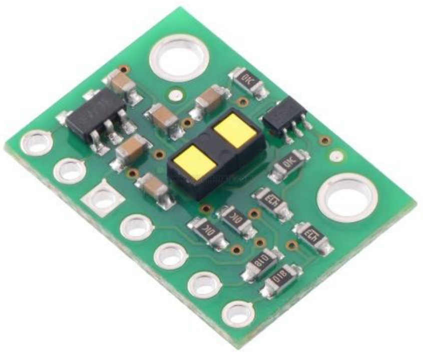

Figure 5 - VL53L1X Long distance ranging Time-of-Flight sensor

6.2 Logistic Scenario

This section introduces the new implementation related to the Logistic scenario.

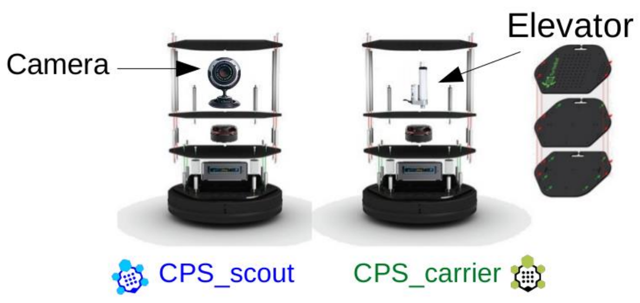

Figure 6 - Robotnik's Turtlebots scout and carrier customization

In this scenario, two types of robots are going to be considered: the CPS_scout robot and the CPS_carrier robot. In Figure 6 the main differences among the two customizations are shown.

The CPS_carrier robots are the ones that can lift carts, thus they will keep the elevator installed between the two last wood stacks of the top of the robot.

The CPS_scout robots do not need to lift carts, so the elevator has been removed. As innovation in the new scenario they have to identify QR codes from the carts thus a camera needs to be integrated. The camera model chosen is FLIR Chameleon CM3 (shown at Figure 6). It has a resolution of 2048x1536 pixels, a frame rate

Figure 7 - FLIR Chameleon CM3 camera

[29] https://www.st.com/en/imaging-and-photonics-solutions/vl53l1x.html

of 55 fps and an USB 3.0 interface. The integration to the Ubuntu Operating System was necessary to increase the bandwidth of the USB ports in order to get the images at the rate of 55 fps.

In addition to the above, the laser sensors for both types of robots have been changed due to the problems experimented during the preparation of the demonstration at M18 with the laser RPLidar A2. The problem with this laser basically that the natural sunlight from the windows of the hangar created invisible objects for the laser that were not really there. Due to this, we decided to change the sensor and integrate the laser Hokuyo UST-10LX in all robots. This laser has a range of 10m, a wide detection angle of 270°, an angular resolution of 0.25° and a precision of 40mm. Like the previous laser, this model is oriented to indoors, but it supports better the light of the sun according to our professional experience. The interface of the previous laser was USB but for the case of Hokuyo, it is connected to the computer by Ethernet and it needs to be fed with 10 to 30 V. Due to this some electrical modifications have been made to feed the laser directly from the battery of the robot.

Figure 8 - Hokuyo UST-10LX lidar

From the software point of view, the package used to integrate the new laser scan was the urg_node[30] and for the chameleon the package was the flir_camera_driver[31]. As these two software components were already provided by the ROS community, they were not included inside the final summary table in section 6.3.

Another new hardware apart from the robots and carts has been integrated in the logistic scenario for safety reasons. In the scenario there will be a fixed camera on a tripod checking all the time that there is no person close to the scenario. For this mission the camera Rubedos VIPER[32] has been selected. This camera is a stereo camera with a depth range from 1 m to over 30m and a diagonal field of view up to 92 degrees. It also incorporates an NVIDIA® Jetson™ TX2 Module to perform on-board image processing. This device will be part of the scenario as another CPS and will notify the robots if it detects humans. The integration of the device inside the scenario is highly based on ROS and the communication with the other CPSs will be managed by the integrated Communication Library.

Finally, the control box, used in the scenario to send specific start/stop/abort messages to the swarm, has also been updated. Now it includes the following list of devices:

- AC-DC converters and wiring: small converters will be powering the devices with 12V and 24V DC.
- Schneider ZBRRC and ZBRT buttons: this module is receiving wireless signals from no-battery pushbuttons, resulting in a clean and easy infrastructure deployment.

Figure 9 - Schneider ZBRRC and ZBRT buttons

[30] http://wiki.ros.org/urg_node
[31] https://github.com/ros-drivers/flir_camera_driver
[32] https://www.rubedos.com/solutions/viper

- Advantech ADAM 6060 I/O module: ADAM-6000 accomplishes the integration of automation and enterprise systems easily through internet technology, so that users can avoid changing the entire architecture of the control system and even remotely monitor the device status more flexibly.

  The module main characteristics are:

  - 6-ch DI, 6-ch RL, Ethernet-based smart I/O
  - Remote monitoring and control with mobile devices
  - Group configuration capability for multiple module setup
  - Flexible user-defined Modbus address
  - Intelligent control ability by Peer-to-Peer and GCL function
  - Active I/O message by data stream or event trigger function
  - Multiple protocol support: Modbus/TCP, TCP/IP, UDP, HTTP, DHCP, SNMP, MQTT
  - ASUS RT-N12 Router 300Mbps 2.4GHz

Figure 10 - ADAM-6060 module

Using the same approach applied for the monitoring fixed camera, the communication capability with the other software entities that were part of the swarm has been enabled installing ROS and integrating the CPSwarm Communication Library on board of the control box.

6.3 ROS Abstraction Library components summary

In the table below we present a list of the developed components divided per layer.

Layer	Name	Description	Responsible	License
Hardware Drivers	obc_ros_lidar	ROS package that provide access to VL53L1X lidar	DIGISKY	Apache 2.0
Hardware Drivers	obc_ros_openmv	ROS package that provide access to OpenMV H7 camera[33]	DIGISKY	Apache 2.0
Hardware Drivers	obc_ros_sonar	ROS package that provide access to MaxBotix MB1242-EZ4 sonar[34]	DIGISKY	Apache 2.0

[33] https://openmv.io/products/openmv-cam-h7
[34] https://www.maxbotix.com/Ultrasonic_Sensors/MB1242.htm

Hardware Drivers	obc_ros_uwb	ROS package that provide access to localization data computed by Decawave UWB module[35]	LINKS	Proprietary License
Sensing and Actuation	area_provider	ROS Services to access the area of the mission in local coordinates	LINKS/LAKE	Apache 2.0
Sensing and Actuation	battery_monitor	ROS module to check the condition of the battery and advise when the percentage is under a pre-defined threshold	LINKS	Apache 2.0
Sensing and Actuation	mavros_gps	A ROS package that provides common operations required when working with GPS coordinates based on MAVROS stack[36]	LINKS/LAKE	Apache 2.0
Sensing and Actuation	mavros_pos_controller	A position controller for CPSs based on the ROS MAVROS stack	LINKS/LINKS	Apache 2.0
Sensing and Actuation	mavros_pos_provider	ROS package to provide the local position of the CPS received from MAVROS	LINKS/LAKE	Apache 2.0
Sensing and Actuation	mavros_vel_controller	ROS package to control the CPS's velocity from MAVROS stack	LINKS/LAKE	Apache 2.0
Sensing and Actuation	mavros_vel_provider	ROS package to provide the velocity received from MAVROS locally	LINKS/LAKE	Apache 2.0
Sensing and Actuation	navigation_pos_controller	A position controller for UGVs based on the ROS navigation stack[37] using the move_base[38] package	LINKS/LAKE	Apache 2.0
Sensing and Actuation	navigation_pos_provider	ROS package to provide the pose received from the ROS navigation stack.	LINKS/LAKE	Apache 2.0
Sensing and Actuation	navigation_vel_provider	ROS package to provide the velocity computed from the ROS navigation stack positions.	LINKS/LAKE	Apache 2.0

[35] https://www.decawave.com/product/dwm1001-development-board
[36] http://wiki.ros.org/mavros
[37] http://wiki.ros.org/navigation
[38] http://wiki.ros.org/move_base

Sensing and Actuation	obstacle_detection	ROS Services to detect obstacles using range sensors and communication	LINKS/LAKE	Apache 2.0
Hardware Functions	mavros_moveto	Function to send a CPS to a specific location based on MAVROS stack (using GPS or local positioning system)	LINKS	Apache 2.0
Hardware Functions	mavros_moveto_target	Function to send a CPS to a specific "target to reach" location based on MAVROS stack (using GPS or local positioning system), listening for possible position updates	LINKS	Apache 2.0
Hardware Functions	mission_aborter	ROS module to force mission abort when some critical condition is identified (e.g. low battery, hardware problems)	LINKS	Apache 2.0
Hardware Functions	moveto	A ROS package that provides an action server to move a CPS to a given position based on standard move_base_msgs[39]	LINKS/LAKE	Apache 2.0
Hardware Functions	uav_mavros_land	ROS module to execute UAV landing based on MAVROS stack	LINKS	Apache 2.0
Hardware Functions	uav_mavros_takeoff	ROS module to execute UAV autonomous takeoff based on MAVROS stack	LINKS	Apache 2.0
Hardware Functions	ugv_elevator	ROS action server to control ROBOTNIK Turtlebot's elevator	ROBOTNIK	Apache 2.0
Hardware Functions	ugv_picknplace	ROS action server to move a CPS to a given position, pick a cart, move to another given position and place it	ROBOTNIK	Apache 2.0

Table 2 - List of Abstraction Library components developed during the project

[39] http://wiki.ros.org/move_base_msgs

7 Conclusions

This deliverable has presented the work done in Task 7.1 to design and implement the final version of the CPSwarm Abstraction Library. As the overall structure of the library was already presented in the previous iteration of this document, a particular attention has been drawn to the presentation of the additional concepts introduced in the second half of the project.

Acronyms

Acronym	Explanation
API	Application Programming Interface
CPS	Cyber Physical System
ROS	Robot Operating System
AADL	Architecture Analysis and Design Language
GPS	Global Positioning System
FSM	Finite State Machine
HFSM	Hierarchical Finite State Machines
REST	Representational State Transfer
WADL	Web Application Description Language
WSDL	Web Services Description Language
JSON	JavaScript Object Notation
UAV	Unmanned aerial vehicle
UGV	Unmanned ground vehicle
SSH	Secure Shell
SSL	Secure Sockets Layer
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

List of figures

List of tables

References

[1] Koubaa, Anis. (2015). ROS As a Service: Web Services for Robot Operating System. Journal of Software Engineering for Robotics. 1. 1.

[2] Keppmann, Felix & Maleshkova, Maria & Harth, Andreas. (2015). Building REST APIs for the robot operating system-mapping concepts and interaction. CEUR Workshop Proceedings. 1359. 10-19.

[3] Bardaro, Gianluca & Semprebon, Andrea & Matteucci, Matteo. (2017). AADL for robotics: a general approach for system architecture modeling and code generation. Journal of Software Engineering for Robotics. 8. 32-44.

ANNEX A – CPSwarm Abstraction Description File (Json schema)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["runtime-env"],
  "properties": {
    "runtime-env": {
      "const": "ROS"
    },
    "sensors/actuators": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/hw_component"
      }
    },
    "functions": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/function"
      }
    }
  },
  "additionalProperties": false,
  "definitions": {
    "msg_field": {
      "type": "object",
      "required": ["class", "name"],
      "properties": {
        "class": {"type": "string"},
        "name": {"type": "string"},
        "description": {"type": "string"}
      },
      "additionalProperties": false
    },
    "constant_field": {
```

```
    "type": "object",
    "required": ["class", "name", "value"],
    "properties": {
      "class": {"type": "string"},
      "name": {"type": "string"},
      "value": {"type": ["number", "string", "boolean", "array"]},
      "description": {"type": "string"}
    },
    "additionalProperties": false
  },
  "message": {
    "type": "object",
    "required": ["topic", "msg"],
    "properties": {
      "topic": {"type": "string"},
      "msg": {
        "type": "object",
        "required": ["class"],
        "properties": {
          "class": {"type": "string"},
          "constants": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/constant_field"
            }
          },
          "fields": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/msg_field"
            }
          }
        },
        "additionalProperties": false
      }
  },
  "additionalProperties": false
 },
 "service": {
  "type": "object",
  "required": ["name","class"],
  "properties": {
   "name": {"type": "string"},
   "class": {"type": "string"},
   "request": {
    "type": "object",
    "required": ["fields"],
    "properties": {
     "constants": {
      "type": "array",
      "items": {
       "$ref": "#/definitions/constant_field"
      }
     },
     "fields": {
      "type": "array",
      "items": {
       "$ref": "#/definitions/msg_field"
      }
     }
    },
    "additionalProperties": false
   },
   "response": {
    "type": "object",
    "required": ["fields"],
    "properties": {
     "constants": {
      "type": "array",
      "items": {
              "$ref": "#/definitions/constant_field"
            }
          },
          "fields": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/msg_field"
            }
          }
        },
        "additionalProperties": false
      }
    },
    "additionalProperties": false
  },
  "action": {
    "required": ["name","class"],
    "properties": {
      "name": {"type": "string"},
      "class": {"type": "string"},
      "goal": {
        "type": "object",
        "required": ["fields"],
        "properties": {
          "fields": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/msg_field"
            }
          }
        }
      },
      "result": {
        "type": "object",
        "required": ["fields"],
        "properties": {
          "fields": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/msg_field"
            }
          }
        }
      },
      "feedback": {
        "type": "object",
        "required": ["fields"],
        "properties": {
          "fields": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/msg_field"
            }
          }
        }
      }
    },
    "additionalProperties": false
  },
  "hw_component": {
    "type": "object",
    "required": ["name", "description", "category", "api"],
    "properties": {
      "name": {"type": "string"},
      "description": {"type": "string"},
      "category": {
        "type": "string",
        "enum": ["Sensor", "Actuator", "Virtual"]
      },
      "param_list": {
        "type": "array",
       "items": {
        "$ref": "#/definitions/constant_field"
       }
      },
      "api": {
       "type": "object",
       "properties": {
        "inputs": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/message"
         }
        },
        "outputs": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/message"
         }
        },
        "services": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/service"
         }
        }
       }
      }
     },
     "additionalProperties": false
    },
    "comm_model": {
     "type": "object",
     "required": ["paradigm", "definition"],
     "properties": {
```

```
    "paradigm": {
      "type": "string",
      "enum": ["rosaction", "rosservice"]
    },
    "definition": {
      "type": "object"
    }
  },
  "allOf": [
    {
      "if": {
        "properties": {"comm_paradigm": {"const": "rosaction"}}
      },
      "then": {
        "properties": {
          "comm_model": {
            "$ref": "#/definitions/action"
          }
        }
      }
    },
    {
      "if": {
        "properties": {"comm_paradigm": {"const": "rosservice"}}
      },
      "then": {
        "properties": {
          "comm_model": {
            "$ref": "#/definitions/service"
          }
        }
      }
    }
  ],
  "additionalProperties": false
```

```
},
"function": {
  "type": "object",
  "required": ["name", "description", "category", "api"],
  "properties": {
    "name": {"type": "string"},
    "description": {"type": "string"},
    "category": {
      "type": "string",
      "enum": ["abstraction-lib", "swarm-lib"]
    },
    "param_list": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/constant_field"
      }
    },
    "api": {
      "type": "object",
      "required": ["comm_model"],
      "properties": {
        "inputs": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/message"
          }
        },
        "outputs": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/message"
          }
        },
        "comm_model": {
          "$ref": "#/definitions/comm_model"
```

```
            }
          }
        },
        "additionalProperties": false
      }
    }
  }
}
```

ANNEX B – UAV Abstraction Description File example

```
{
  "runtime-env": "ROS",
  "sensors/actuators": [
    {
      "name": "GPS",
      "description": "Sensor to provide GPS data",
      "category": "Sensor",
      "api": {
        "outputs": [
          {
            "topic": "mavros/global_position/global",
            "msg": {
              "class": "sensor_msgs/NavSatFix",
              "fields": [
                {
                  "class": "float64",
                  "name": "latitude"
                },
                {
                  "class": "float64",
                  "name": "longitude"
                },
                {
                  "class": "float64",
                  "name": "altitude"
                }
              ]
            }
          }
        ]
      }
    }
  ],
  "functions": [
{
  "name": "uav_mavros_takeoff",
  "description": "Send takeoff command",
  "category": "abstraction-lib",
  "param_list": [
    {
      "class":"number",
      "name": "pos_tolerance",
      "value": 0.1
    },
    {
      "class":"number",
      "name": "frequency",
      "value": 10.0
    },
    {
      "class":"number",
      "name": "stabilize_time",
      "value": 5
    },
    {
      "class":"number",
      "name": "takeoff_steps",
      "value": 1
    },
    {
      "class":"number",
      "name": "initial_yaw",
      "value": 90
    }
  ],
  "api": {
    "inputs": [
      {
        "topic": "mavros/state",
      "msg": {
        "class": "mavros_msgs/State",
        "fields": [
          {
            "class":"stds_msgs/Header",
            "name": "header",
            "description": "ros header"
          },
          {
            "class":"bool",
            "name": "connected"
          },
          {
            "class":"bool",
            "name": "armed"
          },
          {
            "class":"bool",
            "name": "guided"
          },
          {
            "class":"bool",
            "name": "manual_input"
          },
          {
            "class":"string",
            "name": "mode"
          },
          {
            "class":"uint8",
            "name": "system_status"
          }
        ]
      }
    },
    {
      "topic": "pos_provider",
      "msg": {
        "class": "geometry_msgs/PoseStamped",
        "fields": [
          {
            "class":"stds_msgs/Header",
            "name": "header",
            "description": "ros header"
          },
          {
            "class":"geometry_msgs/Pose",
            "name": "pose"
          }
        ]
      }
    }
  ],
  "outputs": [
    {
      "topic": "pos_controller/goal_position",
      "msg": {
        "class": "geometry_msgs/PoseStamped",
        "fields": [
          {
            "class":"stds_msgs/Header",
            "name": "header",
            "description": "ros header"
          },
          {
            "class":"geometry_msgs/Pose",
            "name": "pose"
          }
        ]
      }
```

```
        }
      ],
      "comm_model": {
        "paradigm": "rosaction",
        "definition": {
          "name": "cmd/takeoff",
          "class": "uav_mavros_takeoff/TakeOff",
          "goal": {
            "fields": [
              {
                "class":"float64",
                "name": "altitude"
              }
            ]
          }
        },
      }
    }
  },
  {
    "name": "uav_mavros_land",
    "description": "Send land command",
    "category": "abstraction-lib",
    "api": {
      "comm_model": {
        "paradigm": "rosservice",
        "definition": {
          "name": "cmd/land",
          "class": "std_srvs/Empty"
        }
      }
    }
  },
  {
    "name": "auction_action",
    "description": "Assign a task in a specific position to another CPS",
    "category": "swarm-lib",
    "api": {
      "inputs": [
        {
          "topic": "bridge/events/cps_selection",
          "msg": {
            "class": "cpswarm_msgs/TaskAllocationEvent",
            "fields": [
              {
                "class":"stds_msgs/Header",
                "name": "header",
                "description": "ros header"
              },
              {
                "class":"swarmros/EventHeader",
                "name": "swarmio",
                "description": "cpswarm swarmio swarmros header"
              },
              {
                "class":"int32",
                "name": "task_id",
                "description": "id of the task"
              },
              {
                "class":"float64",
                "name": "bid",
                "description": "bid of the cps for the task (inverse of cost)"
              }
            ]
          }
        }
      ],
      "outputs": [
        {
          "topic": "cps_selected",
          "msg": {
            "class": "cpswarm_msgs/TaskAllocatedEvent",
            "fields": [
              {
                "class":"stds_msgs/Header",
                "name": "header",
                "description": "ros header"
              },
              {
                "class":"swarmros/EventHeader",
                "name": "swarmio",
                "description": "cpswarm swarmio swarmros header"
              },
              {
                "class":"int32",
                "name": "task_id",
                "description": "id of the task"
              },
              {
                "class":"string",
                "name": "cps_id",
                "description": "uuid of the cps to which the task has been allocated"
              }
            ]
          }
        }
      ],
      "comm_model": {
        "paradigm": "rosaction",
        "definition": {
          "name": "cmd/task_allocation_auction",
          "class": "cpswarm_msgs/TaskAllocation",
          "goal": {
            "fields": [
              {
                "class":"string",
                "name": "auctioneer",
                "description": "UUID of the CPS performing the task allocation"
              },
              {
                "class":"uint32",
                "name": "task_id",
                "description": "ID of the task"
              },
              {
                "class":"geometry_msgs/PoseStamped",
                "name": "task_pose",
                "description": "Local position of the task"
              }
            ]
          },
          "result": {
            "fields": [
              {
                "class":"string",
                "name": "winner",
                "description": "UUID of the CPS to which the task is allocated"
              },
              {
                "class":"uint32",
                "name": "task_id",
                "description": "ID of the task"
              },
              {
                "class":"geometry_msgs/PoseStamped",
                "name": "task_pose",
                "description": "Local position of the task"
              }
            ]
```

```
                }
              },
            }
          }
        }
  ]
}
```