WARA-PS Core System Infrastructure

Written by: Caroline Flodin, Combitech

04/04-2022

Version: 1.3

Purpose

The purpose of this document is to provide an introduction and basic understanding of the WARA-PS Core System and its components as well as the basic architecture of how they are connected and interact with each other. This in order to ease the integration of new agents and services into the system, to allow for conducting own experiments and tests. With the included examples, a developer will be able to start exploring collaborating autonomous agents in WARA-PS.

Target group

This document is intended for developers new to the WARA-PS Core System and for those who want to understand how the components are connected and works on a basic level.

Section Overview

1. ***Introduction –*** In this chapter a brief description is given of the goals and principles of the Core System.
2. ***WARA-PS Core System Overview –*** This section provides a short description of what is meant by "Core System" in WARA-PS and displays an image of the main concept as well as another image of the different agents, systems, subsystems, services etc. that make up the Core System.
3. ***Logical System Architecture –*** This chapter presents how the Core System components are connected on a logical level as well as provides a basic description of them. How agents are divided into different levels according to their capabilities is also described in this chapter along with an overview of what each level means.
4. ***WARA-PS Core System Information Flows –*** This chapter provides an overview and a flow chart of how information flows through the Core System to or from agents of different levels.
5. ***Integrating a new agent –*** Here, a short description is given of the three steps we recommend one follows when wishing to integrate a new agent into the Core System.
6. ***Abbreviations and Definitions –*** Lastly, abbreviations used in this document is described along with definitions of words used such as "system", "agent" etc.

References

[1] "WARA-PS: a research arena for public safety demonstrations and autonomous collaborative rescue robotics experimentation", https://rdcu.be/cBtFo

[2] API-Spec, API - Filer - Media and Logs @ WARAPS

[3] LiU TST description (also contain information about ROS2-agents), lrs2 / lrs_doc · GitLab (liu.se)
For questions about transforming ROS2 agents into WARA-PS ROS2-agents, contact Tommy Persson (tommy.persson@liu.se)

Table of Contents

1. Introduction

The main goal for the WARA-PS Core System is to boost research on collaborating autonomous agents, starting from a higher level. This means that the Core System Infrastructure shall enable different agents, systems and services to be integrated and shared in a common overall system. It shall also be easy to take advantage and contribute to the system. More details of WARA-PS research-background can be found in ref [1].

The WARA-PS Core System is based on the following principles:

Domain agnostic

- Uniform ontology as far as possible, independent of domain (air, sea, ground, space, cyber)
- Defined API for systems-systems, systems-subsystems, systems-agents and agents-to-agents

Easy integration

- Agents can be incrementally integrated at different levels with MQTT or ROS
- Adaption to developer needs in order to connect more agents and services

Available 24/7

- Core System is constantly in service for e.g. integration tests
- Virtual copies of agents exist to allow some tests without needing the physical agent
- Services can be configured and run locally if needed (Testing, Private Network, ...)

Transparent and accessible

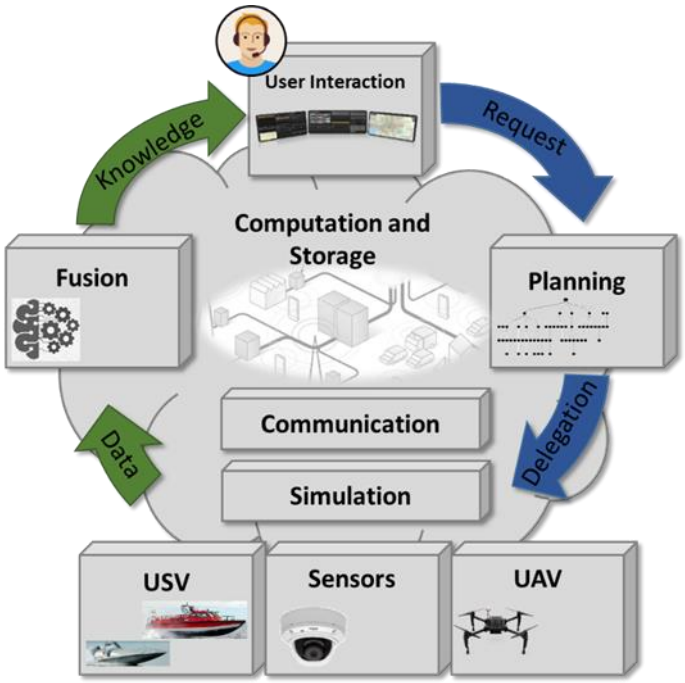
- Graphical Test User Interface on webpage for experiment interaction
- Positioning, sensors, video, commands etc. is accessible through the WARA-PS Integration Test Tool for development and debugging purposes

Capability for recording and analysis

- Recording functions for temporal events are present, e.g. MongoDB and Maria DB, for further analysis
- Repository for storing media, GPS-tracks and other data is available.

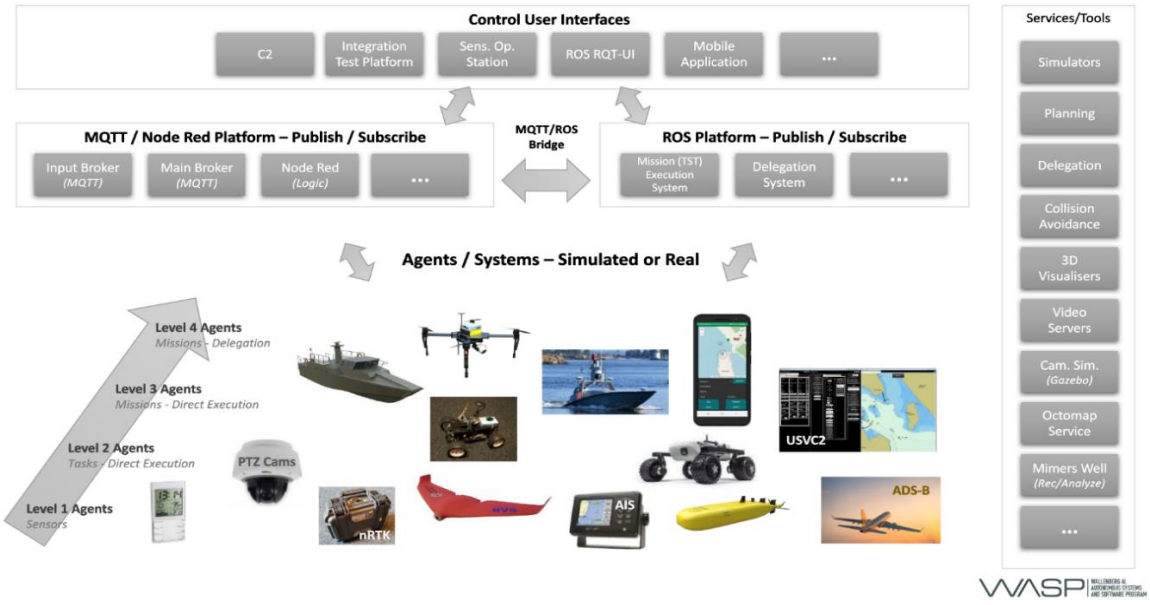
2. WARA-PS Core System Overview

The WARA-PS Core System is a composition of many connected Systems of Systems acting together to achieve a higher capability than any of the systems can do alone.



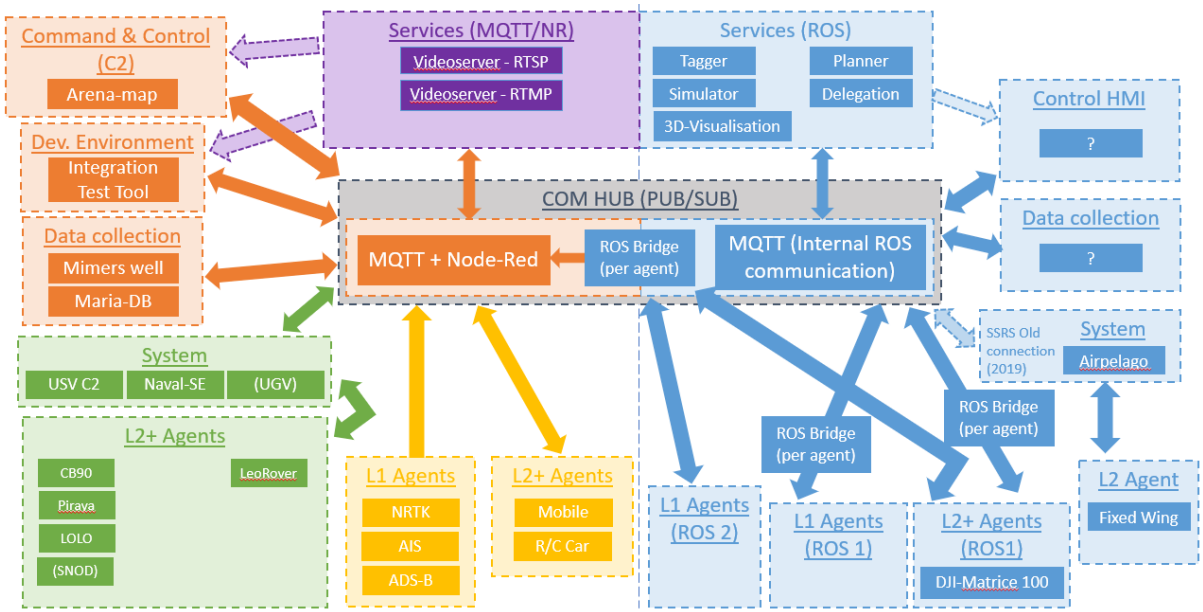
The above image shows the main concept of the WARA-PS Core System. From the user requesting something, it being planned and delegated to agents (simulated or real) who then sends data that is fused together with other data to provide the user with knowledge.

You can think of it more as a reference system, containing services, defined agents and examples of implementations, all to provide a good research environment. For that purpose, the system is also created for easy integration of new agents, whether they are real or virtual, and to stimulate development of own services to replace the example ones.



The above image provides an overview of example systems, subsystems, types of agents, communication platforms as well as services that make up the WARA-PS Core System.

3. Logical System Architecture



The above image represents the logical connections of the 7 different main parts of the Core System architecture, which will be further described in this chapter.

3.1 COM HUB (PUB/SUB)

A central part of the Core System architecture is the “COM HUB”. It is the central node for communication between the user (through the Control HMI), the agents (autonomous vehicles or human agents), the system, services and the data collection systems.

The main parts of the COM HUB are the MQTT + Node-Red, seen in the image to the left, the ROS Bridges (one bridge per agent) and the MQTT used for internal ROS communication, to the right. To communicate, the MQTT’s use a technique called “publish/subscribe”, or “PUB/SUB”* for short. A distinction is also made here about how the Core System contains two alternative environments, for communication and how to integrate agents, systems and different services. Which one to choose when implementing a new agent depends on the user’s legacy.

Some parts of the Core System use the main MQTT broker to communicate, with Node-Red providing the logic for how the messages are to be received and sent. Other parts are ROS dependent and thus uses ROS Bridges and the internal MQTT broker to communicate with each other, and even more ROS bridges to communicate with the main MQTT broker. Note here the plural “bridges”. There is no “central” ROS Bridge for e.g. communicating with the internal MQTT broker or the main MQTT broker, each agent uses its own ROS Bridge to communicate, with different bridges being used depending on whether it is for the internal or the main MQTT. Meaning, each ROS agent has multiple ROS bridges.

These bridges are necessary for translating between the ROS environment and the MQTT brokers and thus for the exchange of data needed to perform certain tasks and functions that cannot be done solely in one environment. This means that while some tasks and functions can be performed by only using one of the two environments, such as the level 2 task move-to, others need to use both at the same time to work. One example of this is how agents of level 3 and level 4 always need to use ROS functions to work, regardless if they are on the ROS side or the MQTT-Node-Red side.

**The PUB/SUB technique works by having different agents/actors/systems publishing a message (information) to a specified topic, managed by a broker, that other agents/actors/systems are subscribing to, thus receiving the message (information) sent from the first part. In the case of the publisher being a sensor, this message would be sensor data while in the case of the publisher being a user, through the Control HMI, the message could be a task for an agent.*

3.2 Command & Control (C2)

The "Command & Control (C2)", seen in the top corners of the image, does not represent any environment or system but is the collective name of user interfaces intended for decision makers and leaders such as rescue leaders. To not overwhelm these users with too much information, these interfaces filter away, what for the user is, unnecessary meta data from the agents and the main capability is to send commands for tasks and missions such as 'search this area'.

The C2 interface available currently is the Arena-map in which agents can be observed on the map and clicked on to see some of their meta-data. To send commands, currently the Integration Test Tool is used to perform some of the capabilities that later will exist in the Arena-map.

3.3 Dev. Environment

The "Dev. Environment", short for Development Environment, is the collective name for developer interfaces between the user and the rest of the Core System, such as the Integration Test Tool which is used the most currently. It is in these interfaces that all the information and meta-data about the different agents is displayed that would not be relevant for e.g. a rescue leader. Commands can also be issued from this interface but that is primarily for tests, experiments and debugging. Meaning they are created for aiding in developers in their research rather than for actual missions.

3.4 Data collection

Most of the data collected from the agents and passing through the core system are stored in various data collection systems such as Mongo DB and Maria DB. There also exist a data collection system on the ROS side that is used to save data from tests.

Of these, Maria DB records all topics that passes through the system, meaning all data except video and images.

3.5 Services

A "Service" in the WARA-PS Core System is defined as a block of code that can be performed by an agent and other services. What this means is that every block of code that can make an agent do something, whether that is simply sending information, moving to a specified point or even search an area, is a kind of service.

Another type of services is complementary services that can aid the agents in performing their tasks. These complementary services, such as a route planner service, do not necessarily need to be used if an alternative to the necessary code, e.g. for planning the route, already exist in the agents own code. On the other hand, if one wants to integrate many agents or does not have an interest of writing one's own functions, e.g. for planning, themselves, it is easier to simply use the

complementary service and not reinventing the wheel. This is also the case for complementary services that perhaps you only want to use once e.g. for research. By using complementary services in those circumstances, one also does not have to rewrite the code for each new agent that one wants to integrate, saving time.

3.6 Agents

3.6.1 General info

The term “Agents” in the context of the Core System refers to actors that receive and perform tasks/ missions from the Core System and/or provide the Core System with information. These actors can take the shape of sensors, (semi-)autonomous vehicles such as drones, boats and ground vehicles but also people carrying mobile phones.

In the WARA-PS project agents are defined in four levels, depending on their capability of acting autonomous (seen in the image below along with examples of agents used in WARA-PS).

3.6.2 L1 Agents – Sensors

Agents at level 1 are only capable of publishing data that they collect/register rather than able to perform a requested action and are thus usually in the form of sensors.

3.6.3 L2 Agents – Tasks - Direct Execution

Level 2 agents are capable of both publishing data as well as receive (/subscribe to) defined tasks specifically directed at them. In other words, agents at this level are able send information gathered from sensors on the agent in question as well as able to follow commands such as “fly to this point” and “scan this area”. Agents at this level are also able to plan their path depending on the task given, so they have some degree of autonomy.

3.6.4 L3 Agents – Missions - Direct Execution

From the perspective of functions, there is no great difference between a level 3 or level 2 agent. What is different is that at level 3, the agents are combined with a cloud service that extracts the content of a Task Specification Tree (TST), which fully specifies; what the mission goal is, which tasks need to be completed, which agent is to complete which task etc. Meaning that the level 3 agents are not any more autonomous than they were at level 2, but at this level a “subsystem” is utilized to make it possible for the agents to cooperate with each other in a larger mission. More specifically, the subsystem receives a (defined) mission from the user, containing which agents are to perform what tasks as well as a mission goal. It then sends the tasks to the defined agents as well as monitor them to get a status of the mission and to determine if/ when a mission is a success/ failure.

In summary, what makes a level 2 agent into a level 3 agent is not a load of new capabilities, as when comparing level 1 and level 2, but rather the ability to cooperate and having their tasks be a part of a greater mission.

3.6.5 L4 Agents – Missions - Delegation

Level 4 agents, like level 3, are really level 2 agents combined with a cloud service that extracts the content of a Task Specification Tree (TST). The difference between level 3 and level 4 is that the Task Specification Tree at level 4 is not fully specified, in other words undefined. What this means is that, at level 3, the mission had already been divided into tasks required to achieve the goal and it was already decided which agent will perform what action. At level 4, e.g. a mission goal is given but it is not specified which tasks to perform to reach the goal or which agents to use. This deciding and planning process falls on the subsystem to complete so the mission can be carried out successfully.

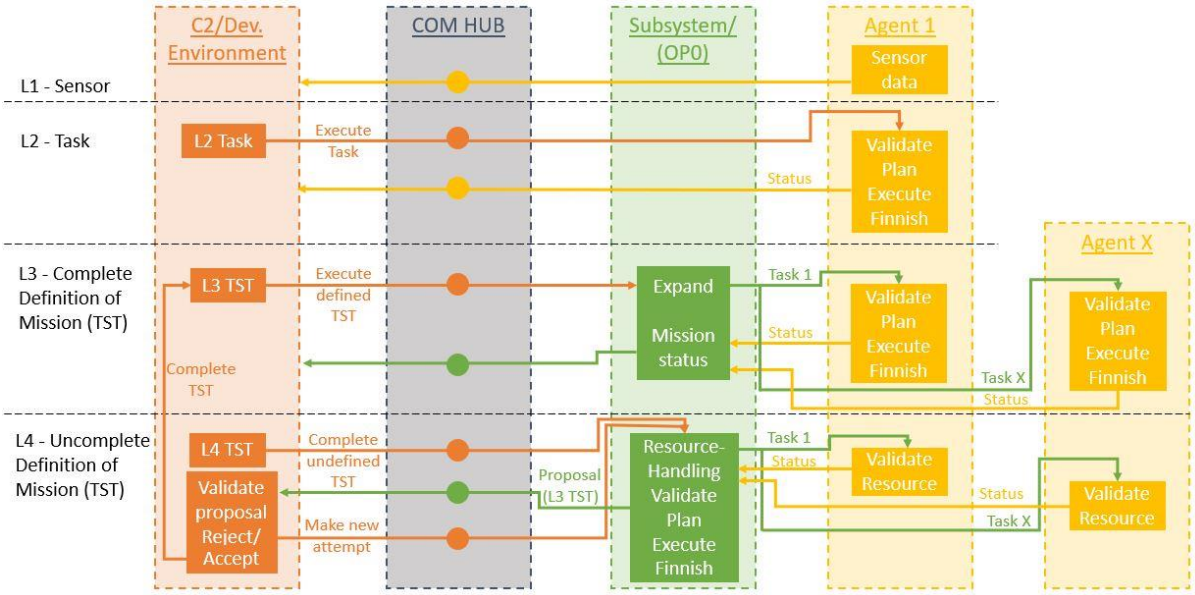
Level 4 agents implement a resource API that enables the cloud service to ask for availability, type of vehicle and other properties that are valid for a specific mission. Thus, making it possible to determine which agents are able, and available, to perform which tasks when planning the mission.

3.7 System with their own agents

A system (with their own agents), as seen in the image above, acts as a kind of intermediary agent by publishing and subscribing to messages through the COM HUB which it receives respectively passes on to the agents it governs. The system also plays a part in planning and delegating missions, more specifically undefined missions. As stated in *3.3.1.4 L4 Agents – Delegation*, undefined missions are missions that lack a defined Task Specification Tree, e.g. it is not specified which tasks are to be performed or which agent is to perform which task.

What this means is that when an undefined mission is sent to a system with its own agents, it is the system who first receives it, not the agents, and acts as the decision maker and planner to fill in the missing mission parameters. In other words, it specifies what needs to be done to complete the mission goal and verifies which agents under it are capable, and available, to do what. The system then uses that information to plan the mission and delegate direct tasks to the agents in question. It then can be likened with a spider in the web, overseeing the progress of each agent and its assigned task and hence the progress of the mission.

4. WARA-PS Core System Information Flows



This image represents information/ task/ mission flows between the C2/Dev. Environments and the agents depending on the level of autonomy. OBS! This image is greatly simplified.

4.1 L1 – Sensor Flows

As stated in the earlier chapter about agents in the Core System architecture, level 1 agents are usually in the form of sensors which publish data they collect. At the top of the image above, we can see how the information would flow through the Core System, starting with the agent publishing sensor data, under a topic, to the COM HUB. The C2/Dev. Environment subscribing to that topic then receives the information from the COM HUB and can make use of it.

4.2 L2 – Task Flows

At level 2, we start to talk about agents not only sending (/publishing) data but receiving (/subscribing to) data in the form of tasks as well. As the flow of sending information has already been described in the previous section, this section will focus on the flow in regards to tasks.

As visualized in the image above, the flow begins with a user sending (/publishing) a task through the C2/Dev. Environment to the COM HUB. This task is then received by the agent subscribing to the topic it was published under in the COM HUB and begins the cycle of going through the task definition. This includes steps such as validating, planning, executing and finally finishing the task. Throughout this whole cycle, the status is continually published to the COM HUB and thus received by the C2/Dev. Environment.

4.3 L3 – Complete Mission & Task Flows

At level 3 we go beyond tasks and start speaking of missions (see definition list in *6. Abbreviations and Definitions*), more specifically missions with a complete definition, meaning that they have a defined Task Specification Tree (TST) (See image of example TST below). At this level, the C2/Dev. Environment does not send each task to each specific agent but rather sends the defined TST through the COM HUB to the Subsystem that then expands the mission. *OBS! The expanding process includes many more parts than can be seen in the image above but for simplicity's sake we do not delve deeper into that in this description.*

When the mission is expanded the system sends each task to each respective agent, which then proceeds to perform the same processes as when receiving a task in level 2. The difference here is that they send their status to the subsystem, to keep it updated on how the mission is proceeding, rather than directly to the C2/Dev. Environment through the COM HUB. Instead, it is the subsystem that regularly sends the status of the mission to the C2/Dev. Environment through the COM HUB.

11.1.5 move-to


```
{
    "com-uuid": "e680c1f5-af53-43f2-9d73-bb4cf74d21c4",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "f2d8abee-5080-4844-a710-8fd4d136e765"
        },
        "name": "move-to",
        "params": {
            "speed": "standard",
            "waypoint": {
                "altitude": 40.0,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            }
        }
    }
}
```


The above image shows an example of the start-tst command for the Move-to function (taken from the WARA-PS API pp. 25-26).

4.4 L4 – Delegation Mission & Task Flows

Level 4 missions are missions whose definition is incomplete, meaning that they lack a completely defined Task Specification Tree (TST). As such, when the level 4 mission is published to the COM HUB, the subsystem subscribing to it need to complete the definition before tasks can be sent to the different agents. This includes steps such as resource handling, validation, planning and execution before it can be finished into a defined mission, meaning a level 3 mission. When the mission has been defined with which agent is to do what, the execution flow looks the same as the flow for level 3. However, before executing the TST it needs to be approved by the user so the completed TST is sent as a proposal which is then either rejected or accepted. If rejected the defining process starts over again, with a new version being proposed, until either the proposal is accepted, making it a level 3, or the mission is aborted.

5. Integrating a new agent

If you are new to the WARA-PS Core System the process of integrating a new agent is, next to directly contacting a member of the Core team and discussing with them what you wish to do, best performed in three steps.

1. Read through this document. To better understand the Core System architecture and how all the different parts are connected on a logical level you are recommended to read through the system description. This will in turn help in understanding how your own agent will be connected to and communicate with other parts of the system. *If you did not skip over the rest of the document and jumped directly to this chapter, you have already done this*.

2. Download and try some of the code examples, of how an agent presents itself in and communicates with the Core System (code-examples - Filer - Media and Logs @ WARAPS). A tutorial is provided for how to integrate an Arduino agent at (FILL IN WHEN TUTORIAL IS DONE AND UP AT THE PORTAL). Note that these code examples are precisely that, examples. This means that they do not contain the full source code, containing definitions for all the commands that currently can be sent through the Integration Test Tool.
    a. If you want to gain a deeper understanding of how the commands are validated, processed and executed, rather than just looking at the code and sending the pre-defined commands to your agent, the recommendation is to play with the example code. More specifically, to tweak some values to e.g. affect the course the agent is traveling or even try to make a basic definition for one of the undefined commands, such as “search-area”.
    A basic definition refers here to making the agent in question not only accept the command as valid but also execute a basic action such as, in the case of search-area, not only traveling along the ridge of the area but also adding more waypoints making it able to cross over and “scan” the area. Note that this is just a recommendation for if you wish to gain a deeper understanding and that for a general overview this is not required.

3. Lastly, we recommend that you read through the “*WARA-PS Core System API Specifications*” ref [2] document as it explains which data types has already been defined and given a specified topic name in the Core System as well as present the framework of how information is to be communicated. By framework, we refer to how the API specification describes and give examples of the strict naming scheme that is used in the Core System for publishing topics to and from agents. This is important because the concept of publishing and subscribing to topics, which is used to transmit data in the Core System, is name sensitive. Meaning that for information to be used and presented correctly in the Core System interface it needs to follow the correct naming scheme.
    a. If you cannot find the data type you wish to use in the API specification, please first contact a member of the Core System team (WARA-PS Portal (waraps.org)). This to ensure that two topic names are not created for one data type, by the case being that a topic name for the data existing but the specification not yet having been updated.
    However, if the case remains that the data type does not have a topic, you can then create your own for the data type in question, preferably in dialog with the Core System team to facilitate possible future inclusion in the API specification. You can also use MQTT-explorer (found at mqtt-explorer.com) to gain an overview of which data types are being published by existing agents today, and through that see whether your data type already has a specified topic name.

6. Abbreviations and Definitions

Following are a description of abbreviations and definitions used in this document and the WARA-PS project in general.

6.1 Abbreviations

Abbreviation	*Description*
ADS-B	Automatic Dependent Surveillance-Broadcast
AIICS	Artificial Intelligence and Integrated Computer Systems Division, IDA, LiU
AIS	Automatic Identification System
API	Application Program Interface
C2	Command and Control
CB90	Combat boat 90
DF	Delegation Framework
ERDC	Ericsson Research Data Center
GIS	Geographical Information System
GPS	Global Positioning System
IP	Internet Protocol
LiU	Linköpings Universitet
LoLo	Long-range Long-endurance
MQTT	MQ Telemetry Transport
NR	Node-Red
NRTK	Network Real Time Kinematic
PhD	Doctor of Philosophy
ROS	Robot Operating System, https://www.ros.org/
RTSP	Real Time Streaming Protocol
RTMP	Real Time Messaging Protocol
SMaRC	Swedish Maritime Robotics Centre
SNOD	SensorNODes (Underwaternodes created by Saab)
SSRS	Svenska Sjöräddnings Sällskapet
TST	Task Specification Tree
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
UxV	Unmanned Vehicle (of any kind)
WARA	WASP Autonomy Research Arena
WARA-PS	WASP Autonomy Research Arena – Public Safety
WASP	Wallenberg Artificial Intelligence, Autonomous Systems and Software Program

6.2 Definitions

Definition	*Description*	*WARA-PS Examples*
Agent	Entity which acts, directing its activity towards achieving goals, upon an environment using observation through sensors and consequent actuators. Agents can be human or robotic.	E.g. boat, drone
Collaboration	The process of two or more entities working together to complete a task or achieve a goal	Ex: swarm behaviour, UAV landing on USV

Command	Give an authoritative or peremptory order	Ex: continue, abort
Core system	Integrated system with collaborating platforms and resources	
Decision	Made by human operator or autonomous vehicles using AI	
Defined Mission	A mission with a complete Task Specification Tree specifying what tasks to perform and which agent is to do which task	
Delegation	Assignment of any authority from one agent to another agent to carry out specific activities	Delegation of tasks between agents (vehicles and humans)
Delegation framework	A framework to distribute tasks developed by LiU AIICS.	See DF - Delegation Framework
Fixed Wing	A flying machine, such as an airplane, with fixed wings, as opposed to a Rotary wing aircraft (helicopter, quadcopter)	Airpelago/SSRS drone
Mission	A high-level assignment given to a person or group of people	Missions are created after an alarm is raised, defining the circumstances and planned course of action. Missions can be either defined missions or undefined missions.
Object	A material thing that can be seen and touched	Ex: small cargo, detection on a map, unknown object
Operator	A person who operates equipment or a machine	Person using the management system
Order	An authoritative command or instruction	Command given by the operator
Piraya	An unmanned surface vehicle project under development by the Swedish shipyard company Kockums AB in collaboration with the Swedish military	USV
Progress	State of a task	
Sensor	A device which detects or measures a physical property and records, indicates, or otherwise responds to it	Ex: sonar, video
Sub-system	A self-contained system within a larger system	A system that is part of a joint system
System	A set of things working together as parts of a mechanism or an interconnecting network; a complex whole	Separate systems such as the Piraya, which can be a part of larger, more complex systems
Task	A piece of work to be done or undertaken	Missions are made up off of smaller assignments divided between vehicles
Task Specification Tree	A set of tasks to be performed in sequence or concurrent.	E.g move to position x,y,z and then search area A. For more info, see ref [3].
Undefined Mission	A mission with an incomplete Task Specification Tree meaning certain parts	

	of the mission is not specified e.g. which tasks to perform and which agents to use for the tasks. Becomes a defined mission when the missing parts are planned out and tasks are delegated to specific agents, completing the TST	
WARA-PS Arena	Position visualization (web based). Consists of a 2D map displaying vehicles (UxV's, boats, aircrafts, field agents) based on position data (AIS, ADS-B, ROS, Kockums, Naval SE, WARA-PS Android tracker). Part of C2.	WARA-PS Arena
WARA-PS Integration Test Tool	Open source solution for translating raw data and making it available in the Arena. Uses Node Red and MQTT.	
Waypoint	Coordinate/”point” that an agent is traveling towards.	