HORIZON 2020

D4.1 Data representation model V1

Augmented Reality Enriched Situation awareness for Border security
ARESIBO – GA 833805

Deliverable Information

Deliverable Number: D4.1
Date of Issue: 29/02/2029
Document Reference: N/A
Version Number: 1.0

Work Package: #4

Nature of Deliverable:
Report

Dissemination Level of Deliverable:
Public

Author(s): CERTH (Responsible)
Keywords: data model, interoperability, ontology, structured knowledge, heterogeneous data, incident and event description, contextual information, alerts, situational awareness
Abstract: This document summarises the performed investigation of the ARESIBO system specifications in order to describe the detailed functionality and information exchange between the ARESIBO components. On the basis of the aforementioned, an interoperable Data Model is defined, for structuring the exchanged content and more specifically for describing incidents, resources and tasks. The deliverable also presents the first iteration of the ARESIBO knowledge base, which is a knowledge representation model for semantically representing notions pertinent to the project.

Document History

Date	Version	Remarks
25.11.2019	0.1	**Deliverable outline (CERTH)**
02.12.2019	0.2	**Contribution in Chapter 3 (CERTH, all)**
10.12.2019	0.3	**Contribution in Chapter 4 (CERTH)**
14.02.2020	0.4	**Conclude introductory parts (CERTH)**
16.02.2020	0.9	**Major refinements (CERTH)**
29.02.202	0.91	**Internal Review (Airbus)**
29.02.2020	**1.0**	**Final version for submission (CERTH)**

Document Authors

Entity	Contributors
CERTH	**Marina, Riga (mriga@iti.gr)** **Ilias, Koulalis (iliask@iti.gr)** **George, Prountzos gprountzos@iti.gr)** **Kostas, Ioannidis (kioannid@iti.gr)**
UoA	**Vassilis, Papataxiarhis (vpap@di.uoa.gr)** **Michael, Loukeris (michael.loukeris@icloud.com)** **Kostas, Kyriakos (kostaskyriakos97@outlook.com)**
ConvCao	**Savvas, Apostolidis (sapostol@iti.gr)** **Thanasis, Kapoutsis (athakapo@iti.gr)**
TEKEVER	**Luis, Sousa (luis.sousa@tekever.com)** **Tiago, Marques (tiago.marques@tekever.com)**
ROBOTNIK	**Marc, Bosch (mbosch@robotnik.es)**
CMRE	**Luca, Berretta (Luca.Berretta@cmre.nato.int)**
OCEANSCAN	**Fernando, Bittencourt (fbittencourt@oceanscan-mst.com)**
IES	**Massimo, Cristaldi (m.cristaldi@i4es.it)** **Giovanni, Tusa (g.tusa@iessolutions.eu)**

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Meaning
CAP	Common Alerting Protocol
CCI	Command and Control Interface
CISE	Common Interface Sharing Environment
CQs	Competency Questions
CUSC	Core UCS
DM	Data Model
DS	Decision Support
EDXL-RM	Emergency Data Exchange Language-Resource Messaging
fps	Frames Per Second
GA	Grand Agreement
GCS	Ground Control Station
GGCS	Generic Ground Control Station
IR	Infrared (camera)
JAUS	Joint Architecture for Unmanned Systems
KB	Knowledge Base
KBS	Knowledge Base Service
OOPS	OntOlogy Pitfall Scanner
ORSD	Ontology Requirements Specification Document
OWL	Web Ontology Language
POI	Point of Interest
RDF	Resource Description Framework
RGB	Red-Green-Blue (camera)
ROI	Region of Interest
RTMP	Real-Time Messaging Protocol
RTSP	Real-Time Streaming Protocol
TBD	To Be Determined
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UCS	UAV Control System
UGV	Unmanned Ground Vehicle
URI	Uniform Resource Identifier
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
VSO	Vehicle Sales Ontology
W3C	WWW Consortium
WWW	World Wide Web
XML	Extensible Markup Language

1 Executive summary

ARESIBO comprises a highly interconnected system of various and independent modules that feed each other with the acquired information/data concerning surveillance and monitoring tasks in border territories. The proposed solution involves three main pillars of processing: (i) a complete configuration at tactical and execution level to optimise the synergies between humans and sensors, (ii) multiple modules for enhancing the understanding of the acquired data and (iii) C2 level capabilities for enhanced event reporting. To complete these objectives, ARESIBO integrates data from multiple sources including various UxVs and sensors such as UAVs and thermal cameras respectively. In addition, the system has to generate additional information on top of the acquired data for augmented situation awareness. The purpose of a common data model for the entire system is to assure and support interoperability and interconnectivity among different modules and to design a European-wide solution. On top of the infrastructure that implements the data model, the system enables multiple end-to-end interactions, leveraging data exchange, access, acquisition, processing and efficient reporting.

Towards these objectives, this document constitutes deliverable D4.1 "*Data representation model V1*" and focuses on presenting the first iteration of the ARESIBO *Data Model* and the ARESIBO *Ontology*. Regarding the ARESIBO *Data Model*, the main objective is to provide the system a proper interoperability framework so that several different UxVs will be fully functional under one common interface, the Generic Ground Control Station (GGCS). The rationale and the adopted solution rely on an extended version of the UCS 3.4 data model according to the main ARESIBO requirements and technologies. In addition, for the ARESIBO *Ontology*, also referred to as the ARESIBO *Knowledge Base* (KB) the framework will be developed to represent and enrich the acquired information. The KB receives input from different components operating within the scope of the ARESIBO system and encompasses a representation, formal editing and definition of the categories, properties and relations between the concepts, data and entities that substantiate many domains of discourse. The scope of the ARESIBO KB is to represent information and infer high-level knowledge directly to the interested system components, implied to the end-user.

The rest of the document describes thoroughly the initial version of the ARESIBO data model and KB and is structured as follows:

⇒ Chapter 2 introduces the main concepts of both the Data model and the ARESIBO Ontology

⇒ Chapter 3 presents all the current status and the main aspects of the initial version of the data model

⇒ Chapter 4 analyzes the main ARESIBO knowledge base/ontology based on which all the detected events will be represented

⇒ Chapter 5 concludes the document with closing remarks and directions for improving the data model and the ontology (including the accompanying tools and mechanisms) towards the final version foreseen to be reported in D4.2.

2 Introduction

The ARESIBO system is an end-to-end solution for collecting information from multiple data sources such as UxVs, detection sensors and legacy systems, processing and assessing events and alerting accordingly the relevant personnel with proper messages. The multitude of data originators and consumers in flexible configurations within the ARESIBO platform mandates a robust framework for data connectivity, integration, processing and exchange among the involved modules and services.

The data-source integration framework focuses on the development of one interoperability layer which consists of software modules that collaborate, coordinate, develop and transfer knowledge for a better situation awareness. More specifically, multiple input-output interfaces of the system can send and receive standardized data structures as part of the information transactions and services. The interoperability layer will rely on the development of these standardized data structures following a predefined template, the ARESIBO data model. Such instances will ensure an efficient and effective data flow in both directions, from and to the main C2 system (Figure 1).

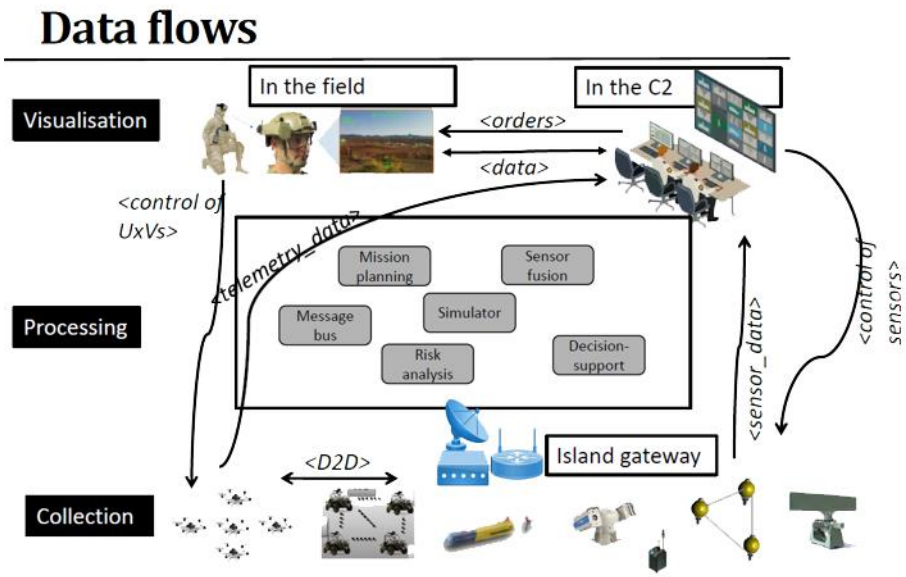

Figure 1: A general diagram of data flows within the ARESIBO System

As the identification of specific components and the finalization of the overall architecture is currently on-going, the initial version for the corresponding data model will comprise the basis of the central infrastructure for appropriate data exchanging and provide the interoperability capacities. Towards this objective, a central message bus will be deployed operating as the main means of exchanging data and processing results between the variant services. Following a micro-services approach for data harmonization, each component is autonomous to a large extend and all the interactions among the components are accomplished via the central message bus. Thus, essentially, the central message bus implements the desired interoperability layer. Hence, each component that processes information that might be relevant and beneficial to other services will update specific topics following one specific format. The data model reflects the common structures that are processed by all system's service.

In order to achieve efficient interoperability capabilities, ontologies play a significant role in resolving semantic heterogeneity. The overall architecture incorporates the use of relevant ontologies for explicit description of the semantics of information sources to facilitate the communication between the different components of the architecture. Ontologies, commonly referred as Knowledge Base (KB), serve as a knowledge representation model for incidents,

resources and tasks of interests that are reported in the context of the ARESIBO system. Figure 2 represents the correlation between the interoperability layer and the relevant ontologies in respect with the foreseen data flow procedure.

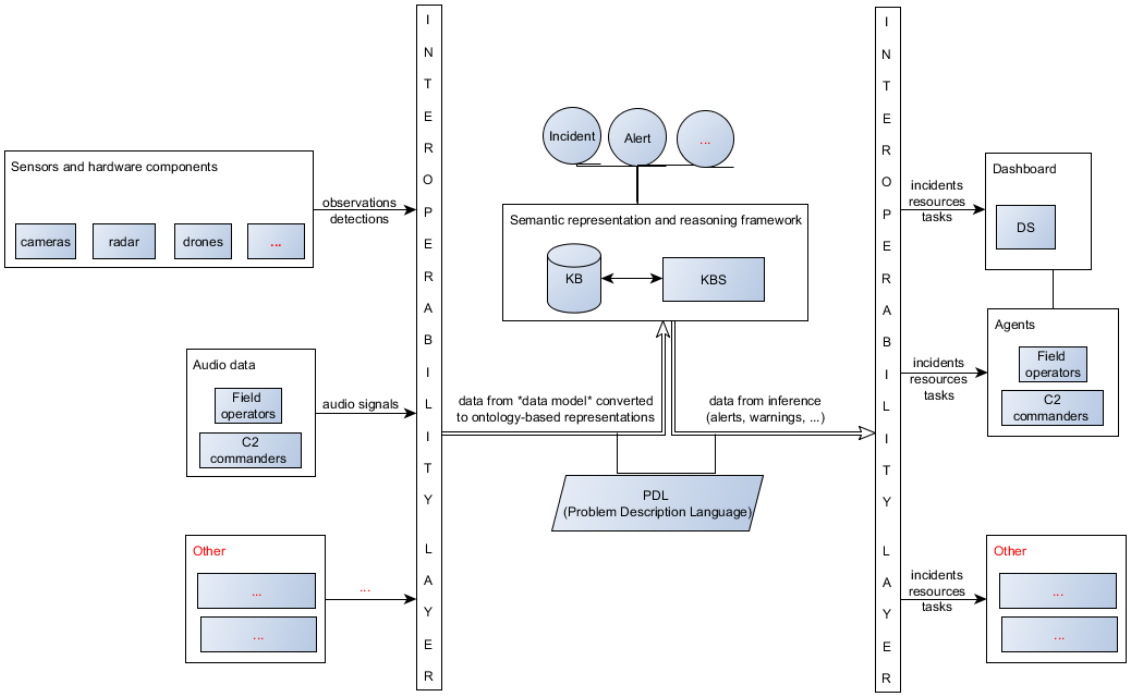

Figure 2: Data flows, within the context of the ARESIBO System, from a technical perspective.

The rest of the deliverable is structured as follows. Section 3 provides a detailed description for the first version of the ARESIBO Data Model. The section is thoroughly analysed with multiple sub-sections where the technical requirements are defined per component. Additionally, existing standards and protocol adaptors are provided. Section 4 involves the analysis of the ontology concept as well as an initial description of the first draft of the ARESIBO ontology. The document concludes with the basic remarks and setups the additional work that should be performed to attain the second and final version of the data model.

3 Definition of the ARESIBO Data Model

The ARESIBO system relies on the operation of multiple types of both sensors (static and dynamic) and UxVs, thus the unified system must be capable of processing vast amounts of multimodal data while the transmission should be performed in distinct ways. Towards this objective, the definition of an interoperability layer implemented through a data model is of paramount importance as it will comprise the core of the system. In general, a data model refers to an abstract model that organizes efficiently data elements, standardizes their interconnection and identifies their properties with the real-world entities. Additionally, the interoperability layer involves the definition of the required communication protocols so that the deployed sensors and UxVs could exchange the desired information. Based on the ongoing analysis performed on the architecture and data models that already exists, an extended UCS 3.4 version might be the most efficient alternative to be adopted as the main data model for the ARESIBO system as it covers multiple relevant domains like JAUS and JANUS. Figure 3 presents the overall concept of the implementation for the interoperability layer and the connection of the overall system with legacy systems through the appropriate data model/connectors.

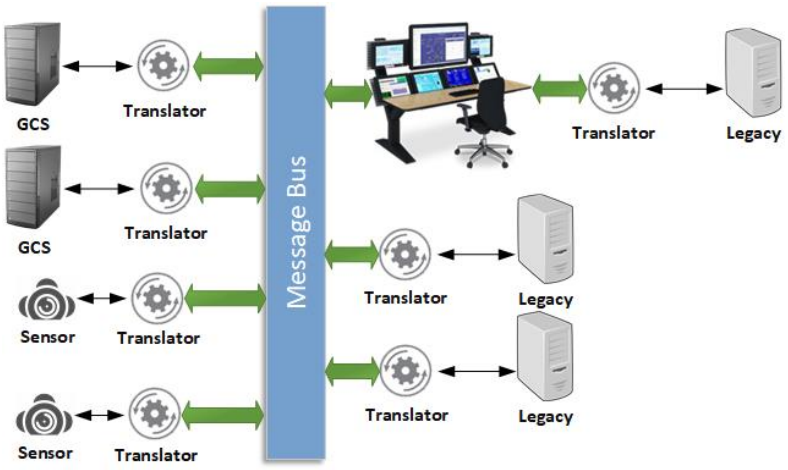

Figure 3: Logical diagram of the Interoperability layer.

3.1 Technical requirements defined per component

3.1.1 UAV and sensors (T3.2)

The main target of the relevant task (T3.2) is to build a reliable sensor infrastructure according to the project's challenging field operations. Within this context, specific data will be exchanged, uplink (from GCS to the platform): piloting and control of the platform and downlink (from the platform to the GCS): sensor data. The outcome of this process will enable the integration of LAUV and marine sensors for communication and operation in conjunction with the ARESIBO system, allowing mission monitoring, analysis and visualization of collected data. More details are summarised in Table 1 below.

Table 1 – T3.2 component details

Component's name:	UAV and sensors
Consumes input from:	• Mission Editor/Swarming Mission Planner • Sensor configuration • Payload control • Plans: A plan is a set of waypoints, where each waypoint has a latitude, longitude, depth, speed, and payload.

	Payload is the set of configurations of each sensor available in the vehicle that must be activated at the waypoint. • Commands: Actions that the vehicle must perform. For example, start a plan, abort, etc.
HW components involved:	• UxV • GCS • Sidescan sonar • Camera • Sonars • GPS • Iridium • GSM
Produces output:	• Sensor data streams • Sidescan Raw • JPG, MJPG • UxVs telemetry: proprietary vehicle log with path, speed, operating time, etc.
Involved standards/protocols:	• IMC[1] (most important are the messages: Plan Specification, Starting manoeuvre, Estimated State) • WGS84 coordinate system • STANAG 4586 • STANAG 4609 • OGC

3.1.2 Swarming robots and human-robot collaboration (T3.3)

The main focus of the relevant task (T3.3) is the development of a module able to exploit fused and raw real-time data towards establishing a fully autonomous operational framework for all surveying assets. The outcome of this task will be a module that will get as input the mission details (provided by the Mission Editor – Section 3.1.7) and relevant telemetry data and will produce as output the coordinates (WGS84 coordinate system) and the waypoints of the involved UxVs. More details are summarised in Table 2 below.

Table 2 – T3.3 component details

Component's name:	**Resource control**
Consumes input from:	• Mission Editor: WGS84 coordinates, polygon ROI, user-defined waypoints, assets participating, missionID, UxV names, etc. • Telemetry data
HW components involved:	• UxVs GCS
Produces output:	• Waypoints of the UxVs: altitude, longitude, latitude, kind of

[1] Available at: https://github.com/oceanscan/imc/blob/master/IMC.xml

	turns
Involved standards/protocols:	• WGS84 coordinate system

3.1.3 Assets' communication (T3.4)

The work that will be carried out under the task T3.4 involves the translation between the ARESIBO data model and the platform specific communication protocols defined on the basis of the different types of assets from different manufacturers. Input data are sourced from the assets' sensors (on board) to the GCS, while output data are produced from the UAV sensors. The detailed information exchanged as input and output to the relevant module is described in Table 3.

Table 3 – T3.4 component details

Component's name:	**Assets communication**
Consumes input from:	• Asset ID: vessel information – 3D location, speed, course, heading • Weather data: local wind speed, wind direction, pressure, wave height, wave direction • Tracks: id, source, type, label, 2D location, details • Low/High Level Tasking: ID, type, 3D location, action • Point of Interest (POI): ID, type, 2D location, action • Area of Interest (AOI): ID, type, 2D location, action
HW components involved:	• Fixed-wing UAV GCS • UUVs, USVs and UGVs
Produces output:	• Aircraft information: 3D location, attitude • Aircraft status, battery, fuel, communication interfaces (comms) • Route information: waypoint list, active waypoint • Sensor information: sensor orientation, field-of-view • Sensor status: available, active • Weather: local wind speed, wind direction, air pressure • Video streams: MPEG-TS/H.264 + MISB 0601 KLV (metadata) • Still imagery: GeoTiff • Tracks: see above (sources: EO, IR, AIS) • Area of interest (AOI), Point of interest (POI): either coming as tasking requests from external systems, or directly introduced by the UAV operator
Involved standards/protocols:	• -

3.1.4 Voice and Video (T3.5)

The main goal of the task (T3.5) is the development of a communication network that will ensure a secure real time video and voice exchange between the field units and the tactical C2 center. The outcome of this task will be a communication network that ensures secure

and highly reliable bidirectional connectivity between all involved parties in the use cases. Table 4 presents more technical details regarding this task. In addition, Viasat provides an IP camera ('AXIS Q6215-LE PTZ Network Camera') mounted on the communication hub (vehicle) with a video player on-board the vehicle able to play the real time video and audio generated by the on-board IP camera.

Table 4 – T3.5 component details

Component's name:	Voice and Video
Consumes input from:	• On-board IP camera (Video stream and Audio stream) • On-field cameras (Video stream and Audio stream) • Field agents radio terminals (Voice)
HW components involved:	• On-board radio terminal • Nomadic satellite antenna • LTE module (cellular router) • Mikrotik RB4011iGS+RM router • Mikrotik CRS112-8P-4S-IN switch
Produces output:	• Video and audio streams from on-board IP camera • Video and audio streams from on-field cameras • Voice from field agents radio terminals
Involved standards/protocols:	• H.264 • H.265 • RTP/RTSP • VoIP

3.1.5 Visual Object Detection (T3.6)

The main objective of the relevant task (T3.6) is to develop novel object recognition algorithms that can identify multiple objects of interest (e.g., person, car, boat, ship, inflated, speedboat, etc.), on the basis of visual data sourced from multiple types of sensors, such as RGB, IR and thermal cameras. The relevant module shall take as input data (video streams) from any kind of sensor providing the system with visual input and will produce as output details about the objects detected (type, timestamp, geolocation, etc.). More details are summarised in Table 5 below.

Table 5 – T3.6 component details

Component's name:	Visual Object Detection
Consumes input from:	• Video streams (over RTMP or RTSP) • Telemetry data: GPS coordinates, timestamp • Asset-based ID
HW components involved:	• UAV/UGV/cameras • Any kind of sensor providing the system with visual input
Produces output:	• Video detections: objects detected, timestamp, geolocation, confidence, bounding box, width, height, fps, asset ID

	(sender)
Involved standards/protocols:	• RTMP • RTSP

3.1.6 Semantic Representation and Reasoning (T4.1)

The main objective of the relevant task (T4.1) is to develop an ontology-based schema for the semantic representation and reasoning of the heterogeneous data communicated within the system, in order to structure and semantically enrich the involved content and to infer new, enhanced information that can increase the situational awareness of the end user. The relevant component will handle as input data from the visual detectors, spatiotemporal information of the detected entities, of the operational assets, etc., and may produce as output warnings or alerts that describe the detected incidents or conditions existing in an area under surveillance, on the basis of specific rules and criteria described by the end users. Details are summarised in Table 6 below.

Table 6 – T4.1 component details

Component's name:	**Semantic Representation and reasoning**
Consumes input from:	• Visual detector: detected entity (person(s), object(s)), location of detected entity, distance from critical points (shore, borders, etc.) • Operational assets' metadata: ID, status (available/not-available), telemetry data
HW components involved:	• Cameras, UxVs
Produces output:	• Incidents • Alerts • Tasking
Involved standards/protocols:	• OWL-compliant representation

3.1.7 Mission Editor (T4.2)

The main focus of the relevant task (T4.2) is to provide a module that can support functionalities related to the definition of missions, in terms of robots' movement and their undertaken actions. The definition of a mission will be realized through commands defined in the Textual editor and/or actions in the Visual editor. Users will have the opportunity to create, update, compile and validate their missions. The Mission Editor module will get as input the availability and the status of the different operational UxVs and sensors and will produce as output a robotic mission with details about the mission type, content and route path as well as the id of the asset who will follow the proposed mission. More details are summarised in Table 7 below.

Table 7 – T4.2 component details

Component's name:	**Mission Editor**
Consumes input from:	• Availability/status of UxV/sensor: asset ID, type, status information (available/not-available)

	• User input for the UxV assets involved in the missions
HW components involved:	• UAV/UGV/USV/UUV
Produces output:	• Robotic missions: mission id/type/content, vehicle id, route path, sequence of waypoints
Involved standards/protocols:	-

3.1.8 Simulation Engine (T4.3)

The simulation engine's focus is to support the learning process of the operators by recreating virtualized training environments. This system's component considers a set of parameters that includes setup configurations about the scenario, vehicles and sensors as well as environmental conditions and generate the required outcomes for evaluation. More details are summarized in Table 8 below.

Table 8 – T4.3 component details

Component's name:	**Simulation Engine**
Consumes input from:	• Mission editor
HW components involved:	• TBD
Produces output:	• Vehicle telemetry: position (latitude, longitude, altitude), attitude (roll, pitch and yaw angles), speeds. The telemetry can be "real" or "estimated", i.e., affected by the sensors' navigation error. • Sensor data: status of the sensor (on/off/working/not working), target in sight, detections • Environmental data: weather conditions (sun, rain, fog, cloud, humidity, air temperature, water temperature, water salinity, waves, water turbidity, etc.
Involved standards/protocols:	• STANAG 4603: Modelling and simulation architecture standards for technical interoperability: HLA

3.1.9 Decision support functionalities (T4.4)

Task 4.4 aims to provide C2 operators with decision support functionalities. These focus on effective resource and task management as well as generation relevant notifications. More details are provided in Table 9 below.

Table 9 – T4.4 component details

Component's name:	**Decision support**
Consumes input from:	• Risk analysis • Sensor Fusion Engine
HW components involved:	• -

Produces output:	• Type of action • Event creation/update • Mission creation/update • Resources create/update • Communication messages
Involved standards/protocols:	• EMSI (Missions, Resources, Events) • CAP (Alerts, Notifications, Events) • EDXL (Missions, Resources, Situation Assessment)

3.1.10 Sensor Fusion Engine (T4.5)

The main objective of the relevant task (T4.5) is to implement a module that will facilitate the real-time integration (fusion) and interpretation of different types of raw data sources originating from different sensors. The Sensor Fusion Engine will get at input numerical data from any kind of sensor operating to the system and will produce as output fused data in the form of real-time alerts. More details are summarised in Table 10 below.

Table 10 – T4.5 component details

Component's name:	**Sensor Fusion Engine**
Consumes input from:	• Sensors: numerical sensor streams (raw data), sensor type, timestamp • Telemetry data
HW components involved:	• Any kind of sensor providing the system with numerical inputs
Produces output:	• Real-time alerts: alert id, type, category, severity, location, timestamp, description
Incidents detected/reported/handled:	• Alerts that can be detected based on numerical data streams
Involved standards/protocols:	• UCS3.4

3.1.11 Risk Models (T4.6)

The main target of the relevant task (T4.6) is to develop a risk analysis framework for incorporating the forecasting models that will be used for the assessment of risks and potential threats. A semantic representation of the CIRAM (Common Integrated Risk Analysis Framework) will be used for the description of potential risks and impact levels and for the assessment of threats. The aforementioned module will get as input any numerical data provided by the sensors and will produce risk predictions and recommendations to mitigate the arisen threats. More details are summarised in Table 11 below.

Table 11 – T4.6 component details

Component's name:	**Risk Models**
Consumes input from:	• Sensors: numerical sensor streams (raw data)

	• Telemetry data
HW components involved:	• Any kind of sensor providing the system with numerical inputs
Produces output:	• Risk prediction (progress of a monitored situation): id, type, category, severity, location, timestamp, description • Recommendations to mitigate a risk
Incidents detected/reported/handled:	• Risk predictions
Involved standards/protocols:	• Frontex CIRAM

3.1.12 XR visualisation (T5.1-4)

T5.1-4 focus on developing tools and services that achieve AR functionalities for the C2, field officers and commanders. They are responsible for providing them with real-time contextual information and conditions in various mediums utilizing projecting hardware. More details are summarised in Table 12 below.

Table 12 – T5.1-4 component details

Component's name:	**AR and time-based visualization**
Consumes input from:	• ARESIBO engine
HW components involved:	• Realwear HMT-1 • Hololens2
Produces output:	• Video/voice stream, Aurio/Video • NATO/other symbols • Notes, Locations, Map, Radar info
Incidents detected/reported/handled:	• NVG (Nato Vector Graphics) • EXDL (Emergency Data exchange Language)
Involved standards/protocols:	• ARESIBO engine

3.1.13 Mission status (T6.2)

Mission Status will provide messages with information about the progression of the mission. It contains telemetry data, the current status of a mission and can detect failures and possible complications. More details are summarised in Table 13 below.

Table 13 – T6.2 component details

Component's name:	**Mission status**
Consumes input from:	• UGVs
HW components involved:	• UGV (cameras, LIDAR.GPS)

Produces output:	• MissionStatus: information about how the mission is being done. • Telemetry: time, current GPS coordinates, orientation wrt North, speed
Involved standards/protocols:	• ROS

3.2 End-user requirements

As the discussions for the identification of the end-user requirements are currently progressing, a dedicated end-user workshop was organized at during M8 (December 2019) in order to specify the main topics of interests. The latter will drive the main requirements that the technical partners should assess and consider during each development circle. Some initial feedback from the end-users was accomplished through the collection of specific information with the use of questionnaires and requirements tables. Thus, the initial assessment resulted into the basic functionalities/capabilities and sensors that will be utilized for the ARESIBO operational scenarios. An initial version of one common data model analysed in this deliverable as well as the main legacy systems that will be incorporated in the main framework were identified.

3.3 Existing Standards and Protocol adaptors

3.3.1 STANAG 4586

In 1998, a NATO Specialist Team comprising members of government and industry, including Common Data Link (CDL) Systems, began work on NATO Standardization Agreement 4586 (NATO 2012), a document conceived to standardize UCS interfaces to help enable UAV systems interoperability (Marques, 2012). The objective of STANAG 4586 is to specify the interfaces that shall be implemented in order to achieve the required Level of Interoperability (LOI) according to the defined concept of operations (CONOPS). STANAG 4586 is the first step towards enabling GCS to control and monitor multiple types of unmanned aircraft, improving overall cost by reusing GCS, and enabling competition at the system level for complete GCS solutions.
The architecture proposed within the STANAG 4586 standard comprises the following components (NATO, 2012):

⇒ the **Core UCS** (CUCS), an interface to handle the UAV common/core processes.
⇒ the **Data Link Interface** (DLI) that enable operations with legacy as well as future UAV systems. In other words, the DLI enables the CUCS to produce and understand messages for control, status, payloads and more.
⇒ the **Command and Control Interface** (CCI) for UAV and UAV payload data dissemination, to support legacy and evolving NATO C4I systems and architectures; and
⇒ the **Human Computer Interface** (HCI) which defines the to support the interface to the UAV system operators.

The STANAG 4586 m*essage handling* approach specifies that each message shall use a wrapper structure with the following fields:

⇒ Source port: Standard UDP header
⇒ Destination port: Standard UDP header
⇒ Packet length: Standard UDP header
⇒ UDP Checksum: Standard UDP header
⇒ Sequence #: Segmented data sequence

⇒ Message length: 16bit Unsigned integer
⇒ Source ID: ID of UAS element
⇒ Destination ID: ID of UAS element
⇒ Message Type: 16bit Unsigned integer
⇒ Message Properties: Bitmapped field
⇒ Optional Checksum: determines presence of errors.

Within the STANAG 4586 standard, generic messages are clustered as Functional Groups, which can support the following entities/concepts:

⇒ System ID: e.g., Vehicle ID
⇒ Flight Vehicle Command: e.g. Air Vehicle Lights
⇒ Flight Vehicle Status: e.g., Vehicle Configuration
⇒ Flight Vehicle Payload Relevant: e.g., Inertial States
⇒ IFF Command: e.g., IFF Code Command
⇒ IFF Status: e.g., IFF Status Report
⇒ ATC Interface Command: e.g., NAVAID Radio Command
⇒ ATC Interface Status: e.g., NAVAID Radio Status
⇒ Vehicle Auxiliary Command: e.g., Vehicle Auxiliary Command
⇒ Vehicle Auxiliary Status: e.g., Vehicle Auxiliary Status
⇒ Mission Command and Status: e.g., AV Route
⇒ Subsystem Status: e.g., Heartbeat Message
⇒ Miscellaneous Messages: e.g., Link Audio Status
⇒ Payload Command: e.g., Terrain Data Update
⇒ Payload Status: e.g., Terrain Data Request
⇒ Weapons Command: e.g., Stores Management System Command
⇒ Weapons Status: e.g., Stores Management System Status
⇒ Data Link Discovery: e.g., Data Link Control Authorization Request
⇒ Data Link Command: e.g., Link Health Command
⇒ Data Link Status: e.g., Data Link to Vehicle ID Report
⇒ Data Link Transition: e.g., Handover Status Report
⇒ General Pre-connection Configuration: e.g., Field Configuration Request
⇒ General Post-connection Configuration: e.g., Display Unit Request
⇒ Autonomy: e.g., Area Definition
⇒ VSM Forced Commands: e.g., Field Change Float Command
⇒ Draw Interface: e.g., Draw Line

Within each Functional Group, a list of more specific fields (messages) is defined. Moreover, complex messages can be composed by combining one or more messages from different Functional Groups. For example, the messages defined in the “Mission Command and Status Messages” (Table 14) shall compose the Mission Command and Status Functional Group of messages. A detailed list of messages and functional groups can be found in (NATO, 2012).

Table 14 – Mission Command and Status Messages

New Msg#	Old Msg#	Description	Push/Pull	Source	Allowable Max Latency (msec)
13000	800	Mission Upload Command	Push	CUCS	1,000

13001	801	AV Route	Push/Pull	CUCS/VSM[2]	2,000
13002	802	AV Position Waypoint	Push/Pull	CUCS/VSM	2,000
13003	803	AV Loiter Waypoint	Push/Pull	CUCS/VSM	2,000
13004	804	Payload Action Waypoint	Push/Pull	CUCS/VSM	2,000
13005	805	Airframe Action Waypoint	Push/Pull	CUCS/VSM	2,000
13006	806	Vehicle Specific Waypoint	Push/Pull	CUCS/VSM	2,000
14000	900	Mission Upload/Download Status	Push	VSM	2,000
	901-999	Unassigned message types in the range of 13000 – 14999 are reserved			

3.3.2 STANAG 4609

The Standardization Agreement 4609 (NATO, 2009) aims to enable and achieve interoperability of motion imagery (MI) systems in a NATO Combined Service Environment. Motion imagery enhances the capabilities of the commanders and operators and helps them meet efficiently the operational and tactical objectives for intelligence, reconnaissance and surveillance. STANAG 4609 is intended to provide common methods for exchange of MI across systems within and among NATO nations.

More specifically, STANAG 4609 describes the requirements about compressed, uncompressed and related motion imagery sampling structures, motion imagery time standards, motion imagery metadata standards, interconnections, and common language descriptions of motion imagery system parameters. It is based on commercial systems and components designed on the basis of existing open standards for providing interoperability between NATO compliant services.

The core attributes of STANAG 4609 for motion imagery are described in (NATO, 2009). The cornerstone is MPEG-2, since both visible light and infrared MI systems shall be able to decode all MPEG-2 transport streams with MPEG-2 compressed data types (Standard Definition, Enhanced Definition, High Definition) up to and including all H.264 compressed data types. The collection of standards comprises of the definitions of:

⇒ **sampling structures**, including standards for *analog video migration, digital motion imagery* and *high definition television systems*.

⇒ **compression systems**, including standards for the *Digital Motion Imagery Compression Systems, Use of MPEG-2 System Streams*, *Motion Imagery Still Frames*, and more.

⇒ **metadata**, including standards for the *Motion Imagery Metadata Dictionary Structure, Time Code Embedding, Time Reference Synchronization, Unmanned Aerial System (UAS) Datalink Local Metadata Set*, and more; and

⇒ **file formats**, including standards for the *use of MPEG-2 System Streams for Simple File Applications, Advanced File Format* and *Timing Reconciliation Universal Metadata Set for Digital Motion Imagery*.

3.3.3 JAUS/JANUS

Joint Architecture for Unmanned Systems (JAUS) standard is an architecture that enables the communication with unmanned air, ground and sea vehicles. JAUS is built upon five characteristics: mission isolation, computer hardware independence, technology independence, and operator use independence so that applicability to the entire domain of unmanned systems is achieved. JAUS was originally proposed to provide an open architecture for the domain of Unmanned Ground Robots. Recently, the standard has expanded to cover additional domains and capabilities to better defined based on a Service Oriented Architecture (Kent et al, 2014). JAUS standard is divided into the Domain Model, which provides the objectives, and the Reference Architecture, which provides engineering

[2] VSM stands for Vehicle Specific Module

specifications for the architecture framework, a message format definition and a set of standard messages.
The top entity of the Architecture Framework is identified as the *System*. A JAUS *System* is structured as a 3-tiered logical hierarchy consisting of *Subsystems*, *Nodes*, and *Components*. A *System* might be consisted of one or more *Subsystems*. The latter typically represents a physical entity in the system network, such as an unmanned vehicle or operator control unit. *Subsystems* can be divided into *Nodes*, which represent a physical computing endpoint in the system. As for example, a *Node* might be a computer or microcontroller within a *Subsystem*. In additions, *Nodes* can host one or more *Components*, which are commonly applications or threads running on the *Node*. Finally, *Components* are constituted by one or more *Services* which eventually provide valuable functionalities to the system (Serrano et al., 2015), (Galluzo and Kent 2011). A JAUS Component is the only addressable entity within the JAUS System and is uniquely identified using a dotted address consisting of SubsystemID.NodeID.ComponentID. There are two special JAUS entities required for routing messages. At the highest level, a communicator is the portal for all messages within a *Subsystem*. Similarly, a *Node* manages the portal for all messages within a *Node*. Communicators and node managers can be viewed simply as routers. Such logical structure is presented in Figure 4.

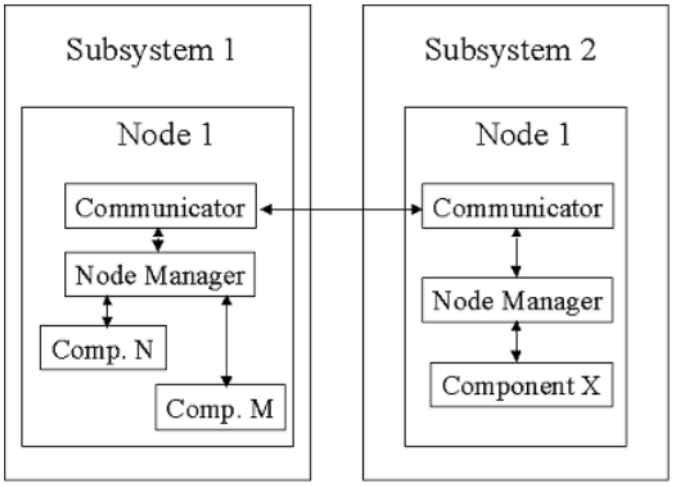

Figure 4: Communication within a JAUS System

The information is exchanged in the form of *Messages* having 16-bit headers and can be categorized into seven distinct classes: (i) Command, (ii) Query, (iii) Inform, (iv) Event setup, (v) Event notification, (vi) Node management and (vii) Experimental having 16-bit headers that define the below characteristics.

- ⇒ Message Properties
- ⇒ Command Code
- ⇒ Destination Instance ID
- ⇒ Destination Component ID
- ⇒ Destination Node ID
- ⇒ Destination Subsystem ID
- ⇒ Source Instance ID
- ⇒ Source Component ID
- ⇒ Source Node ID
- ⇒ Source Subsystem ID
- ⇒ Data Control
- ⇒ Sequence Number

Standardization Agreement 4748, Digital Underwater Signalling Standard for Network Node Discovery & Interoperability, aims to enable and achieve communication interoperability between underwater military (UUVs) and civilian maritime assets and sharing information among various heterogeneous sensors, ships, submarines, UAVs, gateway buoys and sensor networks. To this end, **JANUS** is proposed as the physical standard, which specifies the layer-coding scheme allowing the transmittance of information in a common format that can be decoded by compliant assets. The schema, that is Frequency Hopped (FH) Binary Frequency Shift Keying (BFSK), is simple to implement and robust to temporal and frequency fading in the harsh UW acoustic propagation environment.
In the JANUS FH-BFSK scheme, binary data bits are mapped into one of a pair of time-windowed CW tones of unspecified phase, selected from 13 evenly spaced tone pair choices spanning the frequency band, having the initial frequency band allocation at 9440 – 13600 Hz. The process to generate a Janus Packet, is shown in the following diagram (Figure 5).

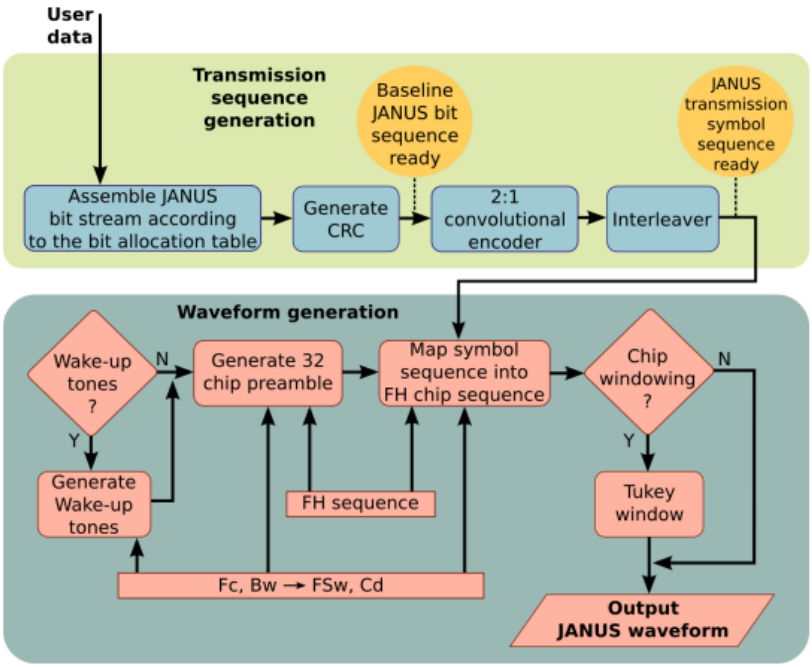

Figure 5: Block diagram of the JANUS Baseline Packet encoding process

3.3.4 UCS and UCS 3.4

The Unmanned Aircraft System (UAS) Control Segment (UCS) Architecture is a software interface, data-model, and business system architecture that defines the rules and conventions for developing interoperable software components for UAS Ground Control Stations (GCS). The operational objectives include support for both UA platform and sensor C2, sensor product availability, and UA status.
UCS architecture is developed with the aim of enabling integration, reuse of services between programs and Services along with the decrease of costs of unmanned systems. STANAG 4586 was a first step towards interoperability of control segments and, since a single common monolithic architecture renders upgrades slower and more expensive, UCS Architecture progressed it by defining an open and scalable infrastructure that supports flexible integration of ground control system services across a variety of deployment scenarios.
More specifically UCS Architecture:

- ⇒ Identifies additional system and equipment use cases.
- ⇒ Defines a modeling framework for the specification, integration, implementation and deployment of control station software.

- ⇒ Is designed on a platform independent model, which allows implementations on different infrastructures.
- ⇒ Includes an open Application Architecture Data Model, which is based on real-world entities, describes the information required by UCS domains in their internal and external interactions and define the semantics of all interoperable implementations
- ⇒ Defines and validates the Application Architecture Domain Model
- ⇒ Updates the requirements for the UCS Architecture tool environment and UCS Architecture Quality Management System
- ⇒ Includes airworthiness, system safety and Information Assurance views.

In order to achieve and maintain interoperability of compliant UCS systems, a Model Driven Architecture is usually selected. It separates the UCS business logic and data from the underlying technology of the application platform and the software runtime architecture by expressing the application software as a set of Platform Independent Models (PIMs) and then transformed into Platform Specific Models (PSMs) taking into account platform and software runtime architecture choices. The latter is presented in Figure 6 .

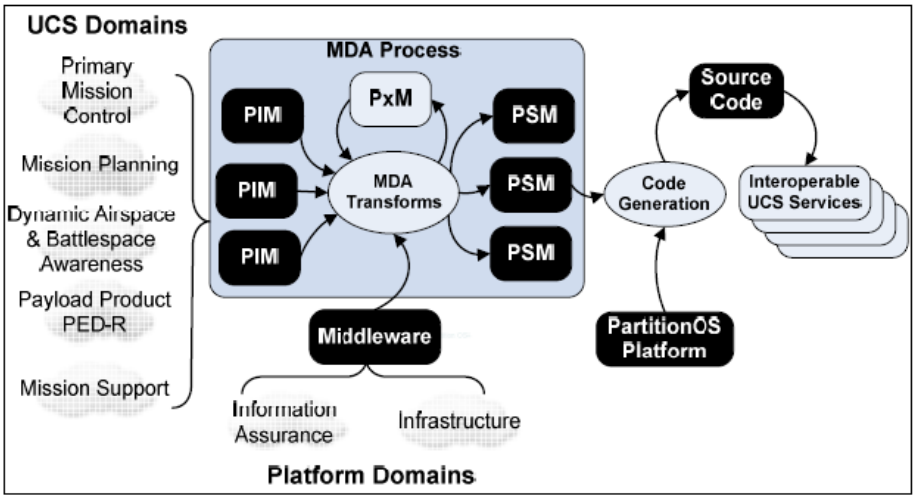

Figure 6: UCS Model Driven Architecture Process

3.4 ARESIBO Data Model

All services and modules that are planned to operate within the ARESIBO system will use the proposed structures defined in the ARESIBO data model in order to exchange data. For the first prototype of the ARESIBO internal data model, a set of messages was defined, following some basic rules:

- ⇒ the definition of messages will be mainly based on the UCS3.4 data model,
- ⇒ whenever is required, the extension of the UCS3.4 will be used, as this was proposed in the ROBORDER EU-funded project (ROBORDER 2017).
- ⇒ whenever needed, new data models were proposed as an extension of all the above. These definitions will be mainly considered as pure definitions of the ARESIBO data model from the beginning.

For every concept that follows, each included table comprises a result of the end-user requirements assessment which were collected from all the technical partners. Each responsible technical partner, either a UxV manufacturer or a service provider, identified the main topics of interests as well as the corresponding fields and their length. Such structure is particularly essential so as each main component could be able to consume data and produce knowledge.

3.4.1 Plan

This field contains instructions for a specific UxV operating in a mission. The instructions include information regarding the *waypoints* that the UxV should follow and *commands* (on/off, gimbal moves, etc.) concerning the payload and sensors that it carries. The description that follows is not included in the UCS3.4, so it should be considered as a potential extension.

Table 15 – Plan structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	vehicle identification
mission_id	int	4	None	a unique identification number, describing a mission that is likely to contain more than one UxVs
speed	float	4	m/s	the proposed speed for a mission for a specific UxV
route_path_segment	array of structures	-	-	definition of every path segment included in a mission
sensors[3]	structure	-	-	a structure containing commands for the UxV payload

Table 16 – route_path_segment structure

Field	Type	Length	Units	Description
route_path_segment_id	int	4	None	a unique identification number, describing each route_path_segment
waypoint	structure	-	-	a structure containing WGS84 coordinates and special instructions for a mission point to be reached

3.4.2 Waypoint

This field describes a way point used for the guidance of UxVs during a mission. It contains the exact position that a UxV should approach (formatted in WGS84 coordinates) and a characterization of the way point to provide instructions for those that need special treatment to be approached. A detailed description of the involved fields is presented in Table 17. Note that the field *altitude* can get both positive and negative values, in order to provide altitude and depth information, depending on the type of UxV used.

Table 17 – WaypointType structure

Field	Type	Length	Units	Description
altitude	float	4	m	indicates the altitude of the Waypoint
position	Position2DType	16	-	position of waypoint
waypoint_kind	byte	1	Enumerated	type of waypoint WaypointKindEnumTypeLDM: APPROACH = 0, APPROACH_FINAL_POINT = 1, APPROACH_INITIAL_POINT = 2, HARD_DITCH = 3, NAV_ONLY = 4, PASSIVE = 5, RUNWAY_LIMIT = 6,

[3] It should be noted that the *sensors* structure remains unidentified in the current version, and will be specified in future versions of the data model, on the basis of the requirements and the capabilities that the actual involved sensors will have.

				RUNWAY_START = 7, RUNWAY_THRESHOLD = 8, TAKEOFF = 9, TAKEOFF_FINAL_POINT = 10, TAKEOFF_INITIAL_POINT = 11, TAXI = 12, TOUCHDOWN = 13

3.4.3 Command/Action

Whenever a UxV that can support teleoperation needs to be tasked, guided, or steered, a direct steering command is provided with this structure. The typical use case is when the command is given by a module that is detached by the vehicle simulator, in which case the latter is in charge of computing and updating the dynamic/kinematic model of the UxV and receive the reference commands from the guidance module. The guidance module sends reference commands using the *VehicleSteeringCommand* data type.

Table 18 – VehicleSteeringCommand structure

Field	Type	Length	Units	Description
platform_id	Integer32	4	-	unique ID of the platform
target_speed	Float64	8	m/s	commanded forward speed
target_heading	Float64	8	deg	commanded heading wrt True North
target_altitude	Float64	8	m	commanded altitude wrt the mean sea level (negative values for depth below water surface)

3.4.4 Payload

Payload is the set of configurations of each sensor available in the vehicle that must be activated at each waypoint. Such structures provide better flexibility in operating more appropriate the used equipment. The structure provides basic information about the status and the type of the available payload.

Table 19 – PayloadType structure

Field	Type	Length	Units	Description
next_availability_time	double	8	s	nextAvailabilityTime is a TimeType which specifies the next time the payload carried by this air vehicle will be available.
payload	PayloadType	324+	-	payload is a PayloadType which specifies the payload carried by this air vehicle for each PayloadType.
payload_recorder	PayloadDataRecorderType	106	-	payloadRecorder is a PayloadDataRecorderType which specifies the payload data recorded carried by this air vehicle for each PayloadType.
payload_report	SubsystemReportType	153?	-	payloadReport is a SubsystemReportType which specifies the information about the definition and location of the report for PayloadType.
pedestal	PedestalType	51	-	pedestal is a PedestalType which specifies the attachment point for PayloadType.
power_status	byte	1	Enumerated	powerStatus is a PowerStatusType which specifies the power state for PayloadType. The power state is specified as an enumeration: {power_on, power_standby, power_off, emergency_power} PowerStatusEnumTypeLDM: EMERGENCY_POWER = 0, POWER_OFF= 1, POWER_ON = 2, POWER_STANDBY = 3

system_operating_mode	byte	1	Enumerated	systemOperatingMode is a PayloadSystemOperatingModeType which specifies the operational state for PayloadType. The operational state is specified as an enumeration: {off, initializing, standby, active, calibrate, running_bit, on, inactive, deployed, stowed, caged} PayloadSystemOperatingModeEnumTypeLDM: ACTIVE = 0, CAGED = 1, CALIBRATE = 2, DEPLOYED = 3, FAULT = 4, INACTIVE = 5, INITIALIZING = 6, OFF = 7, ON = 8, OPERATE = 9, RUNNING_BIT = 10, SHUTDOWN = 11, STANDBY = 12, STARTUP = 13, STOWED = 14
temperature	float	4	m/s	temperature is a TemperatureType which specifies the temperature for PayloadType

Table 20 – PayloadDataRecorderType structure

Field	Type	Length	Units	Description
recording_location	TimeAddressCapabilityType	24	-	recordingLocation is an AddressCapabilityType which specifies the location of data recording for PayloadDataRecorderType.
recording_speed	CommsRateMegabitsPerSecondCapabilityType	28	-	recordingSpeed is a CommunicationRateCapabilityType which specifies the data recording speed for PayloadDataRecorderType.
recording_status	byte	1	Enumerated	recordingStatus is a RecordingStatusType which specifies the status of the data recording for PayloadDataRecorderType. The state is specified as an enumeration: {stop, ready, recording, play, seek} RecordingStatusEnumTypeLDM: PLAY = 0, READ_Y = 1, RECORDING = 2, SEEK = 3, STOP = 4
replay_location	TimeAddressCapabilityType	24	-	replayLocation is an AddressCapabilityType which specifies the location in the data recording for the replay of data for PayloadDataRecorderType.
replay_speed	CommsRateMegabitsPerSecondCapabilityType	28	-	replaySpeed is a CommunicationRateCapabilityType which specifies the data recording replay speed for PayloadDataRecorderType.
replay_status	byte	1	Enumerated	replayStatus is a ReplayStatusType which specifies the status of the data recording replay for PayloadDataRecorderType. The status is specified as an enumeration: {Stop, ActiveNotReady, ReadyPause, Reading}

				ReplayStatusEnumTypeLDM: ACTIVE_NOT_READY = 0, READING = 1, READY_PAUSE = 2, STOP = 3

Table 21 – SubsystemReportType structure

Field	Type	Length	Units	Description
is_detailed	bool	1	None	isDetailed is a BooleanType which specifies the state of the details of the subsystem status report for SubsystemStatusReport.
mission_communications_state	byte	1	Enumerated	missionCommunicationsState is a MissionCommunicationStateType that specifies the state of communications between a system and its controlling system for the SubsystemReportType. MissionCommsStateEnumTypeLDM: ACTIVE = 0, EMCON = 1, LOST = 2, PLANNED_LOST = 3
report_text	string	50?	None	reportText is a DescriptionType which specifies the text describing the subsystem status report for SubsystemStatusReport.
report_text_uri	string	50?	-	reportTextURI is a DescriptionType which specifies the URI context for the location of subsystem status report for SubsystemStatusReport.
source	byte	1	Enumerated	source is a SystemSourceType that specifies whether the source for the SubsystemReportType is an actual system data or an estimation service. SystemSourceEnumTypeLDM: ACTUAL = 0, ESTIMATED = 1
vehicle_specific_report_uri	string	50?	-	vehicleSpecificReportURI is a DescriptionType which specifies the URI context for the location of the vehicle specific report for SubsystemStatusReport

Table 22 – PedestalType structure

Field	Type	Length	Units	Description
attachment_orientation	Orientation3DType	12	-	attachmentOrientation is a OrientationCapabilityType which specifies the angular position or attitude for PedestalType.
attachment_status	byte	1	Enumerated	attachmentStatus is an AttachmentStatusType which specifies the attachment status for PedestalType. The status is specified through an enumeration: {loaded, discarded, released, jettisoned} DISCARDED = 0, JETTISONED = 1, LOADED = 2, RELEASED = 3
is_stabilized	bool	1	None	If the Pedestal provides the ability to actively cancel out host platform vibrations or rigid body motion (i.e. fix the Pedestal orientation in the inertial frame) then this element is TRUE.
number_of_axes	byte	1	None	Indicates the number of rotational degrees of freedom the Pedestal.
pointing_	Orientation3	12	-	pointingOrientation is a OrientationCapabilityType

orientation	DType			which specifies the pointing angular position or attitude for PedestalType.
pointing_orientation_velocity	OrientationVelocityType	12	-	pointingOrientationVelocity is a OrientationVelocityCapabilityType which specifies the rate at which the pointing angular positions or attitude is changing for PedestalType.
relative_position	Position3dPlatformXYZType	12	-	relativePosition is a PositionType specifying the payload's location relative to the navigational center of the vehicle. This is the offset of the payload relative to the vehicle position.

Table 23 – CommsRateMegabitsPerSecondCapabilityType structure

Field	Type	Length	Units	Description
comms_rate	float	4	Mbit/s	
comms_rate_domain	CommsRateMegabitsPerSecondSpecificationType	12	-	Defines the following: lower_limit (float num), step_size (float num) and upper_limit (float_num), all expressed in Mbit/s
size_set_point	CommsRateMegabitsPerSecondRequirementType	12	-	The same description goes here

Table 24 – Orientation3DType structure

Field	Type	Length	Units	Description
roll_x	float	4	rad	
pitch_y	float	4	rad	
yaw_z	float	4	rad	

Table 25 – OrientationVelocityType structure

Field	Type	Length	Units	Description
pitch_rate_y	float	4	rad/s	pitchRateY specifies the rate of change of the platform's rotation about the lateral axis (e.g. the axis parallel to the wings) in a locally level, XYZ coordinate system centered on the platform
roll_rate_x	float	4	rad/s	rollRateX specifies the rate of change of the platform's rotation about the longitudinal axis (e.g. the axis through the body of an aircraft from tail to nose) in a locally level, XYZ coordinate system centered on the platform
yaw_rate_z	float	4	rad/s	yawRateZ specifies the rate of change of the platform's rotation about the vertical axis (e.g. the axis from top to bottom through an aircraft) in a locally level, XYZ coordinate system centered on the platform

Table 26 – Position3dPlatformXYZType structure

Field	Type	Length	Units	Description
x_a	float	4	m	xA specifies the X-axis position which is in the forward (toward the nose) direction
y_a	float	4	m	yA specifies the Y-axis position which is in the right (starboard) direction
z_a	float	4	m	zA specifies the Z-axis position which is in the down (toward the centre of the Earth) direction

3.4.5 Mission

The Mission Editor module will be able to output and share planned routes with the UxVs. These routes will consist of lists of waypoints for the UxVs to follow and they will be translated to each UxV's specific protocol and sent to its Control Station. The UxV pilots will validate the mission before it is uploaded to the vehicles. Once the mission is validated (and eventually tweaked) it will be published back as an active route using the same message type.

Table 27 – Mission structure

Field	Type	Length	Units	Description
mission_id	int	4	None	Mission identifier
detailed	bool	1	None	Indicates whether the route is the result of detailed mission planning (TRUE), or is a simple stick route (FALSE).
first_path_in_route	RoutePathType	142183+	-	Indicates the unique ID of the first Path of the Route. The first Path of the Route would generally be of PRIMARY Path Type.
route_kind	byte	1	Enumerated	Type of route path. (LineSegmentEnumTypeLDM) GREAT_CIRCLE = 0, RHUMB = 1
route_path	RoutePathType	142183+	-	A series of path segments. A route can contain many paths. Some paths provide alternate routes and contingency routes which branch from the primary path. Other paths can be standalone, disconnected paths. Elements given here are not necessarily in mission/flight order; it is necessary to follow the linkages provided in NextPathSegment and/or ConditionalPathSegment to traverse the segments in mission/flight order.
DatastructureType	string	32	-	Describes the type of structure
SenderID	string	16	-	Describes the unique id of the data sender
Mission_Type	array			Contains the fields that will be used for the mission: Singles, Groups, Coverage

Table 28 – RoutePathType structure

Field	Type	Length	Units	Description
first_segment_in_path	SegmentType	225	-	Indicates the unique ID of the first Path Segment of the Path.
metrics	MetricsTyp e	34769	-	Indicates metrics related to the "cost" to complete the Path. When the Path is of EGRESS Path Type, this element should at a minimum specify fuel amount needed to egress, including any reserve desired for landing. When the Path is of INGRESS Path Type, this element should at a minimum specify the fuel amount that will be available at the end of the ingress Path. The allocators and route planners for mission operations will use these fuel amount to ensure that sufficient fuel remains upon the transition to the egress Path.
path_id	RoutePathT ype	142183+	-	Indicates the unique ID for this Path. Other Paths or Path Segments may branch to this path by referring to this ID.
path_kind	byte	1	Enumerated	Type for this path. PathKindEnumTypeLDM: ALTERNATE = 0, EGRESS = 1,

				EMERGENCY_LANDING = 2, HARD_DITCH = 3, INGRESS = 4, LANDING = 5, LOSS_OF_COMM = 6, PRIMARY = 7, RETURN_TO_BASE = 8, SOFT_DITCH = 9, TAKEOFF = 10
route_path_segment	RoutePathSegmentTyp e	107180 +	-	A PathSegment is defined from the previous path segment EndPoint (or current vehicle state if no previous path segment) to the current path segment EndPoint. Elements given here are not necessarily in mission/flight order; it is ecessary to follow the linkages provided in NextPathSegment and/or ConditionalPathSegment to traverse the segments in mission/flight order.
start_time	double	8		Indicates the start time of the first Path Segment of this Path. The start time of subsequent Path Segments is the end time of the previous Path Segment.
vehicle_id	string	16	-	Describes the unique id / name of each UxV
speed	int	4	m/s	Describes the speed for each UxV set by the user
Sensors	Array		-	Contains the sensor data for each UxV
time	int	4	None	Sensor time of deployment
sensor_type	string	16		Describes the type of the sensor
sensor_status	string	16		States the status of the sensor: Activated, Deactivated.
Scanning_density	int	4	None	Describes the radius that a group of UxV's is able to cover
Coverage_Cardinality	int	4	None	Declares how many vehicles participate in a Coverage mission

Table 29 – SegmentType structure

Field	Type	Length	Units	Description
foreign_segment_id	ForeignKeyType	60	-	foreignSegmentID is a ForeignKeyType that is a used to specify a foreign key for a segment.
line	LineSegmentRequirementType	112	-	line is a LineRequirementType from which route paths and routes are composed.
path_segment_locked	bool	1	None	pathSegmentLocked is a BooleanType that specifies that the vehicle path is locked and cannot be modified for the SegmentType.
path_segment_modified	bool	1	None	pathSegmentModified is a BooleanType that specifies that the vehicle path has been modified since creation for the SegmentType.
path_segment_source	byte	1	Enumerated	pathSegmentSource is a PathSegmentSourceType that specifies the source of the path as either operator defined or autorouted for the SegmentType. (PathSegmentSourceEnumTypeLDM) AUTO_ROUTED = 0,

				OPERATOR_DEFINED = 1
version	string	50?	None	version is a UniqueIDType that indicates a segment's version ID.
path_segment_id	int	2?	None	Pathe_segment_id is a UniqueIDType that indicates a segment's path ID.
waypoint_kind	byte	1	Enumerated	Type of Waypoint. WaypointKindEnumTypeLDM: APPROACH = 0, APPROACH_FINAL_POINT = 1, APPROACH_INITIAL_POINT = 2, HARD_DITCH = 3, NAV_ONLY = 4, PASSIVE = 5, RUNWAY_LIMIT = 6, RUNWAY_START = 7, RUNWAY_THRESHOLD = 8, TAKEOFF = 9, TAKEOFF_FINAL_POINT = 10, TAKEOFF_INITIAL_POINT = 11, TAXI = 12, TOUCHDOWN = 13
altitude	float		M	Describes the altitude set by the user
position	object		-	Contains the coordinates of each waypoint
latitude	float		None	latitude
longitude	float		None	longitude

3.4.6 MissionStatus/MissionChange

MissionStatus will provide messages with information about how the mission is progressing. It contains telemetry data, the current status of a mission and can detect failures and possible complications.

Table 30 – MissionStatus structure

Field	Type	Length	Units	Description
id	String	20		
time	String			
type	String		enumarated	Specifies the type of the message
latitude	double	8		Latitude of the vehicle (if provided)
longitude	double	8		Longitude of the vehicle (if provided)
orientation	float	4	degree	specifies the angular position of the UxV
speed	float		m/s	specifies the magnitude of the velocity
current_status	byte	1	enumarated	specifies the status of the given mission

3.4.7 TelemetryData

Telemetry provides the ARESIBO platform with real-time real or estimated position, speed and orientation of the different UxV assets, improving situational awareness. Each vehicle will transmit its own telemetry to its protocol translator, which will inject it into the ARESIBO data distribution system as a kinematic platform message. The following structure defines telemetry information, i.e., a collection of data related to the current pose, velocity and acceleration of an asset. Its definition has been made on the basis of the VehicleType data

model from UCS3.4, by adjusting the content as a new structure and by excluding also some of its non-applicable fields to our domain of interest.

Table 31 – TelemetryData structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	vehicle identification
location	Position3DType	24	None	specifies the position of the GroundVehicle in WGS_84 coordinates
location_error	Position3DCovarianceType	32	None	specifies the variance is the location attribute of the GroundVehicleType
velocity	VelocityType	12	m/s	specifies the rate of change in position in terms of ground speed components for GroundVehicleType
acceleration	AccelerationType	12	m/s	specifies the longitudinal, lateral, and vertical acceleration components for GroundVehicleType
attitude	AttitudeType	12	rad	specifies the angular positions for GroundVehicleType. Angular positions are normally specified as *roll*, *pitch*, and *yaw*
attitude_velocity	VelocityType	12	rad/s	specifies the pitch, roll, and yaw rate components for GroundVehicleType
attitude_acceleration	AccelerationType	12	rad/s²	specifies the pitch, roll, and yaw acceleration components for GroundVehicleType
speed	float	4	m/s	specifies the magnitude of the velocity (i.e., speed)
attitude_rate	float	4	rad/s	specifies the angular rate of change of heading for GroundVehicleType. Angular rate change of the projection of the longitudinal axis onto the horizontal plane, and that projection's bearing relative to true North
course	float	4	rad	specifies the direction of the platform's motion relative to true north. The measurement is stated in radians between 0 and 2 pi
heading	float	4	rad	represents a projection of the longitudinal axis of the platform onto the horizontal plane, relative to true north. The measurement is stated in radians between -pi and pi

Table 32 – Position3DCovarianceType structure

Field	Type	Length	Units	Description
covariance	double []	9N	m²	Covariance for Position3DType

Table 33 – InertialType structure

Field	Type	Length	Units	Description
domain_velocity	VelocityType	12	m/s	Vehicle's velocity in the current operating domain of the system.
ground_velocity	VelocityType	12	m/s	Vehicle's velocity in the current operating domain of the system.
orientation	OrientationType	12	rad	Euler Angle Sequence describing the orientation of the vehicle in the order yaw, pitch, roll. The angles are given in a locally level, North-East-Down coordinate system centred on the vehicle
position	Position3D_WGS84_Tuple	12	-	Physical location of the referenced item in geospatial coordinates
position_uncertainty	UncertaintyType	12	m/s	Uncertainty of the physical location of the referenced item

Table 34 – OrientationType structure

Field	Type	Length	Units	Description
pitch_y	float	4	rad	-
roll_x	float	4	rad	-
yaw_z	float	4	rad	-

3.4.8 AreaOfInterest

An area of interest is a geographic area and airspace that includes the objective of the operation or could impact on the successful conduct of that operation. The ARESIBO operators may transmit an area of interest through the ARESIBO central system to the UxV Control Station, which can be associated to a command (e.g. initiate a search pattern). On the other hand, the UxV operating crew can send an AOI back to the ARESIBO system, as an alert to signal e.g. interesting activity such as a detected target in that area.

Table 35 – AreaOfInterest structure

Field	Type	Length	Units	Description
overlay_id	int	4	None	Unique identifier used to create a new overlay or update an existing one
major_axis	float	4	m	majorAxis specifies the length of the rectangle's semi-major axis, which is drawn away from the referencePoint along the bearing defined by orientationOfMajorAxis. (rectangle)
minor_axis	float	4	m	minorAxis specifies the length of the rectangle's semi-minor axis, which is drawn away from the referencePoint along the bearing defined by (orientationOfMajorAxis minus 0.5 pi). (rectangle)
orientation_of_major_axis	float	4	rad	orientationOfMajorAxis specifies the orientation of the rectangle as the angle between the semi-major axis and true north. (rectangle)
reference_point	Position2D Type[]	16N	-	referencePoint specifies the location of the intersection of the two axes of the rectangle at a specific time.
altitude	float	4	m	Altitude
ui_type	byte	1	Enumerated	0:map – map overlay 1:video – video overlay MAP = 0, VIDEO = 1
type	byte	1	Enumerated	0:poi – Point of interest 1:polygon – polygon detection 2:rectangle – rectangle detection 3:path – mission planning waypoints POI = 0, POLYGON = 1, RECTANGLE = 2, PATH = 3
source	string	100	None	Origin of the overlay – software provider or algorithm
source_type	string	50	None	-
label	string	50	None	Mouse over tooltip header
label_image	byte[]	N	None	Mouse over tooltip image
data_type	byte	1	Enumerated	0:default – creates the overlay object using the Mission system default UI elements (color and/or image) 1:custom – uses the values from color or ui_resource fields to represent the overlay object 2:symbollib – uses a symbol library for the object representation

				DEFAULT = 0, CUSTOM = 1, SYMBOLLIB = 2
color	byte[]	3	None	RGB values
ui_resource	byte[]	N	None	Image byte[] value

3.4.9 AerialVehicleType

The AerialVehicleType structure defines details about vehicles operating in the air. This structure constitutes the same data model as the VehicleType proposed in the extended version of UCS3.4, since the latter standard is only UAV-oriented. The aforementioned structure incorporates details about the id, type, acceleration, altitude, attitude, fuel info, heading, speed, etc. More details are given in the following table.

Table 36 – AerialVehicleType structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	Vehicle identification
cucs_id	int	4	None	CUCS identification
acceleration	AccelerationType	12	-	specifies the longitudinal, lateral, and vertical acceleration components for AerialVehicleType
altimeter_pressure	float	4	-	realizes PressureType: an Entity used to describe a pressure, defined as follows. Pressure is the weight or force per unit area that is produced when something presses or pushes against something else
altitude	float	4	m	altitude in meters
altitude_type	byte	1	Enumerated	defines altitude type (reference frame) for all altitude related fields in this message. PRESSURE = 0, BARO = 1, AGL = 2, WGS_84 = 3
altitude_rate	float	4	m/s	specifies the estimated vertical velocity for AerialVehicleType
atc_call_sign	string	32	None	ATC Call Sign
attack_angle	float	4	rad	specifies the angle-of-attack for AerialVehicleType. Angle of attack specifies the angle between the chord line of the wing of a fixed-wing aircraft (or the plan of the main rotor) and the vector representing the relative motion between the aircraft and the atmosphere.
attitude	AttitudeType	12	-	specifies the angular positions for AerialVehicleType. Angular positions are normally specified as roll, pitch, and yaw.
attitude_acceleration	AccelerationType	12	-	specifies the pitch, roll, and yaw acceleration components for AerialVehicleType.
attitude_velocity	VelocityType	12	None	specifies the pitch, roll, and yaw rate components for AerialVehicleType.
bingo_fuel	float	4	kg	specifies the amount of fuel that would allow a safe return to base of intended landing for AerialVehicleType.
center_of_gravity	float	4	m	specifies how far the centre of gravity is from the nose of the vehicle. point at which this AerialVehicleType is balanced

course	float	4	rad	specifies the direction of the platform's motion relative to true north. The measurement is stated in radians between 0 and 2 pi.
course_heading_mode	byte	1	Enumerated	is a mutually exclusive set of values that defines the method of control of an air vehicle along a path (whether in a pre-planned or override mode). CONFIGURATION = 0, MANUAL_OVERRIDE = 1, MANUAL_OVERRIDE_UNTIL_POINT = 2
flap_angle	float	4	rad	specifies the angle between the chord of the flap and the chord of the aircraft's wing. The measurement is stated in radians between -pi and pi.
flight_mode	byte	1	Enumerated	mutually exclusive set of values that defines the air vehicle flight modes for use in modelling air vehicle performance in conjunction with Flight Performance Models. AERM = 0, AUTOLAND_ENGAGE = 1, AUTOLAND_WAVE_OFF = 2, AUTOPILOT_GENERAL = 3, AUTOPILOT_NAVAID_SLAVED = 4, AUTOPILOT_TERRAIN_AVOIDANCE = 5, BRANCH = 6, CONTIGENCY = 7, FLIGHT_DIRECTOR = 8, GOTO_IAF = 9, GROUND_CONTROLLED_STEARING = 10, JUMP_TO_WAYPOINT = 11, LAUNCH = 12, LOITER = 13, LOITER_NOW = 14, NO_MODE = 15, ON_ROUTE_LOITER = 16, RTB = 17, SLAVE_TO_SENSOR = 18, WAYPPOINT = 19
g_load_capacity	float	4	rad	specifies the magnitude of the rate of change of an object's velocity
heading	float	4	rad	represents a projection of the longitudinal axis of the platform onto the horizontal plane, relative to true north. The measurement is stated in radians between -pi and pi
launch_option	byte	1	Enumerated	mutually exclusive set of values that defines the action to be done as part of the vehicle launch operation. LAUNCH_ABORT = 0, LAUNCH_START = 1, TAXI_ABORT = 2, TAXI_START = 3
location_error	string ?	50?	?	specified the variance is the location attribute of the aerial vehicle
magnetic_variation	float	4	rad	Magnetic variation in rad
mass	float	4	kg	Mass is defined as Inertial mass which has been shown to be equivalent to active gravitational mass or passive gravitational mass.
maximum_air_speed	float	4	m/s	specifies the maximum not to exceed dash speed for AerialVehicleType.
optimum_cruis	float	4	m/s	specifies the optimum cruising speed for

e_airspeed				AerialVehicleType
optimum_endurance_airspeed	float	4	m/s	specifies the airspeed for minimum fuel flow (minimum power) for AerialVehicleType
radar_signature	string	50?	None	element that is defined as the radar signature of an object. It can be a UAV, Tank or any other such item
recovery_option	byte	1	Enumerated	describes submodes for the recovery phase of flight. RECOVERY_ABORT = 0, RECOVERY_RETURN_TO_BASE = 1, RECOVERY_START = 2
sideslip_angle	float	4	rad	specifies the angle between the actual direction of travel and the heading for AerialVehicleType. It is the direct result of movement in which a relative flow of air moves along the lateral axis, resulting in a sideways movement from a projected flight path, especially a downward slip toward the inside of a banked turn
speed	float	4	m/s	specifies the magnitude of the velocity (i.e. speed)
speed_brake_angle	float	4	rad	specifies the angle of the movable speed brake air foil for AerialVehicleType
speed_mode	byte	1	Enumerated	speedMode is an AirVehicleModePreferenceType which specifies the enumeration value of speed mode for AerialVehicleType INDICATED_AIRSPEED = 0, CALIBRATED_AIRSPEED = 1, TRUE_AIRSPEED = 2, GROUND_SPEED = 3, MACH = 4
identification_number	string	16	None	null terminated string with the tail number designated by the owning country's certifying agency
turn_rate	float	4	rad/s	specifies the angular rate of change of heading for AerialVehicleType. Angular rate change of the projection of the longitudinal axis onto the horizontal plane, and that projection's bearing relative to true North.
velocity	VelocityType	12	-	specifies the rate of change in position in terms of ground speed components for AerialVehicleType

Table 37 – AccelerationType structure

Field	Type	Length	Units	Description
roll_rate	float	4	rad/s	angular rotation rate of the vehicle about the longitudinal axis, + is clockwise looking from the rear of the UA towards the front
pitch_rate	float	4	rad/s	angular rotation rate of the vehicle longitudinal axis (tail to nose) relative to the local horizontal, + is u
turn_rate	float	4	rad/s	angular rate change of the projection of the longitudinal axis onto the horizontal plane, and that projection's bearing relative to true North

Table 38 – AttitudeType structure

Field	Type	Length	Units	Description
roll	float	4	rad	rotation of the vehicle about the longitudinal axis relative to the local horizontal plane, + is clockwise looking from the rear of the UA towards the front.
pitch	float	4	rad	angle of the vehicle longitudinal axis (tail to nose) relative to the local horizontal, + is up.
yaw	float	4	rad	heading projection of the longitudinal axis onto the horizontal

				plane, and that projection's bearing relative to true North.

Table 39 – VelocityType structure

Field	Type	Length	Units	Description
u_speed	float	4	m/s	speed component along true north vector in m/s
v_speed	float	4	m/s	speed component along true east vector in m/s.
w_speed	float	4	m/s	inertial vertical speed component pointing down in m/s.

Alternative data models for specific sub-concepts included in the aforementioned descriptions, could be the following:

⇒ The EnduranceType data model from UCS3.4 for providing relevant fuel information,
⇒ The BatteryType UCS3.4 data model for providing relevant battery information, and
⇒ The DataLinkStatusType UCS3.4 data model for providing relevant communications information.

Table 40 – EnduranceType structure

Field	Type	Length	Units	Description
duration	double	8	s	Estimated remaining time of operation with current fuel/charge/power.
footprint	EnduranceFootprintType	73	-	Indicates endurance in terms of maximum ground/surface distance that can be reached given the current System state.
fuel	float	4	kg	Measure of fuel in kg.
percent	float	4	%	Percent of fuel/charge/power remaining as compared to total capacity.

Table 41 – EnduranceFootprintType structure

Field	Type	Length	Units	Description
altitude	float	4	m	Indicates the Height Above Ellipsoid (HAE) reference for the footprint.
boundary	EnduranceFootprintBoundaryType	61	-	Indicates the boundary of the endurance footprint.
duration	double	8	s	Indicates estimated remaining time of operation with current fuel/charge/power plus the estimated time remaining until the endurance footprint shrinks to its smallest size.

Table 42 – EnduranceFootprintBoundaryType structure

Field	Type	Length	Units	Description
ellipse	PositionedEllipseType	36	-	Indicates the footprint boundary as a ground/surface ellipse, any part of which can be reached given the remaining endurance.
polygon	PolygonType	25	-	Indicates the footprint boundary as a ground/surface polygon, any part of which can be reached given the remaining endurance.

Table 43 – PositionedEllipseType structure

Field	Type	Length	Units	Description
center_point	Position2DTimeType	24	-	centerPoint specifies the center of the ellipse at a location on the surface of the Earth at a given point in time.
major_axis	float	4	m	majorAxis specifies the length of the longest diameter of the ellipse.
minor	float	4	m	minorAxis specifies the length of the shortest diameter

_axis				of the ellipse.
orientation_of_major_axis	float	4	rad	orientationOfMajorAxis specifies the true north bearing of the major axis of the ellipse. The measurement is stated in radians between 0 and 2 pi.

Table 44 – BatteryType structure

Field	Type	Length	Units	Description
connected_power_bus	PowerBusType	8	-	connectedPowerBus is a PowerBusType which defines the set of Power Buses to which this BatteryType is connected. The multiplicity indicates the number of currently connected PowerBusTypes.
energy_available	float	4	J	energyAvailable is an EnergyCapabilityType which defines the available energy from the BatteryType.
energy_usage_rate	float	4	W	energyUsageRate is a PowerCapabilityType which describes the rate of energy being used by the BatteryType.
temperature	float	4		temperature is a TemperatureCapabilityType which describes the temperature of the BatteryType.

Table 45 – PowerBusType structure

Field	Type	Length	Units	Description
current	float	4	A	current is a CurrentCapabilityType which describes the amount of electrical current flowing on the PowerBusType.
voltage	float	4	V	voltage is a VoltageCapabilityType which describes the amount of electrical potential energy available on the PowerBusType.

Table 46 – DataLinkConnectionType structure

Field	Type	Length	Units	Description
comm_address	string	50?	None	commAddress is a DescriptionType that specifies the recipient node address for the DataLinkConnectionType.
comm_protocol	string	50?	None	commProtocol is a DescriptionType that specifies the protocol used for the DataLinkConnectionType. Comm protocols include RS232 or FireWire for serial connections and TCP/IPV4 for IP. Additionally the protocol could be a composite to add an application layer such as FTP over TCP/IP.
comm_rate	float	4	Mbit/s	commRate is a CommsRateType that specifies the communication rate, usually baud or bps, for the DataLinkConnectionType.
comm_type	string	50?	None	commType is a DescriptionType that specifies the communication stack general type such as serial, IP, ATM, etc. for the DataLinkConnectionType. This field determines the available options for the commProtocol attribute.
data_encryption	string	50?	None	dataEncryption is a DescriptionType that describes the encryption method used during COMM for the DataLinkConnectionType.
data_format	string	50?	None	dataFormat is a DescriptionType that describes the format of the data being transferred across the data link for the DataLinkConnectionType.

3.4.10 UnderwaterVehicleType

The *UnderwaterVehicle* structure is a complete description of the system in terms of parameters such as position, orientation and velocities at a particular moment in time. The system position is given by a North-East-Down (NED) local tangent plane displacement (x, y,

z) relative to an absolute WGS-84 coordinate (latitude, longitude, height above ellipsoid). The symbols for position and attitude as well as linear and angular velocities were chosen according to SNAME's notation (1950). The body-fixed reference frame and Euler angles are depicted. The frequency of sending the message is one message per second while the vehicle is on the surface. During the time that the vehicle is submerged, no data is sent, when returning to the surface the data is sent again with the current position.

Table 47 – UnderwaterVehicleType structure

Field	Type	Length	Units	Description
name_vehicle	String		None	Name of vehicle
id_vehicle	int	4	None	Id of vehicle
lat	Float	8	rad	WGS-84 Latitude
lon	Float	8	rad	WGS-84 Longitude
height	Float	8	rad	Height above the WGS-84 ellipsoid
x	Float	4	m	The North offset of the North/East/Down field with respect to LLH.
y	Float	4	m	The East offset of the North/East/Down field with respect to LLH.
Z	Float	4	m	The Down offset of the North/East/Down field with respect to LLH.
phi	Float	4	rad	The phi Euler angle from the vehicle's attitude
theta	Float	4	rad	The theta Euler angle from the vehicle's attitude.
psi	Float	4	rad	The psi Euler angle from the vehicle's attitude.
u	Float	4	m/s	Body-fixed frame xx axis velocity component
v	Float	4	m/s	Body-fixed frame yy axis velocity component.
w	Float	4	m/s	Body-fixed frame zz axis velocity component.
depth	Float	4	m	Depth, in meters. To be used by underwater vehicles. Negative values denote invalid estimates.
alt	Float	4	m	Altitude, in meters. Negative values denote invalid estimates.

3.4.11 GroundVehicleType

The GroundVehicleType structure may define details about vehicles operating on the ground (GroundVehicle type). It constitutes a proposed extension of UCS3.4, as the latter is only UAV-oriented. The aforementioned structure incorporates details about the id, type, status (available/not-available, idle, deployed, etc.) and navigation mode, accompanied with telemetry data. More details are given in the following table.

Table 48 – GroundVehicleType structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	vehicle identification
identification_tag	string	50	None	null terminated string with the tail number designated by the owning country's certifying agency
telemetry	TelemetryData	136	None	specifies the current telemetry of the GroundVehicle
navigation_mode	byte	1	None	a mutually exclusive set of values that defines the current navigation mode of the ground vehicle. MANUAL = 0, REMOTE = 1, AUTONOMOUS = 2
status	Vehicle Status	?	None	type that describes current status of the asset, with generic and shared attributes with other Vehicle assets
radar_sign	string	50	None	element that is defined as the

ature				radar signature of an object. It can be a UAV, Tank or any other such item

3.4.12 WeatherData/EnvironmentalConditions

Weather information is crucial for the launch and recovery of UxV, and it must be provided by the ARESIBO system. Additionally, the UAVs can also provide local and real-time weather data to the ARESIBO system. An environmental related data model has been created as new, with the aim of providing the simulation entities with Meteorological and Oceanographic (METOC) data. This data can be used as an input to models from across the ARESIBO system. The model contains information obtained from publicly available sources about the air, water surface, water column, land and sea floor environments. The environment is divided into five gridded zones: the Air Column, the Water Surface, the Water Column, the Seabed and the Land. Each zone is broken into a series of 'data cubes' that contain all of the relevant environmental attributes. These cubes are referenced by row, column and, in the case of the air and water zones, by layer values.

Table 49 – VS_Environment structure

Field	Type	Length	Units	Description
num_rows	Unsigned32	4	-	number of rows of the grid
num_columns	Unsigned32	4	-	number of columns of the grid
NW_corner_latitude	Float64	8	rad	latitude of the Noth-West corner of the environmental grid
NW_corner_longitude	Float64	8	rad	longitude of the Noth-West corner of the environmental grid
SE_corner_latitude	Float64	8	rad	latitude of the South-East corner of the environmental grid
SE_corner_longitude	Float64	8	rad	longitude of the South-East corner of the environmental grid

Table 50 – AirColumn structure

Field	Type	Length	Units	Description
grid	VS_Environment	4	-	environmental grid
number_of_layer	Integer32	4	-	number of layer of the cube
lower_layer	Float64	8	m	height above mean sea level of the lowest layer of the cube
higher_layer	Float64Array	8	m	lowest above mean sea level of the lowest layer of the cube
sun_azimuth	Float64Array	Variable lengthArray	rad	
sun_elevation	Float64Array	Variable lengthArray	rad	
fog_cloud_density	Float64Array	Variable lengthArray	%	
wind_gust_direction	Float64Array	Variable lengthArray	rad	
wind_gust_duration	Float64Array	Variable lengthArray	s	
wind_gust_intensity	Float64Array	Variable lengthArray	m/s	
rainfall_rate	Float64Array	Variable lengthArray	mm/h	
relative_h	Float64Array	Variable	%	

umidity		lengthArray		
snowfall_rate	Float64Array	Variable lengthArray	mm/h	
solar_radiation	Float64Array	Variable lengthArray	w/m2	
sustained_wind_dir	Float64Array	Variable lengthArray	rad	
sustained_wind_speed	Float64Array	Variable lengthArray	m/s	
temperature	Float64Array	Variable lengthArray	C	
wind_shear	Float64Array	Variable lengthArray	m/s	
pressure	Float64Array	Variable lengthArray	hPa	

Table 51 – Bathymetry structure

Field	Type	Length	Units	Description
grid	VS_Environment	4	-	environmental grid
depth_values	Float64Array	Variable lengthArray	m	depth of the seabed wrt the water surface

Table 52 – Land structure

Field	Type	Length	Units	Description
grid	VS_Environment	4	-	Environmental grid
land_region_Id	Float32	Variable lengthArray	enum	
terrain_type	Float64	Variable lengthArray	enum	
snow_accumulation	Float64Array	Variable lengthArray	mm	
ice_accreation	Float64Array	Variable lengthArray	mm	

Table 53 – WaterColumn structure

Field	Type	Length	Units	Description
grid	VS_Environment	4	-	Environmental grid
number_of_layer	Integer32	4	-	Number of layer of the cube
shallower_layer	Integer32	4	-	
deeper_layer	Integer32	4	-	
water_current_intensity	Float64Array	Variable lengthArray	m/s	
water_current_direction	Float64Array	Variable lengthArray	rad	
salinity	Float64Array	Variable lengthArray	ppm	
temperature	Float64Array	Variable lengthArray	C	
transparency	Float64Array	Variable lengthArray	%	

Table 54 – SeaBed structure

Field	Type	Length	Units	Description
grid	VS_Environment	4	-	Environmental grid
breaking_wave_zone	Float32Array	4	-	Number of layer of the cube
wave_frequency	Float32Array	Variable lengthArray	Hz	
wave_height	Float32Array	Variable lengthArray	m	
wave_direction	Float32Array	Variable lengthArray	rad	

Table 55 – WeatherStateType structure

Field	Type	Length	Units	Description
weather	WeatherType		-	Current or forecasted weather for a defined area and period of time.

Table 56 – WeatherType structure

Field	Type	Length	Units	Description
barometric_pressure	PressureType			weather is a WeatherType that describes observed weather in a defined weather area for the WeatherObservationType.
cloud_cover	CloudCoverStateType			cloudCover is a CloudCoverStateType that defines the cloud cover for the WeatherType. The description of cloud cover includes the type of cloud cover e.g. clear, scattered, etc. and also the cloud ceiling and floor.
forecast_model	DescriptionType			forecastModel is a DescriptionType that defines the Meteorological Model used to derive weather information for the WeatherType.
humidity	HumidityType			humidity is a HumidityType that defines the relative humidity for a defined area for the WeatherType.
icing_severity	WeatherSeverityType			icingSeverity is a WeatherSeverityType that describes the extent of icing for the WeatherType. Icing severity includes none, light, moderate, severe, extreme, etc.
precipitation	PrecipitationStateType			precipitation is a PrecipitationStateType that defines the precipitation state for the WeatherType. The precipitation state includes the type, amplification and probability of precipitation.
remarks	DescriptionType			remarks is a DescriptionType that describes additional comments and details about the weather for the WeatherType. These remarks could be generated by the operator or the alert system.
temperature	TemperatureType			temperature is a TemperatureType that specifies the air temperature for the WeatherType.
thunderstorm_potential	SizeType			thunderstormPotential is a SizeType that defines the probability that there is a thunderstorm in a defined area for the WeatherType.
turbulence_severity	WeatherSeverityType			turbulenceSeverity is a WeatherSeverityType that defines the severity of air turbulence for the WeatherType. The turbulence severity can be characterized as none, light, moderate, severe, extreme, etc.

visibility	DistanceType			visibility is a DistanceType that defines the distance at which an object or light can be clearly discerned for the WeatherType.
weather_effects	WeatherEffectsType			weatherEffects is a WeatherEffectsType that describes the effects of weather on the surrounding environment (road state, sea state, terrain state, etc.) for the WeatherType.
wind_velocity	VelocityType			windVelocity is a VelocityType that defines the wind velocity for a defined area for the WeatherType.

3.4.13 Sensor

This type of messages will be utilized for sensor reporting. Similar to the *EOIRStatusRptType* structure of the UCS 3.4, these structures will be used to track the current status of the sensors and retrieve useful information.

Table 57 – EOIRStatusRptType structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	Vehicle identification
payload_id	int	4	None	Payload identification
built_in_test_status	byte	1	Enumerated	The status of the Built-In Test for the EO/IR sensor. BIT_FAILED = 0, BIT_PASSED = 1, BIT_SUSPENDED = 2, OFF_ABORT = 3, RUNNING_BIT = 4
field_of_view_azimuth	float	4	rad	The current azimuth/yaw component of the sensor's field of view.
field_of_view_elevation	float	4	rad	The current elevation/pitch component of the sensor's field of view
focus_automation	byte	1	Enumerated	The current focus adjustment mode of operation for the EO/IR payload sensor. AUTOMATIC = 0, MANUAL = 1, SEMI_AUTOMATIC = 2
image_center_location	Position3DType	20	None	The current coordinates for the center of the entity at which
image_output_state	byte	1	Enumerated	The current output channels of an imaging sensor system. NONE = 0, EO = 1, IR = 2, BOTH = 3, PAYLOAD_SPECIFIC = 4
pointing_mode	byte	1	Enumerated	The current pointing mode for the EO/IR sensor. NIL = 0, ANGLE_RELATIVE_TO_UA = 1, SLEWING_RATE_RELATIVE_TO_UA = 2, SLEWING_RATE_RELATIVE_TO_INERTIAL = 3, LAT_LONG_SLAVED = 4, TARGET_SLAVED = 5, STOW = 6, LINE_SEARCH_START_LOCATION = 7,

				LINE_SEARCH_END_LOCATION = 8
pointing_orientation	Orientation3DType	12	-	The current pointing direction (roll, pitch and yaw) for the sensor, relative to the airframe.
power_status	byte	1	Enumerated	The current power status (on, off, standby, etc.) of the EO/IR sensor. POWER_OFF = 0, POWER_ON = 1, EMERGENCY_POWER = 2, POWER_STANDBY = 3

Table 58 – PayloadSteeringReportType structure

Field	Type	Length	Units	Description
vehicle_id	int	4	None	Vehicle identification
payload_id	int	4	None	Payload identification
field_of_view_azimuth	float	4	rad	The current azimuth/yaw component of the sensor's field of view.
field_of_view_elevation	float	4	rad	The current elevation/pitch component of the sensor's field of view
image_center_location	Position3DType	20	None	The current coordinates for the center of the entity at which
pedestal_pointing_orientation	Orientation3DType	20	None	The current point direction (roll, pitch and yaw) for the pedestal, relative to the airframe
pedestal_pointing_orientation_velocity	OrientationVelocity3DType	20	None	The current rate of change for each axis of the pedestal orientation
pointing_mode	byte	1	Enumerated	The current pointing mode for the EO/IR sensor. NIL = 0, ANGLE_RELATIVE_TO_UA = 1, SLEWING_RATE_RELATIVE_TO_UA = 2, SLEWING_RATE_RELATIVE_TO_INERTIAL = 3, LAT_LONG_SLAVED = 4, TARGET_SLAVED = 5, STOW = 6, LINE_SEARCH_START_LOCATION = 7, LINE_SEARCH_END_LOCATION = 8
zoom_direction	byte	1	Enumaretd	The current zoom direction (in, out, none) for the imaging sensor. NO_ZOOM = 0, ZOOM_IN = 1, ZOOM_OUT = 2

3.4.14 XR (AR/MR/VR) device

This field describes the data that interacts with the XR devices. The input from the system to the devices may include external data (sensor data etc.), textual information, images, audios and videos. While the output from devices to the system may include strings, images, audios and videos.

Table 59 – ContentPlaceHolder structure

Field	Type	Length	Units	Description
id	String	20		Content placeholder id
position	String	24		Content placeholder position

rotation	String	24		Content placeholder rotation

Table 60 – UserProfile structure

Field	Type	Length	Units	Description
id	String	20		User id
role	String	20		User role

Table 61 – DeviceProfile structure

Field	Type	Length	Units	Description
id	Long	20		
change_date	String			Device information changed date
changed_by	String	40		Device information changed User
client_id	String	64		The device id
client_name	String	64		The device name
create_date	String			The device enrolment date
current_user_id	Long	20		The user id
device_type	String	200		The type of device
ip_address	String	32		Device IP address
status	String	24		Device status (connected/disconnected)
wifi	String	40		Device wifi status
recent_user_id	Long	20		The user id of last login
current_build_id	String	64		The device build information
bluetooth_mac	String	40		The device Bluetooth MAC
device_category	String	140		
os	String	200		The device OS
serial_number	String	200		Serial number of the device
wifi_mac	String	40		The device WiFi MAC
domain_id	Long	20		
sdk	int	11		

3.4.15 VideoDetection

The Visual detection component is responsible for providing a set of *bounding boxes* of the targets detected in video streams. The structure will include the *confidence scores* for each object that exists inside the image frame and its bounding box (i.e., where this object is located inside the image). As visual detection details are not part of the UCS model; thus, they can be considered as an extension field.

More specifically, at the top level of the data structure, some generic fields are suggested to be included related to the input data fed to the visual detection component. The indicated fields are the *source URL*, the *sender id* for the identification of the asset that provides the visual information and the *geolocation* of the asset, the *pixel size* of the frame, and the *timestamp* when the detection occurred. A set of *targets* will be also listed based on the number of targets detected within the frame. The information per target will be the following: the type of the *detected object* (car, person, boat, etc.), an *id* for the identification of the object per frame, the *confidence score* of the detection, the *bounding box* of the detection in pixel coordinates and a *path* to a local server where the frame of the detection is stored.

Table 62 – VideoDetection structure

Field	Type	Length	Units	Description

timestamp	long		None	The timestamp of the detection in Zulu time format
sender_id	int	4	None	Vehicle identification
source	string	50	None	RTMP or RTSP url
width	int	4	None	Frame width in pixels size
height	int	4	None	Frame height in pixels size
latitude	double	4	None	Latitude of the vehicle (if provided)
longitude	double	4	None	Longitude of the vehicle (if provided)
target_class	string	10	None	Class type of the detected object
target_object_id	int	4	None	Unique object id per detection
target_confidence	float	4	None	The confidence score of the detection
target_im_analysed	string	50	None	Path on a local server with a snapshot of the 1st frame of the detection
bbox_top	int	4	None	This is the smallest pixel value of the box along the x axis
bbox_left	int	4	None	This is the smallest pixel value of the box along the y axis
bbox_width	int	4	None	The width of the box in pixels along the x-axis
bbox_height	int	4	None	The height of the box in pixels along the y-axis

3.4.16 AlertType

According to the ARESIBO GA, there are different components that target to provide alerts (messages) to the end users, within the operation of the ARESIBO system. The exact location, the type, a short description and various other fields are included in the data structure of this type of messages. For example, the *sensor fusion engine* as well as the *semantic representation and reasoning module* will provide real-time alerts regarding a specific detection (based on sensor measurements) or a current severe condition for which information should be disseminated directly to the interested parties. As an additional example, the risk analysis module will communicate risk predictions to the end users, regarding the progress of a monitored situation, recommendations to mitigate a risk, etc. The concept of *alerts* is part of the extended version of the UCS3.4 (named AlertType). However, within the context of the ARESIBO the structure will be extended with one additional field, named 'category', which will represent the origin of the alert (incident/sensor fusion/risk).

Table 63 – AlertType structure

Field	Type	Length	Units	Description
alert_id	int	4	None	alert identification
vehicle_id	int	4	None	vehicle identification
subsystem_id	byte	1	Enumerated	identifier associated with the subsystem for which status information is being reported. ENGINE = 0, MECHANICAL = 1, ELECTRICAL = 2, COMMS = 3, PROPULSION_ENERGY = 4, NAVIGATION = 5, PAYLOAD = 6, RECOVERY_SYSTEM = 7, ENVIRONMENTAL_CONTROL_SYSTEM = 8, VSM_STATUS = 9, VDT = 10, CDT = 11, RESERVED_1 = 12, RESERVED_2 = 13, RESERVED_3 = 14, RESERVED_4 = 15, RESERVED_5 = 16,

				RESERVED_6 = 17, RESERVED_7 = 18, RESERVED_8 = 19, VSM_SPECIFIC_1 = 20, VSM_SPECIFIC_2 = 21, VSM_SPECIFIC_3 = 22, VSM_SPECIFIC_4 = 23, VSM_SPECIFIC_5 = 24, VSM_SPECIFIC_6 = 25, VSM_SPECIFIC_7 = 26, VSM_SPECIFIC_8 = 27, VSM_SPECIFIC_9 = 28, VSM_SPECIFIC_10 = 29, VSM_SPECIFIC_11 = 30, VSM_SPECIFIC_12 = 31
alert	byte	1	Enumerated	alert is an AlertKindType which specifies the enumeration value for the type of non-normal subsystem condition for AlertType. ACKNOWLEDGEABLE = 0, ACKNOWLEDGEABLE_CLEARABLE = 1, CLEAR = 3, CLEARABLE = 4, FIXED_TIME = 5, NOT_CLEARABLE = 6
alert_end_time	double	8	s	alertEndTime is a TimeType which specifies the date and time value relative to the end of the alert for the subsystem alert for AlertType.
alert_group	byte	1	Enumerated	alertGroup is an AlertGroupType which specifies the enumeration value of the group category for the subsystem alert for AlertType AIR_COLLISION = 0, AV_PLATFORM = 1, ENGINEERING = 2, HAZARDOUS_AREA = 3, MAINTENANCE = 4, PAYLOAD = 5, RESTRICTED_AREAD = 6, SYSTEM = 7
alert_level	byte	1	Enumerated	alertLevel is an AlertLevelType which specifies the enumeration value indicating the alert level for the subsystem for AlertType. ADVISORY = 0, CAUTION = 1, CLEARED = 2, WARNING = 3
alert_notification	byte	1	Enumerated	alertNotification is a NotificationType which specifies the enumeration value indicating the reason for receiving the alert for AlertType. The notification will be in response to a specific request or as a result of subscription. SPECIAL_REQUEST = 0, SUBSCRIBED_TO_REQUEST = 1
alert_priority	UInt32	4	None	alertPriority is an OrderType which specifies the priority level of the AlertType
alert_start_time	double	8	s	alertStartTime is a TimeType which specifies the date and time value relative to the start of the alert for the subsystem alert for AlertType
alert_status	byte	1	Enumerated	alertStatus is an AlertStatusType which specifies

				the enumeration value indicating the status of posted alert information for the subsystem for AlertType. ALERT_ACTIVE = 0, ALERT_ACTIVE_ACKNOWLEDGED = 1
alert_text	string	80	None	alertText is a DescriptionType which specifies the text for describing the alert for AlertType
latitude	double	4	None	latitude of the alert
longitude	double	4	None	longitude of the alert
altitude	float	4	m	altitude of the alert
category	byte	1	Enumerated	category is used to indicate the source module that the alert was created from. INCIDENT = 0 (incident detection) SFE = 1 (sensor fusion engine) RISK = 2 (risk analysis module)

3.4.17 Position/Geospatial data

In need to describe the location (geographical position) of an entity/resource (asset, human, etc.) or an incident (detection, activity, condition, etc.) we utilise from the UCS3.4 the relevant data structures, i.e. the Position2DType and the Position3DType, as described below.

Table 64 – Position2DType structure

Field	Type	Length	Units	Description
latitude	double	8	rad	Latitude value describing the current position of the involved entity in WGS84 coordinates format
longitude	double	8	rad	Longitude value describing the current position of the involved entity in WGS84 coordinates format

Table 65 – Position3DType structure

Field	Type	Length	Units	Description
height	float	4	m	
position	Position2DType	16	-	

3.4.18 Decision Support/Action

Information handled by the Decision Support (DS) module will be based, amongst the other, on existing standard formats used to share data in the emergency/security/safety fields. These standard formats, whose carried information will be consumed and visualized within the interconnected command and control systems, include:

- Common Alerting Protocol (CAP) (Oasis 2010)
- Emergency Data Exchange Language-Resource Messaging (EDXL-RM) (Oasis 2008)

The OASIS CAP[4] protocol is an XML data format for exchanging both events' related information and alerts over all kinds of media. It contains an alert block with generic event/alert information, multiple info blocks with multilingual information to describe events or alerts details, multiple resource blocks for attaching multimedia content and, in general, additional resources and multiple area blocks to define geographic features, such as events' location or the area which a given alert refers to.
Building on the XML schema prescribed by the CAP specifications, ARESIBO will define personalized profiles for the exchanged CAP messages, according to the specific communication contexts and needs.

[4] http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.pdf

A CAP Profile is a customization of the base Common Alerting Protocol, which will be used in the project to provide the ability to collect and relay information for the foreseen types of events and alerts from/to a variety of interconnected systems.
The definition of a CAP Profile for use in all countries involved in the pilots represent an improvement over the current situation, since it will allow the creation of messages according to specific needs of all services involved in border security operations. Figure 7 shows the elements of the CAP structure. Information targeted to specific applications' needs according to the ARESIBO CAP profile, will be carried through custom, pre-defined *alert.info.parameter* fields.
The EDXL-RM[5] protocol defines an XML schema to facilitate sharing of information on resources. It provides several data structures and a complete mechanism to request, offer and describe employed resources, specifically to realize the following actions on resources:

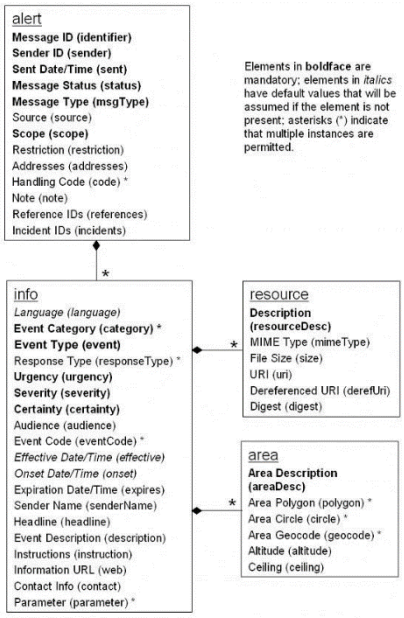

Figure 7. CAP message structure

⇒ RequestResource
⇒ ResponseToRequestResource
⇒ CommitResource
⇒ RequestInformation
⇒ ResponseToRequestInformation
⇒ OfferUnsolicitedResource
⇒ ReleaseResource
⇒ RequestResourceDeploymentStatus
⇒ ReportResourceDeploymentStatus
⇒ RequestExtendedDeploymentDuration
⇒ ResponseToRequestExtendedDeploymentDuration

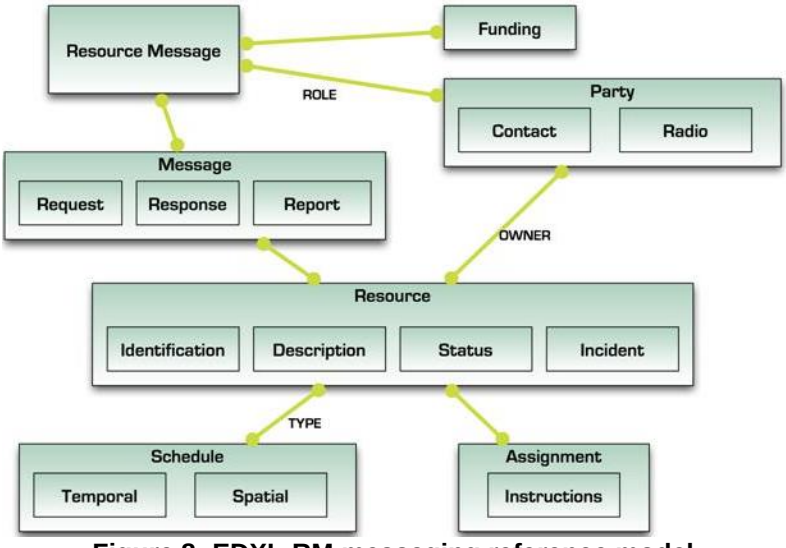

Figure 8: EDXL-RM messaging reference model.

[5] http://docs.oasis-open.org/emergency/edxl-rm/v1.0/EDXL-RM-SPEC-V1.0.pdf

Table 66 shows a basic event/alert information structure, from the mapping between the most relevant information needed for describing alerts and events, and some of the fields foreseen by the CAP standard:

Table 66 – BasicEventAlertInformation structure

Field	Type	Length	Units	Description
msg_id	String			Unique CAP message identifier
msg_timestamp	String			Date/time when the message is created
msg_sender	String			Message sender
msg_status				Message status (e.g. Actual)
evt_category	String			Event/Alert category (e.g. Security, Safety)
msg_type	String			Message type (e.g. Alert, Update)
evt_type	String			Event/Alert type
evt_desc	String			Event/Alert description
evt_location	String WGS84			Event location or Alerting area (textual description). Event location or Alerting area (WGS84 coordinates of points or polygons)
evt_severity	String			Event/Alert severity (e.g. Severe, Extreme, Moderate)
evt_urgency	String			Event/Alert urgency (e.g. Immediate, Expected)
evt_certainty	String			Event/Alert certainty (e.g. Observed, Likely)
msg_recipients	String			List of intended recipients
evt_resources	String			Additional resources associated with the Event/Alert (e.g. external links, media)

Table 67 shows a basic structure for a *ResourceDeployStatus* message, from the mapping between the most relevant information needed for describing resources deploy status, and some of the fields foreseen by the corresponding EDXL-RM message:

Table 67 – ReportResourceDeployStatus structure

Field	Type	Length	Units	Description
msg_id	String			Unique message identifier
msg_timestamp	String			Date/time when the message is created
msg_content_type	String			Message content type (ReportResourceDeployStatus in the considered case)
msg_sender	String			Message sender
res_id	String			Resource identifier in the system
res_name	String			Resource description (text)
res_type	String			Resource type (e.g. border patrol)
res_owner	String			Resource owner contact info
res_deploy_status	String			Resource deployment status (e.g. In transit)
res_availability	String			
res_quantity	String			Measurable quantity of resources of the given type
res_schedule_info	String			Schedule type information (e.g. departure, arrival)
res_schedule_time	String			Schedule time (e.g. actual foreseen departure or arrival time)
res_schedule_location	String			Resource schedule location (e.g. Russia/Finland customs clearance)

3.4.19 VoiceStream

The voice streams between the on-field units and the C2 centre will be exchanged using the IP protocol. Therefore, the digital information is packetized and encapsulated into IP packets

and sent through a VPN tunnel established between the communication hub and the C2 centre.

Table 68 – IPPacketVoice structure

Field	Type	Length	Units	Description
IP header		20-60	Bytes	The IPv4 header is variable in size due to the optional 14th field (options)
IP payload		0-65,535	Bytes	The IPv4 payload is variable. It is notable due to the MTU of the network being 1500 Bytes, any packet larger than that value will be fragmented into packets smaller than 1500 Bytes.

3.4.20 VideoStream

The real time video streams will be encapsulated into IP packets and sent through a VPN tunnel established between the communication hub and the C2 centre. All the packets maintain their format which the communication network doesn't modify.

Table 69 – IPPacketVideo structure

Field	Type	Length	Units	Description
IP header		20-60	Bytes	The IPv4 header is variable in size due to the optional 14th field (options)
IP payload		0-65,535	Bytes	The IPv4 payload is variable. It is notable due to the MTU of the network being 1500 Bytes, any packet larger than that value will be fragmented into packets smaller than 1500 Bytes.

4 Definition of the ARESIBO Knowledge Base (KB)

The following section presents the first iteration of the ARESIBO ontology, also referred as "the ARESIBO Knowledge Base (KB)". The ARESIBO KB will serve as a knowledge representation model for semantically representing notions pertinent to incidents, resources and tasks that are reported and handled within the context of the ARESIBO system. More specifically, the KB will receive input from the different multimodal sensors "attached" to the operational assets, in order to process heterogeneous data and detection results from lower levels of implementation, with a target aim to describe semantically the defined events/incidents. Coupling the "sensed" data for the available resources and tasks with contextual information will increase the situational awareness of the operator/end-user of the system. The different ARESIBO components that are linked to the reporting, analysis and transmission of data from sensors interact with the KB via the Knowledge Base Service (KBS). The latter can be conceived as the interface to the ontology, which semantically integrates the different sourced data into the ontology. It also receives output from the semantic reasoning process (inference) running on top of the KB and forwards the inferred, high-level knowledge back to other interested system modules, like for example to the ARESIBO Decision Support or to the Dashboard. The interaction between the KB and the KBS is established with the use of proper ontology-based queries (SPARQL/SPIN), which can insert/delete/fetch relevant data from the components to the KB and vice versa. The communication between the KB, the KBS and the different components and sensors is depicted in Figure 9.

In the following subsections we specify details about the technologies utilised (Section 4.1), the ontology engineering process (Section 4.2). In addition, the requirements of the ARESIBO KB (Section 4.3.1) are specified while an extensive overview of the existing ontologies is provided and describe different parts of the context of our domain of interest (Section 4.3.2). The analysis is continued with details about the implementation (Section 4.3.3) and conceptualisation (Section 4.3.4) of the ARESIBO KB, and we evaluate its key

metrics (Section 4.3.5). Finally, we frame the first iteration of the ontology reasoning processes, by demonstrating specific use cases (Section 4.4).

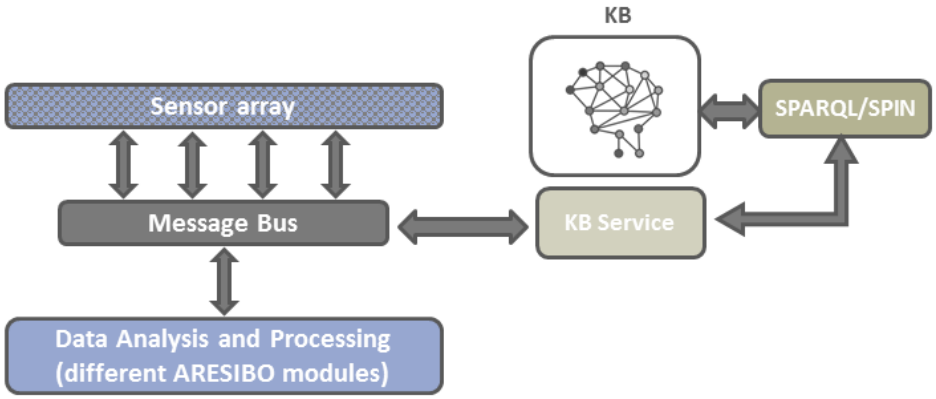

Figure 9: Interaction of KB, KBS and the different ARESIBO component and sensors

4.1 Ontologies and Semantic Web

The **Semantic Web** is "*a web of data that can be processed directly and indirectly by machines*" (Berners-Lee et al., 2001). It is an extension of the World Wide Web (WWW), in which web resources are augmented with semantics describing their intended meaning in a formal, machine-understandable way. The term was coined by Tim Berners-Lee, the inventor of WWW and director of the World Wide Web Consortium (W3C), which oversees the development of proposed Semantic Web standards. The standards proposed by W3C promote common data formats and exchange protocols on the Web. The Semantic Web is thus regarded as an integrator across different content, information applications, and systems. **Ontologies** play a key role in the Semantic Web, providing the machine-interpretable semantic vocabulary and serving as the knowledge representation and exchange vehicle. The **Web Ontology Language** (**OWL**) has emerged as the official W3C recommendation for creating and sharing ontologies on the Web (Bechhofer, 2009).

4.2 Ontology Engineering Process

There are several ontology engineering methodologies existing in literature, describing the different approaches in the design and implementation of ontological frameworks. In Table 70, we summarize the thorough comparison conducted for the most established methodologies, including Sensus (Swartout et al., 1997), KACTUS (Bernaras et al., 1996), DOGMA (Jarrar and Meersman, 2008), METHONTOLOGY (Fernandez et al., 1997), DILIGENT (Pinto et al., 2004), On-To-Knowledge (Sure et al., 2004), Cyc (Lenat and Guha, 1989), Unified (Uschold, 1995), Grüninger and Fox (Grüninger and Fox, 1994), and Neon (Suárez-Figueroa et al., 2009). The aforementioned methods are compared on the basis of a specific set of characteristics:

- ⇒ **Well-documented** – this term shows the depth and details provided for each process and guideline of the methodology,
- ⇒ **Reusability** – this characteristic shows whether the reuse/reengineering of existing ontologies is supported by the methodology,
- ⇒ **Dynamic/Adaptive** – this value defines the level of adaptability to various stages of development,
- ⇒ **Structured representation** – this parameter shows if the methodology incorporates a structured description of the ontology requirements.

Table 70 – Comparison of Ontology Engineering Methods

Methodology	Well-	Reusability	Dynamic/	Structured representation

	documented		Adaptive	
Sensus	Medium	Yes	Low	No
KACTUS	Low	Yes	Low	No
DOGMA	High	No	Low	Tuples[6]
METHONTOLOGY	Medium	Yes	Low	No
DILIGENT	Medium	No	Low	No
On-To-Knowledge	High	No	Low	No
Cyc	Medium	Yes	Low	No
Unified	Low	No	Low	No
Grüninger and Fox	High	No	Medium	CQs[7]
NeOn	High	Yes	High	ORSD[8]

While all examined methodologies presented interesting perspectives towards building ontologies, either from scratch or by additionally inheriting existing ones, the NeOn methodology outstands all others since it adequately covers all the aforementioned aspects in the most efficient way. The NeOn methodology is a scenario-based process which guides the ontology engineer to define efficiently the requirements and characteristics of the ontology, it considers the existence of multiple ontologies in ontology networks, and thus it supports the reuse/reengineering of knowledge resources. It consists of the following components:

- ⇒ **The NeOn Glossary** - a well-established glossary that includes 59 predefined processes and activities. Its purpose is to provide a standard vocabulary, created by ontology experts that can be used for well-described and structured processes.
- ⇒ **Scenarios for building ontologies and ontology networks** - unlike other methodologies, NeOn approaches a variety of scenarios for ontology engineering, while each scenario is decomposed into different processes or activities.
- ⇒ **Two ontology network life cycle models** - these models, named the Waterfall Model and the Iterative/Incremental Model indicate how to establish the development processes and activities.
- ⇒ **A set of methodological guidelines for processes and activities** - These are specific guidelines in order to fulfil the activities and processes mentioned in the NeOn Glossary.

In the current task, we adopt the NeOn methodology for specifying the requirements of the ARESIBO ontology, the details of which is presented in the following subsection.

4.3 The ARESIBO Ontology

4.3.1 Specification of Ontology Requirements

The key aim of the ARESIBO ontology is to semantically represent all notions that are pertinent to the project, serving as the model for semantically integrating information coming from the various sensors and analysis components of the system. In a sense, we are primarily interested in processing the heterogeneous content and detection results sourced from lower levels of implementation to higher levels of interpretation, by semantically

[6] In DOGMA framework, a tuple is a description of conceptual relations in the form <γ: Term1, Role, InvRole, Term2>, where Term1 and Term2 are linguistic terms, γ is a context identifier, and Role/InvRole are lexicalisations of the paired roles in any binary relationship; for each pair (γ, Term) is assumed to refer to a uniquely identifiable concept (Jarrar and Meersman, 2008).

[7] Competency questions (CQs) constitute an indicative (non-exhaustive) list of questions that the ontology should be competent to answer (Grüninger and Fox, 1995)

[8] The Ontology Requirements Specification Document (ORSD) is a structured document that captures the aims and scope, the main uses, the targeted end-users, as well as the functional and non-functional requirements of the ontology to be implemented.

describing the required events/incidents and by facilitating the information/assessment of hazardous situations. By enriching data for the available resources and tasks, with contextual information, we target to increase the situational awareness of the operator for the current condition of the system and potentially augment their navigation and communication capabilities. In other words, the ARESIBO ontology will serve as the bridge between visually identified concepts (detections) and communicated content (alerts) to the end user.

Driven by the aforementioned objectives, we describe the process of designing and implementing the first iteration of the ARESIBO ontology. Starting from its purpose and scope, the ARESIBO ontology aims at fulfilling the needs for:

(i) *Data integration at semantic level*. The ARESIBO semantic model will be built as a network of interconnected ontologies that will act as the "glue" in order to link heterogeneous concepts and individual component-level data models. The core ontology will be the common representation framework of the project for the semantic modelling and integration of information stemming from several modules, sensors and other external resources (e.g., legacy systems, C2) that will be linked to ARESIBO system. If needed, a set of domain specific ontologies will be linked to the core ontology to specialise concepts and terminology whenever this is needed. For instance, the *CIRAM ontology*[9] will extend the core ARESIBO ontology to represent risk concepts that will describe detected/predicted risks and potential threats in border surveillance operations.

(ii) *System intelligence.* A set of reasoning tasks will be supported over the knowledge base that will be built in order to facilitate a wide range of decision-making processes and system functionalities. The ARESIBO KB will be capable of integrating, combining and, finally, inferring new knowledge based on existing data that has been fed to the system. In particular, ontological models will be interconnected with a querying and a reasoning module (e.g., a SPARQL endpoint for semantic queries and a DL-reasoner, respectively). Models and reasoning rules will be expressed in different Semantic Web languages, such as RDF(S), OWL(2), SWRL, SPARQL and SPIN to balance between the expressiveness of the knowledge representation methodologies and the performance (i.e., tractability) of query answering tasks (e.g., instance checking, classification, consistency checking).

Within the project, the ARESIBO KB will be fed by other modules (e.g., visual recognition, sensor fusion, risk analysis) and will perform semantic reasoning to deduce new knowledge. In this context, information such as **real-time alerts**, **detections** or even **predictions** will be combined in order the system to **infer new incidents** (e.g. border crossing) or to **extract hidden knowledge** from a sequence of incidents (e.g., speedboat approaching the shore). A set of semantic reasoning scenarios that the ARESIBO ontology can address related to the considered PUCs of the project is included in the following table.

Table 71 – Potential semantic reasoning scenarios based on the project's PUCs.

Semantic reasoning scenario	Relevant PUCs
Infer land border trespassing based on trespasser's location.	PUC1: Land border Tresspassing
Automated human detection based on visual recognition.	PUC1: Land border Tresspassing
Estimate target's speed and moving direction.	PUC2: Smuggling of goods
Infer target's location based on sensors measurements.	PUC3: Human Trafficking
Keep track of the drifting location of Floating Unwanted Packages (FUPs).	PUC4: Drug Trafficking
Assess whether a target is a threat or not: this will depend on detected	All PUCs

[9] This work belongs to T4.6 and will be delivered in D4.6.

objects and activities (e.g. firearms, potentially illegal activities like smuggling etc.).	
Assess the severity of a risk; CIRAM defines specific levels of risk severity.	All PUCs

4.3.2 Reuse of Existing Resources

Ontology reuse is the process of adopting and efficiently integrating available ontological schemas when developing a new ontology. It is generally considered as a key factor in developing cost-effective, high-quality and interoperable ontologies, since (a) it avoids "re-inventing the wheel", i.e., rebuilding existing ontologies and resources from scratch, and (b) it takes advantage of already formalised ways of representing specific entities in domains of interest. Such domains can be either (a) *general* (upper level ontologies), regarding abstract/common concepts, or (b) domain specific, involving more concrete conceptualisations of the abstract notions that apply to specific use cases and fields of interest. Since the project's domain can be substantially wide, and the user requirements specification is still an ongoing process, we had to focus on the most relevant (at that point) domains of interest, on the basis of the existing requirements, and the available use cases (PUCs) described in the GA of the project. As a result, we have discriminated in the following subsections a list of ontologies that can potentially be adopted as concepts in the ARESIBO ontology and be aligned (extended) within its scope. A structured representation of the relations between the ARESIBO KB and the existing knowledge is depicted in Figure 10.

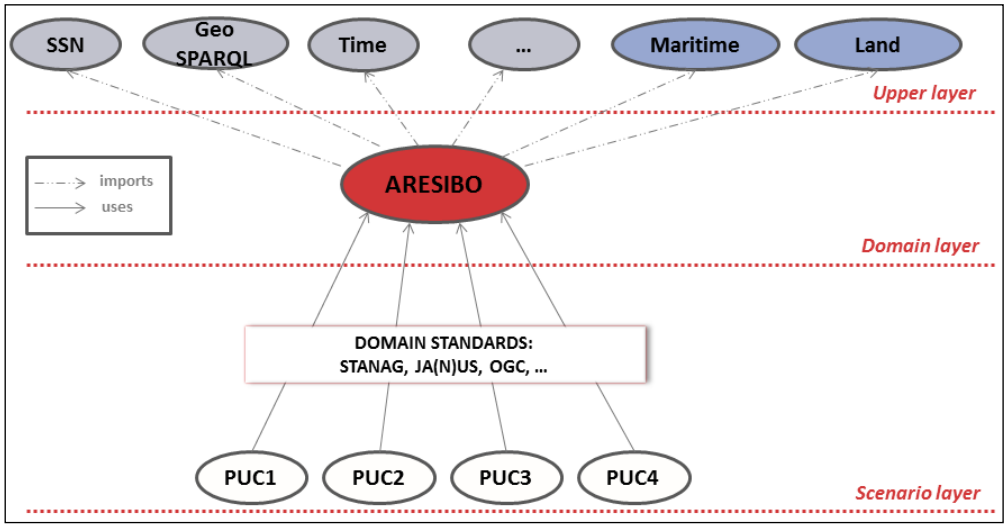

Figure 10: Structuring the process of adopting domain ontologies (blue ellipse) and upper level ontologies (grey ellipse) within the context of the ARESIBO ontology

4.3.2.1 IoT and Sensors

There has been great effort in representing sensors and their observations, properties and features of interest. Towards this objective, the most well-known are the Semantic Sensor Network (SSN) (Compton et al., 2012) and Sensor, Observation, Sample, Actuator (SOSA) (Janowicz et al. 2019) ontologies, which in general they describe the notions of sensors, their observed properties, the involved procedures and actuators. They have been applied in various use cases and applications including satellite imagery, large-scale scientific monitoring, industrial and household infrastructures, social sensing, citizen science, observation-driven ontology engineering, and the Web of Things.

Additionally, sensors are essential for the intelligence, surveillance, and reconnaissance (ISR) domain, wherein tools and ontologies have been developed to support decision-making and improve mission planning. As for example, a sensor knowledge repository, namely OntoSensor (Russomanno et al., 2005), which establishes a widely accepted terminology of sensors, their properties, capabilities, and services. In addition, a similar framework is the

Missions & Means Framework (MMF) (Gomez et al., 2008) and the corresponding ontology which expresses MMF's complete Level and Operator set from both mission Synthesis and Employment perspectives (Figure 11: The MMF Ontology.).

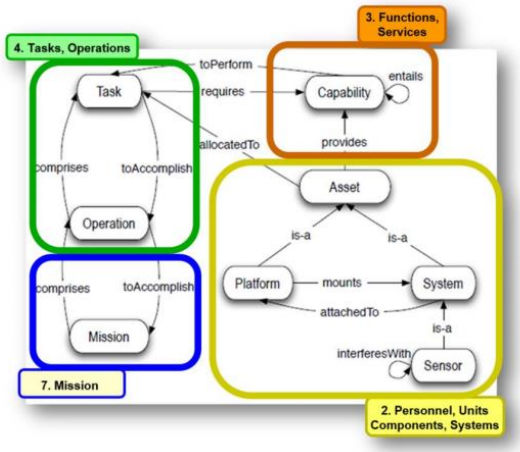

Figure 11: The MMF Ontology.

4.3.2.2 Time/Events

In most real-world applications, temporal information is vital and having knowledge of the temporal relationships between various transactions, events and orders is often critical. Such requirements are addressed by the development of the OWL-Time Ontology (Hobbs et al., 2006), (Pan et al., 2005). OWL-Time can describe the temporal properties of any real-world denoted resource and provides various and flexible representations that assist with queries and reasoning applications. It has a very expressive vocabulary for the ontology's core principles including interval, durations, and time positions and can represent temporal aggregates.

The Event Ontology (Liu et al., 2010) is cantered around the notion as an instance that occurs in a certain time, environment, including some participants and presenting some action features. An event may have a location, time, active agents, factors, products or relate with other events as seen in Figure 12: Core event model.

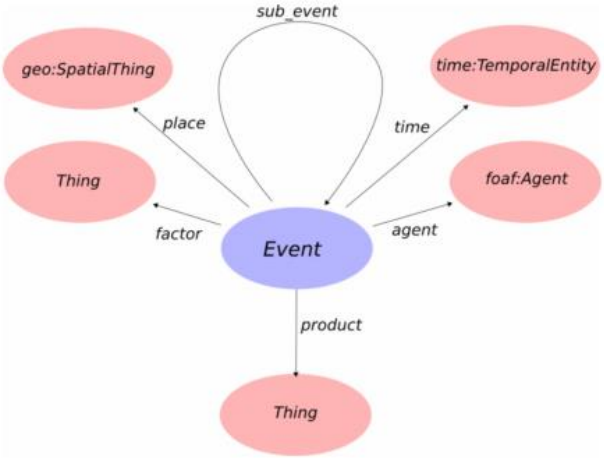

Figure 12: Core event model

Another ontology describing events and situations is ESO (Segers et al., 2015): The Event and Implied Situation Ontology, which formalizes the pre-, during-, and post-situations of *events* and the *roles* of the entities affected by an event. A *situation* is calculated as some abstract state where some properties and values hold. If an event occurs in the world, some

of these values will be modified. Overall, ESO ontology describes many relevant concepts and classes like Removing, Destroying, and Escaping.

4.3.2.3 Geospatial data

In the last years, a significant part of the research community focused on the production of geospatial data, their semantics and their use in Geographic Information Systems (GIS) as they offer powerful retrieval methods that enable users to execute complex queries. There are several ontologies and vocabularies that have been designed to express semantically the main notions of spatial data. The foremost of such ontologies are the GeoSPARQL 1.0 (Perry et al., 2012), an Open Geospatial Consortium Standard that defines an RDF/OWL vocabulary for representing spatial information and the query language for the RDF data. GeoSPARQL also includes a variety of powerful rules and functions that allows precise searching for relevant spatial information about the objects of interest (locations in geo-coordinates, functions for calculating distances between areas, etc.).
Complementarily, the NeoGeo Geometry Ontology[10] and NeoGeo Spatial Ontology[11] have also been proposed which comprise vocabularies for describing geographical regions in RDF and describe topological relations between features, respectively. In addition, WGS84 Geo Positioning (Brickley 2004) comprises a vocabulary for representing latitude, longitude and altitude information in the World Geodetic System geodetic reference datum. Additionally, relevant ontologies can be utilized like the Frame, Pixel, Place, Event (FraPPE) vocabulary, which enables Visual Analytics operations on geo-spatial time varying data and eases the correlation operations on geo-spatial data from different sources evolving over time.

4.3.2.4 Surveillance/Safety

Exchanging data and information is crucial in any integrated surveillance system rendering it faster, cheaper and more efficient. Towards this objective, the CISE data model (Berger et al., 2017), a common information-sharing environment, has been developed and implemented into the EUCISE-OWL ontology (Riga et al., 2019). The EUCISE-OWL ontology is a serialisation of the EUCISE2020 Data Model as an OWL ontology, conducted within the context of the ROBORDER EU-funded research project. They consider the corresponding data standards and identify the most useful aspects for maritime monitoring authorities representing all relevant sectors at EU and national level in a neutral, flexible, extensible and understandable way. The EUCISE-OWL ontology comprises of hundreds of classes and properties and can be integrated in a decision support or information system for supporting knowledge representation, event triggering, action inference, and information dissemination to the authorities. An excerpt of the concepts defined as classes is presented in Figure 13.

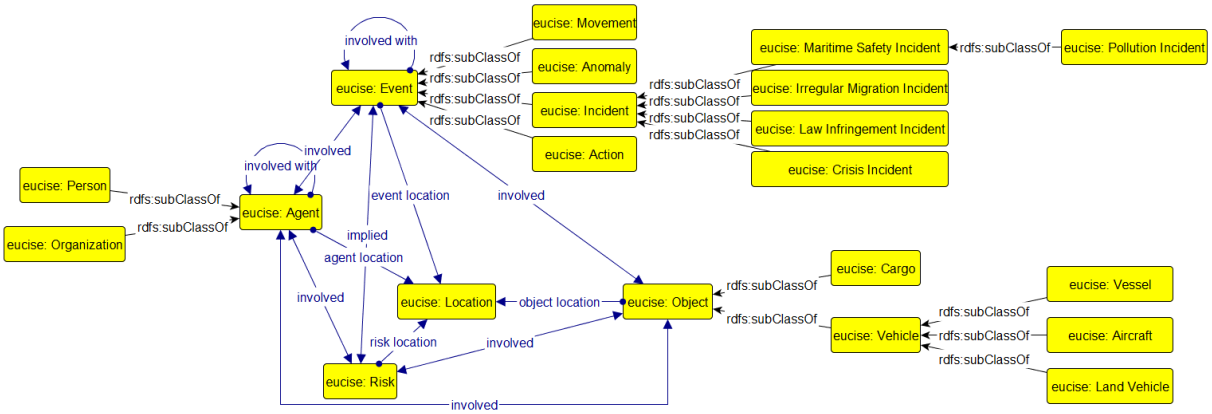

Figure 13: Core classes and main interrelationships of the EUCISE-OWL ontology.

[10] http://geovocab.org/geometry
[11] http://geovocab.org/spatial

The aforementioned ontology can be extended by including the concepts and descriptions in the Vehicle Sales Ontology (VSO[12]). VSO is a vocabulary for describing various types of land and maritime vehicles and could be used in the system to describe the operational assets. Moreover, the BeAware ontology (Baumgaftner et al., 2010) comprises another relevant example of useful KB. BeAware Ontology represents semantically all aspects pertinent to crisis management, some of which include emergencies, sensor data analysis, incidents and impacts. It introduces the concept of spatio-temporal primitive relations between observed real-world objects improving the situation awareness of the system operators.

4.3.2.5 Alerts

Numerous relevant works have also been proposed towards identifying ontology-based policies and alerts. Alerts can be mapped into attack contexts identifying the relevant policies and reacting accordingly to the corresponding threats. Well known ontology-based policy frameworks are Rei (Tonti et al., 2003) and KAos (Uszok et al., 2003). KAos domain/policy services and tools allow for the specification, management, conflict resolution and enforcement of policies within the specific contexts. Rei relies on an application-independent ontology to represent the concepts of rights, prohibitions, obligations, dispensations, policy rules as well as actions.
The notifications and the alerts that these policies produce are of paramount importance when an emergency is about to occur while it is critical for emergency systems to broadcast the relevant messages to all recipients. A well-known ontology that addresses the information needs for sharing and integrating emergency notification messages is the Simple Emergency Alerts 4 [for] All (SEMA4A) (Malizia et al., 2017) ontology. SEMA4A aims at establishing a deep correlation among available information about the user, the context of use and the situation and is composed by four elements as seen in Figure 14.

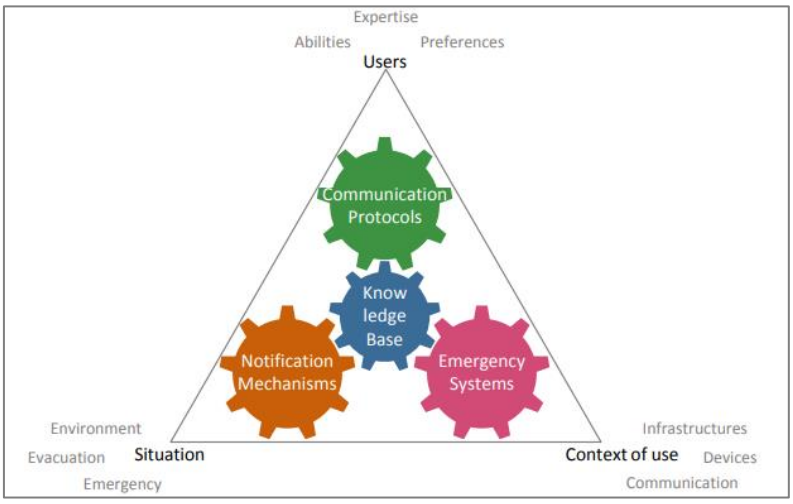

Figure 14: The SEMA4A architecture.

4.3.3 Ontology formalisation and implementation

The ARESIBO ontology is expressed in OWL 2 (W3C, 2012), a knowledge representation language widely used within the Semantic Web community for developing ontologies. Thus, we capitalise on its wide adoption as well as its formal structure and syntax, based on Description Logics (DL), a family of knowledge representation formalisms characterised by logically grounded semantics and well-defined reasoning services.
OWL 2 ontologies can be used along with information written in RDF, and themselves are primarily exchanged as RDF documents. Data is structured in RDF triples, which are

[12] https://www.w3.org/2001/sw/wiki/VSO

statements in the form `<subject predicate object>`. Each entity within a triple is associated with a Uniform Resource Identifier (URI), usually in the form of an http address, which is a unique identification that serves the principles of the Semantic Web. URIs involve two main parts: (a) the base URI (the leftmost) part of the URI which is common across multiple entities in a specific ontology; a short form (prefix) can be specified to represent the commonly used part of the URI; and (b) the URI fragment part which is the part of the URI after a delimiter (usually #); this part denotes a recognisable name for the described entity which should follow the basic rules and guidelines for naming and labelling ontologies (Noy and McGuinness, 2001).

The main building blocks of ontologies are *concepts* (or else *classes*) representing sets of objects (e.g., `Person`), *roles* (or else *properties*) representing relationships between objects (e.g., `worksIn`), and *individuals* (or else *instances*) representing specific objects (e.g., `Alice`, as an instance of `Person` class). Properties are further classified as: (a) *object* properties, which describe how classes and their individuals can be related to each other; (b) *data* properties, which attribute data values to individuals, either using default data types (e.g. string, integer, boolean, etc.) or within pre-defined data range expressions; and (c) *annotation* properties, which give additional description to the domain being modelled, without having any effect on the logical aspects of the ontology.

For developing and deploying the ontology that is described in the following subsection, we relied on the following tools:

⇒ **TopBraid Composer**[13] Free Edition, which is a visual modelling environment for creating and managing domain models. Its graphical user interface (GUI) enables the fast design and development of ontologies,

⇒ **SPARQL** (Harris & Seaborne, 2013) and **SPIN** (Knublauch et al., 2011), which serve as the semantic query language for submitting (insert/delete/update/fetch) queries to the ontology and running rules on top of the model,

⇒ **GraphDB**[14], which is a popular graph database for locally hosting test versions of the ontology and serving queries as a SPARQL endpoint,

⇒ **yEd Graph Editor**[15] and **Graffoo**[16] – yEd is a general-purpose diagramming program that can be used to draw many different types of diagrams via an intuitive user interface. Graffoo is a graphical framework for ontologies that can be loaded as a separate section in the yEd palette (Falco et al., 2014). We use both technologies to visualise information modelled in the ARESIBO ontology with a well-established, recognisable and easily interpretable way.

4.3.4 Ontology conceptualisation and mapping

In the current section, we describe in detail the conceptualisation of the first version (v1) of the ARESIBO ontology. In this first iteration, specific third-party vocabularies were adopted, which are indicated, in text and in figures that follow, with the use of their relevant prefixes in front of the class names (Table 72). For simplicity, those classes and properties, which have no prefix defined in descriptions and visualisations that follow, belong to the core ARESIBO ontology.

Table 72 – A list of utilised prefixes and their relevant ontologies

Prefix	Ontology	Namespace URI
aresibo	ARESIBO	http://160.40.51.22/mklab_ontologies/ARESIBO/aresibo#
eucise	EUCISE-OWL	http://160.40.51.22/mklab_ontologies/ROBORDER/eucise#

[13] https://www.topquadrant.com/products/topbraid-composer/
[14] https://www.ontotext.com/products/graphdb/
[15] https://www.yworks.com/products/yed
[16] http://www.essepuntato.it/graffoo

foaf	Friend-Of a-Friend	http://xmlns.com/foaf/0.1
geo	GeoSPARQL	http://www.opengis.net/ont/geosparql#
owl	OWL	http://www.w3.org/2002/07/owl#
rdfs	RDF Schema	http://www.w3.org/2000/01/rdf-schema#
sosa	Sensor, Observation, Sample, Actuator	http://www.w3.org/ns/sosa
xsd	XML Schema Definition Language (XSD)[17]	http://www.w3.org/2001/XMLSchema#

Thus, the core classes specified in the first version of the ARESIBO semantic model are described in detail below, while their between hierarchy is visualised in **Erreur ! Source du renvoi introuvable.**:

Alert: This class represents the alert messages that the ARESIBO KB will produce when specific conditions are met, increasing the situational awareness of the end-user. The class `Alert` is further divided into 4 subcategories (`Advisory`, `Caution`, `Cleared` and `WarningAlert`) as those where defined through the Alert data model (Section 3.4.16). An instance of `Alert` type can be asserted with an ID (`alertID` property), a level (`alertLevel` property) and a short description (`alertDescription` property). Also, an instance of alert can be associated with an instance of type `Detection` via the `producedByDetection` property (inverse of `producesAlert` property).

Context: This class represents either the spatial (`SpatialContext`) or the temporal context (`TemporalContext`) of an entity, meaning details about its location and time reference correspondingly. In the ARESIBO ontology, we have further specialised the class SpatialContext into more concrete definitions for covering different spatial relations among the involved entities. More details are given in subsection 4.3.4.2.

Dataset: This class represents the dataset produced by an analysis/monitoring ARESIBO component. In practice, it involves the fields (as defined in corresponding data models) and the actual values reported/produced by the different components of the ARESIBO system.

DetectedEntity: Any entity that is detected and reported by the ARESIBO components, can be an instance of `DetectedEntity` type.

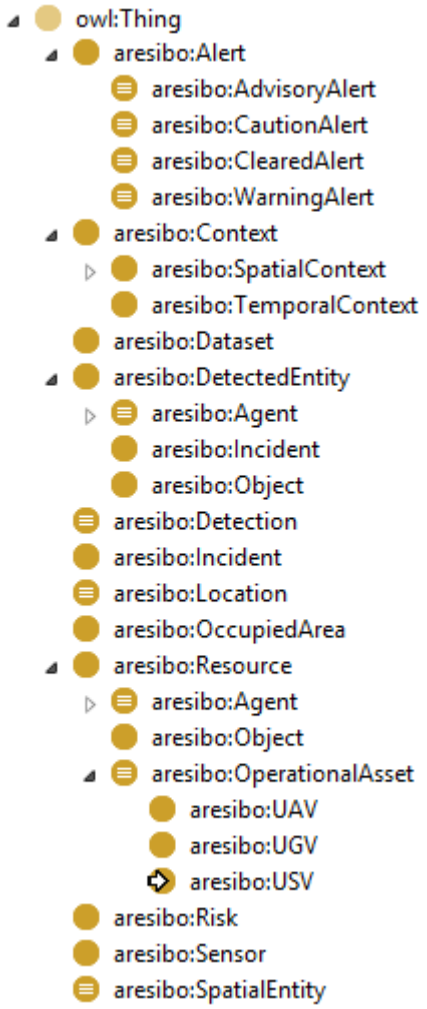

Figure 15: The hierarchy of the core classes of the ARESIBO ontology (v1)

Detection: This class represents all detections defined in a dataset. An instance of `Detection` class is asserted with one or more instances of `DetectedEntity` class via the property `detects`. Detected entities can be also of `Incident`, or of `Agent`, or of `Object` type.

Incident: This class represents an event taking place during a surveillance operation. Incidents can be further specialised into different types, which are enumerated in the ontology via the adopted EUCISE-OWL ontology.

[17] https://www.w3.org/TR/xmlschema11-1/

Location: This class represents a location (point or area), indicated by latitude, longitude, and radius. Any entity that is related to a location is of `SpatialEntity` type as well.
Resource: Within the ARESIBO context, we describe as resources the different operational forces, which can be either instances of `OperationalAsset` type (i.e., the different UxVs, drones, etc.), instances of `Agent` type (`Person`, group of Person, etc.), or instances of `Object` type (other type of vehicles, etc.)
Risk: This class is used to represent a more or less probable situation involving exposure to danger, or to a not desirable condition, that is affected by a, existing incident is the surveillance area. More details will be defined, within the ARESIBO Risk Model that will be implemented within T4.6 of the project.
Sensor: This class represents any instrument that observes a property or phenomenon with the goal of producing an estimate of the value of a parameter.
SpatialEntity: This class represents any entity that has a spatial reference defined either via the `Location` or via the `SpatialContext` type.
Task: This class represents a mission assigned to an operational asset (person, object) when an incident occurs.
A detailed representation of the associations defined between the core classes of the ARESIBO ontology is given in Figure 16. For the shake of brevity, we have omitted data type properties, as well as extensive class hierarchies. Generally, in illustrations that follow, based on the Grafoo notation, the yellow rectangles indicate the different classes, while the green rectangles denote data properties (i.e., properties that take a raw data value, like e.g. integers and strings).

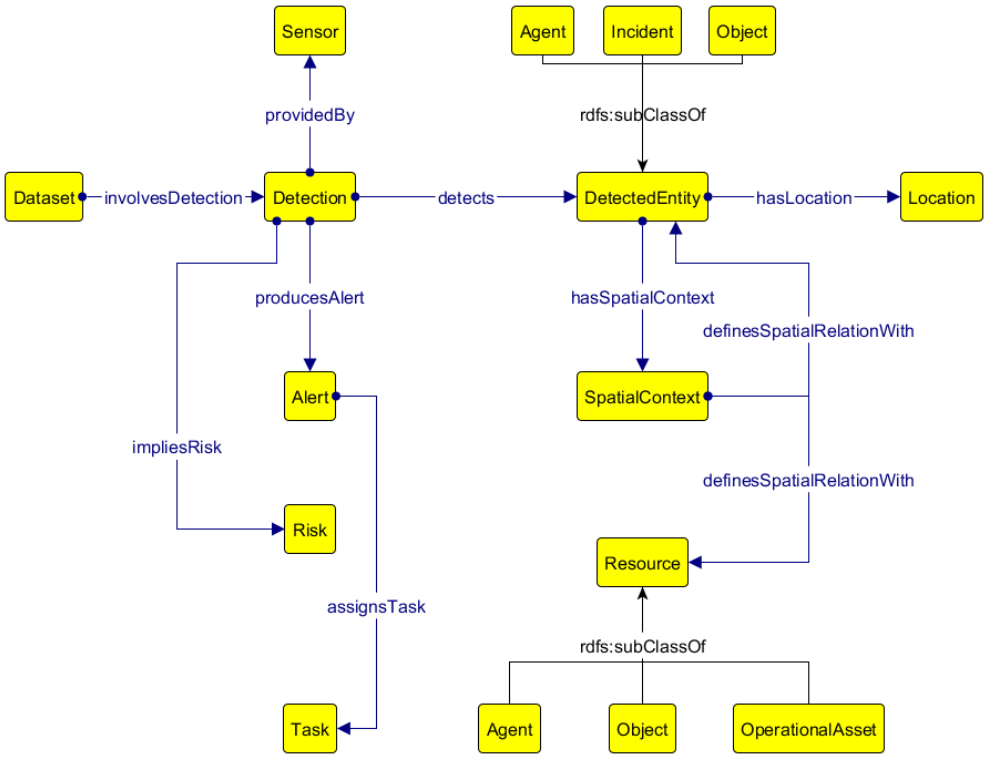

Figure 16: High-level overview of the core classes of the ARESIBO ontology v1

As previously mentioned, we have adopted and extended different concepts from other ontologies, and especially from EUCISE-OWL; the latter is the most relevant to our domain of interest, enabling information sharing among involved parties under surveillance issues, even though it is targeted for the maritime domain. We have collected in Table 73 the concepts that could be considered as similar, in terms of semantics, relations and the context where they are defined. The so-called *ontology mapping* enables the establishment of

semantic interoperability between new and existing sources, by defining the direct linking of classes/properties with third party ontologies and standards.

Table 73 – Mapping the core ARESIBO ontology concepts with third-party ones

ARESIBO Ontology Concept	Relation	Third-party Ontology Concept
`Analysis`	<Not defined yet>	-
`Agent`	`owl:equivalentClass` `rdfs:subClassOf`	`eucise:Agent` `foaf:Person`
`Alert`	<Not defined yet>	-
`Context`	<Not defined yet>	-
`Dataset`	`rdfs:subClassOf`	`eucise:Document`
`Detection`	`rdfs:subClassOf`	`eucise:Event`
`Incident`	`rdfs:subClassOf`	`eucise:Event`
`Location`	`owl:equivalentClass`	`eucise:Location`
`Object`	`rdfs:subClassOf`	`eucise:Object`
`OperationalAsset`	`owl:equivalentClass`	`eucise:OperationalAsset`
`Risk`	<Not defined yet>	-
`Sensor`	`rdfs:subClassOf`	`sosa:Sensor`
`SpatialEntity`	`owl:equivalentClass` `rdfs:subClassOf`	`geo:SpatialObject` `geo:Feature`
`Task`	`rdfs:subClassOf`	`eucise:Action`

4.3.4.1 Representing Analysed Data and Detections-ok

As already mentioned, the ARESIBO ontology encompasses information relevant to the analysis of input data coming from the various sensors of the framework. This information is fed to the ontology from the analysis components; the core constructs in the ontology are illustrated in Figure 17.

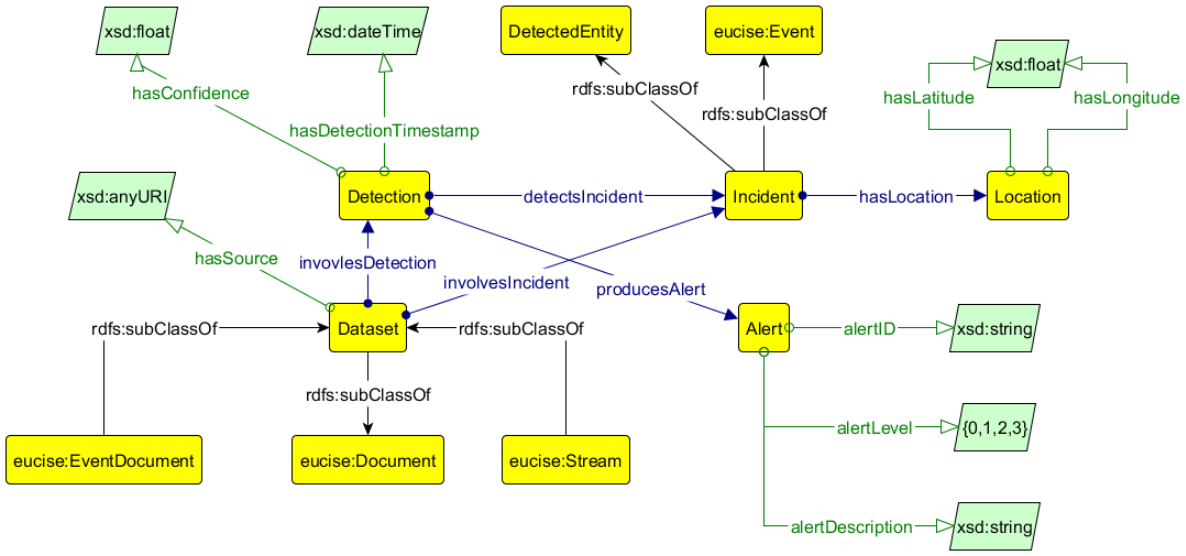

Figure 17: Representation of the analysed data in the ARESIBO ontology

More specifically, any data or analysis reported by a sensor or a component operating within the ARESIBO system, can be represented in the ontology as an instance of `Dataset` class. The latter is an extension of `eucise:Document` and inherits all its relevant declarations. Also, a dataset can be associated with a unique URI via the `hasSource` property. The dataset is asserted to an instance of type `Detection` through the use of the `involvesDetection` property. In the example, we assume that the detected entity is an instance of `Incident` type

(subclass of `eucise:Event`), but it could be also an instance of `Agent` or of `Object` type, in general. The occurrence of a specific incidence or the detection of specific spatial objects can raise an `Alert`, which is associated with the detected entity via the property `producesAlert`.

4.3.4.2 Representing Spatial Relations

Most of the requirements defined in Section 4.3.1 on the basis of the ARESIBO PUCs, describe use cases that are relevant to trespassing an area, border crossing, and thus involve the notion of Location. In order to handle such issues within the ontology, we need to include concepts and properties that will enable the definition of the spatial relation between the examined parameters. Thus, within the ARESIBO KB, we define the class `SpatialContext` as the concept that may describe the spatial relation between two or more involved instances of `SpatialEntity` type (i.e., entities that may have a spatial representation/reference asserted to them). The class `SpatialContext` is further discretised in different types of spatial relations, like for example the `AbsoluteDistanceSpatialContext`, which defines the absolute distance between two (or more) entities of interest (Figure 18).

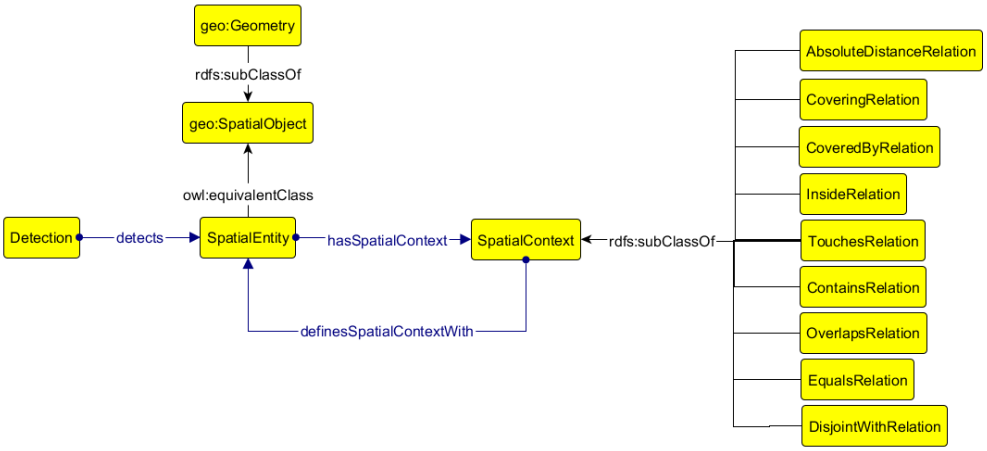

Figure 18: Representation of the spatial relations between spatial entities in the ARESIBO ontology

These definitions may form the basis for creating a proper semantic reasoning framework (Section 4.4), by implementing relevant SPARQL rules that will enable for example the constant monitoring of the distance between two involved entities and the raise of a proper alert to the end user (C2, operator) when the current distance is less than a specific threshold. The GeoSPARQL ontology that we adopt, implements important query functions that support the definition of different spatial relations, such as the Egenhofer (Egenhofer, 1989) and the RCC8 (Cohn, 1997) relation family (Table 74), as well as of the Euclidean distance (`geof:distance`[18]) between two geometries. By adopting these functions, we achieve to enrich the operational capabilities of the ARESIBO KB, in terms of defining spatial relations between the involved entities and resources, without any additional effort in expressing the actual calculations of the given spatial relations.

Table 74 – SPARQL query functions adopted from GeoSPARQL

Relation family	GeoSPARQL function	Comment
Egenhofer	`geof:ehEquals`	two geometries are topologically equal if their interiors intersect and no part of the interior or boundary of one

[18] geof: is the prefix of GeoSPARQL Functions

		geometry intersects the exterior of the other
	geof:ehDisjoint	x and y have no point in common
	geof:ehMeet	x and y have at least one point in common, but their interiors do not intersect
	geof:ehOverlap	x and y have some but not all points in common, they have the same dimension, and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves
	geof:ehCovers	when x covers y means that every point of y is a point of (the interior or boundary of) x
	geof:ehCoveredBy	every point of x is a point of (the interior or boundary of) y (extends Inside relation)
	geof:ehInside	x lies in the interior of y
	geof:ehContains	when x contains y means that geometry y lies in x, and the interiors intersect
RCC8	geof:rcc8eq	x is identical with y
	geof:rcc8dc	x is disconnected from y
	geof:rcc8ec	x is externally connected to y
	geof:rcc8po	x partially overlaps y
	geof:rcc8tpp	x is a tangential proper part of y
	geof:rcc8tppi	x is a tangential proper part inverse of y
	geof:rcc8ntpp	x is a nontangential proper part of y
	geof:rcc8ntppi	x is a nontangential proper part inverse of y

For utilising the GeoSPARQL functions in practice, we define the involved instances of SpatialEntity class as instances of geo:Feature type as well (relation denoted in Figure 18). This mapping ensures that the ontology remains consistent, on the basis of the GeoSPARQL definitions that are adopted, and also that the instance of SpatialEntity may inherit all the relevant properties asserted to the geo:Feature, such as the geo:hasGeometry that connects a spatial object with a specific geometry representing a point or an area (polygon), and the geo:asWKT that connects the geometry with specific latitude and longitude values. A detailed instantiation of spatial entities and relations is given in Figure 19; this example involves the representation of a detected person and its distance from a restricted area. With pink circles we present the instantiations (individuals/instances) of the different classes; the labels that are attached to them are written in the form of "instance_label::classType(s)", where "**::**" is used as a delimiter between the name of the instance (instance_label) and the type of the class/-es[19] that it belongs.

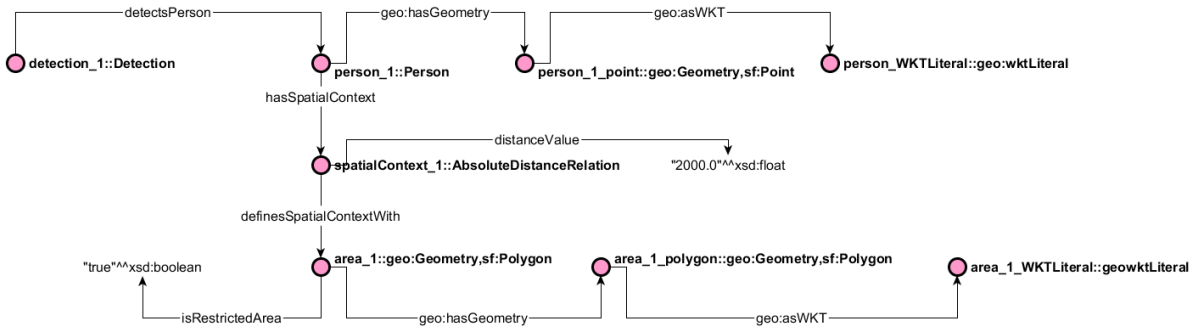

Figure 19: Representation of a detected person close to a restricted location, on the basis of the ARESIBO ontology.

[19] If more than one classes are defined, then these are divided with the use of comma (,), moving from the more general (e.g., geo:Geometry) to the more specific (e.g., sf:Polygon) class.

4.3.5 Ontology Evaluation

The evaluation of ontologies is an emerging field of research in the Ontological Engineering community that deals with the problem of assessing a given ontology from the point of view of a particular criterion of application. Existing ontology evaluation methods generally propose automated or semi-automated approaches that focus in specific qualitative (number of classes, properties, axioms, etc.) or quantitative criteria (consistency, completeness, expandability, sensitiveness, etc.) used to assess the examined ontology. An integrated review of ontology evaluation methods is attributed in (Gangemi et al., 2005; Brank et al., 2005). Such techniques will help uncover errors in implementation, and inefficiencies regarding the modelling, the complexity and size of the ontologies. Nevertheless, no evaluation method (either as stand-alone or in combination) can guarantee a good ontology; on the contrary, it can definitely recognize problematic parts of it in terms of structure and consistency (Vrandečić, 2009).

For the current task, we have performed different types of ontology evaluations, with regards to its consistency, quality and structure. The results follow in subsections below.

4.3.5.1 Evaluating the consistency and quality

For evaluating the overall consistency and quality of the ontology we used OOPS (OntOlogy Pitfall Scanner), an online tool for detecting the most common pitfalls[20] in ontologies (Poveda-Villalón et al., 2014). After analysing the ontology, OOPS provides a list with all the pitfalls it detected along with the associated negative consequences and suggests modifications in order to improve the quality of the ontology. The tool can detect:

⇒ Critical pitfalls affecting the ontology's consistency, which are crucial to be corrected,

⇒ Important pitfalls, which are not equally critical, but are considered also important to be corrected,

⇒ Minor pitfalls, which do not cause any critical problems, but correcting them will improve the quality of the ontology.

We submitted the current version (v1) of the ontology to OOPS and we have already corrected all the detected pitfalls, which were critical but were made due to accidentally wrong definitions in domain/range values of object properties. The current version of the ontology has no more pitfalls, with the exception of some pitfalls concerning the imported ontologies, which, as a consequence were left unresolved.

4.3.5.2 Evaluating the structure

For evaluating the structure, we relied on OntoMetrics[21], an online framework that validates ontologies based on established metrics. Table 75 presents the results derived from the aforementioned analysis. Base Metrics comprise of simple metrics, like the count of classes, axioms, objects etc.; these metrics show the quantity of ontology elements. Schema metrics, on the other hand, address the design of the ontology; metrics in this category indicate the richness, width, depth, and inheritance of an ontology schema design.

Table 75 – Ontology metrics of the implemented ARESIBO ontology (v1), as generated by OntoMetrics tool

Base Metrics	aresibo ontology	imported ontologies	Total
Number of triples	297	13597	**13849**
Class count	28	256	**284**
Object property count	22	207	**229**
Data property count	14	172	**186**

[20] A catalogue of common pitfalls is given at http://oops.linkeddata.es/catalogue.jsp
[21] https://ontometrics.informatik.uni-rostock.de

Individual count	0	863	**863**
DL expressivity	**ALCHIF(D)**		
Schema Metrics	**Total**		
Attribute richness	0.285714		
Inheritance richness	0.678571		
Relationship richness	0.6		
Axiom/class ratio	3.696429		
Inverse relations ratio	0.409091		
Class/relation ratio	0.589474		

Starting with the base metrics, the total count of classes and properties of the ARESIBO ontology indicates that it is a rather lightweight model, which could be easily adopted by various applications, contrary to heavier “monolithic” ontologies that pose significant challenges in integration. However, important additions should and will come up in the next versions of the ontology, where the functional and non-functional requirements will be further specialised according to the end-users and to the technical partners as well.
Regarding the schema metrics, the definitions that follow are adopted from (Gandemi et al., 2005). Attribute richness is defined as the average number of attributes per class and can indicate both the quality of ontology design and the amount of information pertaining to instance data. The more attributes that are defined the more knowledge the ontology conveys. Inheritance richness is defined as the average number of subclasses per class and is a good indicator of how well knowledge is grouped into different categories and subcategories in the ontology. This measure can distinguish a horizontal ontology (where classes have a large number of direct subclasses) from a vertical ontology (where classes have a small number of direct subclasses). The respective value in the table indicates that the proposed ontology lies somewhere in between; this is reasonable, since the ontology covers many aspects (breadth) while thoroughly modelling some of them (depth). Relationship richness refers to the ratio of the number of non-inheritance relationships (i.e. object properties, equivalent classes, disjoint classes) divided by the total number of inheritance (i.e. subclass relations) and non-inheritance relationships defined in the ontology. This metric reflects the diversity of the types of relations in the ontology. Finally, axiom/class ratio, class/relation ratio, and inverse relations ratio describe the ratio between axioms-classes, classes-relations, and inverse relations-relations, respectively, and are indications of the ontology’s transparency and understandability.

4.4 Semantic Reasoning

An indicative interaction between the Visual Analysis tool, the KB and the Dashboard is framed within Figure 20; the represented concept is based on a specific ARESIBO Project Use-Case (PUC), which regards an incident of land-border trespassing.
The term "semantic reasoning" refers to the process of deriving facts that are not explicitly expressed in an ontology. Consequently, a "semantic reasoner" (also often referred to as "reasoning engine", "rules engine" or simply "reasoner") is a piece of software able to infer logical consequences from a set of asserted facts or axioms in an ontology. A few examples of tasks required from a semantic reasoner are as follows:

⇒ Satisfiability of a concept, i.e., to determine whether a description of the concept is not contradictory, namely, whether an individual can exist that would be an instance of the concept,

⇒ Sub Sumption of concepts, i.e., to determine whether concept C subsumes concept D, namely, whether description of C is more general than the description of D,

⇒ Check an individual, i.e., to check whether the individual is an instance of a concept,

⇒ Retrieval of individuals, i.e., to find all individuals that are instances of a concept,

⇒ Realization of an individual, i.e., to find all concepts which the individual belongs to, especially the most specific ones.

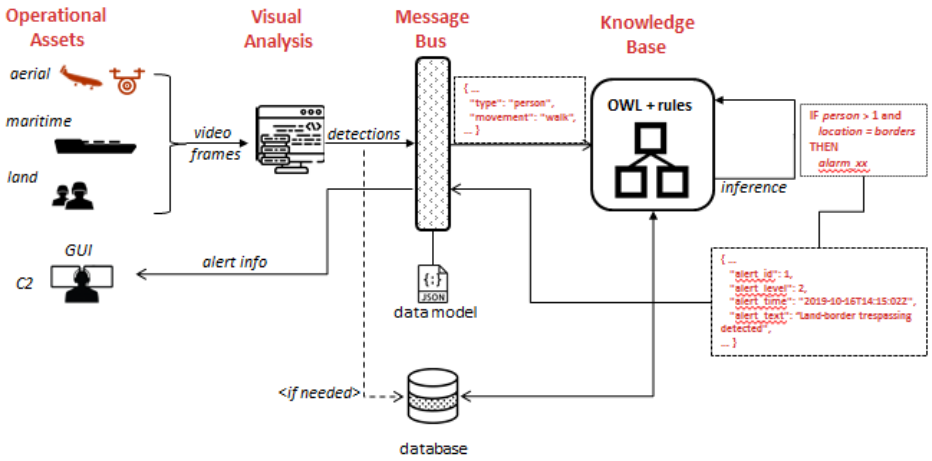

Figure 20: The role of the ARESIBO KB, on the basis of PUCs

Within the project, the ARESIBO ontology will accept input from other modules analysing sensor outputs and will perform semantic reasoning via SPARQL-based rules. Rule-based reasoning satisfies the above points, plus additional aspects, like for example finding all concepts that satisfy a defined rule, or creating new instances of all concepts that satisfy a defined rule. A set of indicative semantic reasoning scenarios that the ontology will address are outlined below – a more complete list of semantic reasoning scenarios will be determined once the user requirements analysis is concluded.

⇒ Determine position and proximity of objects of interest: e.g. where is my smartphone?
⇒ Determine position and proximity of locations of interest: e.g. where is the exit?
⇒ Determine position and proximity of persons of interest: e.g. where is my companion?
⇒ Infer potential risks in user's current situation: e.g. stairs ahead, vehicle approaching, etc.
⇒ Infer types of activities performed by people in the vicinity of the user: e.g. two people in front of each other means that they are probably discussing.
⇒ Determine set of suggested actions in order to achieve something: e.g. get out of the room or issue a ticket on the bus.

Regarding the implementation of the semantic reasoning module, the following parameters are foreseen:

⇒ Input: The semantic reasoning does not require any specific input, other than the triggering of the reasoning execution.
⇒ Output: The output from semantic reasoning is an ontology file including both the initially asserted and the inferred information.
⇒ Execution intervals: Every several minutes or on demand (e.g. whenever new knowledge is inserted into the ontology).
⇒ Involved technologies: The relevant RDF Service module will be based on Java/Python 2.x or 3.x, Apache Jena, SPARQL, SPARQLWrapper, RDFLib; a REST API will be deployed with a configured public IP/port or domain name.
⇒ Critical factors: A valid rule set for the reasoning process (see reasoning scenarios above) is essential for the inference of meaningful knowledge. Hence, it is critical for all involved domain experts (e.g. end user partners) to contribute actively to the task of assembling this rule set.

5 Conclusions and future work

This deliverable presents the first iteration of (i) the ARESIBO Data Model, being a common protocol for the communication between the ARESIBO components, and (ii) the ARESIBO Knowledge Base, for facilitating the semantically-enriched representation of incidents, resources and tasks that substantially exist/act/are detected within the operational field. The proposed implementation of the ARESIBO Data Model and the semantic technologies adopted for the ARESIBO Knowledge Base are uniform and modular and can be easily enriched with additional concepts and schemas by extending the already existing definitions of the model/schema correspondingly.

The already presented implementations, both in the ARESIBO Data Model and the KB, will be refined once the end-user requirements analysis process (WP2) is concluded, which will further specify the technical requirements and functional operations of the proposed system. An additional point to consider in our future work revolves around the actual architecture of the ARESIBO system, which may further specify the communication needs and message details that should be exchanged within the ARESIBO system, and thus the overall data model will be enhanced. These issues will be further explored in the coming months. Another important utility that will be covered once the user requirements are finalised, is the creation of a semantic reasoning framework, which will integrate a proper set of ontology-based rules that will handle the process of inferring meaningful information to the end user. Within the context of the aforementioned task, we will investigate the adoption of Semantic Complex Event Processing techniques and Stream Reasoning techniques (Taylor and Leidinger, 2011) that may facilitate the management of heterogeneous data sourced from different sensors simultaneously and support inferencing.

References

Baumgartner, Norbert, Wolfgang Gottesheim, Stefan Mitsch, Werner Retschitzegger and Wieland Schwinger. “BeAware! - Situation awareness, the ontology-driven way.” *Data Knowl. Eng.* 69 (2010): 1181-1193.

Bechhofer, S, “OWL: Web ontology language. , pp.,” in *Encyclopaedia of Database Systems*, US, Springer , 2009, pp. 2008-2009.

Bernaras, A., Laresgoiti, I. and Corera, J. , “Building and reusing ontologies for electrical network applications,” in *Proceedings of the European Conf. on Artificial Intelligence (ECAI’96)*, 1996.

Berger, D., Hermida, J., Oliveri, F. & Pace, G. “The Entity Service Model for CISE - Service Model Specifications” Technical Report, Joint Research Centre of the European Commission, 2017

Berners-Lee, T., Hendler, J., & Lassila, O, “The semantic web.,” *Scientific American,* vol. 284, no. 5, 2001.

Brank, J., Grobelnik, M. and Mladenić, D., “ A survey of ontology evaluation techniques,” in *Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005)*, Ljubljana, Slovenia., 2005.

Brickley, D. "WGS84 geo positioning: an RDF vocabulary." *Dostupné* (2004).

Cohn, A.G., Bennett, B., Gooday, J. and Gotts, N.M., “Qualitative spatial representation and reasoning with the region connection calculus,” *GeoInformatica,* vol. 1, no. 3, pp. 275-316., 1997.

Compton, Michael, et al. "The SSN ontology of the W3C semantic sensor network incubator group." *Web semantics: science, services and agents on the World Wide Web* 17 (2012): 25-32.

Egenhofer M.J., “ A formal definition of binary topological relationships,” in *Foundations of Data Organization and Algorithms. FODO 1989.*, Berlin, Heidelber., 1989.

Falco, R et al., “Modelling OWL ontologies with Graffoo.,” in *European Semantic Web Conference (pp. 320-325)*, Cham, 2014.

Fernandez, M et al, “METHONTOLOGY: From Ontological Art Towards Ontological Engineering.,” in *AAAI Technical Report SS-97-06, pp. 33-40.*, 1997.

Galluzzo, T. and Kent, D., “The OpenJAUS Approach to Designing and Implementing the New SAE JAUS Standards,” OpenJAUS, 2011.

Gangemi, A. et al., “ A theoretical framework for ontology evaluation and validation. Proceedings of the Semantic Web Applications and Perspectives (SWAP),,” in *2nd Italian Semantic Web Workshop*, Trento, Italy, 2005.

Gomez. Mario et al., “An Ontology-Centric Approach to Sensor-Mission Assignment,” in *International Conference on Knowledge Engineering and Knowledge Management*, Berlin, Heidelberg, 2008.

Grüninger, M. and Fox, M.S. , “The role of competency questions in enterprise engineering.,” in *Workshop on Benchmarking – Theory and Practice,pp. 22-31. Springer US.*, 1994 .

Harris, S., Seaborne, A. and Prud’hommeaux, E., SPARQL 1.1 query language, W3C recommendation, 21(10), 2013.

Hobbs, Jerry R., and Feng Pan. "Time ontology in OWL." *W3C working draft* 27 (2006): 133.

Janowicz, Krzysztof, et al. "SOSA: A lightweight ontology for sensors, observations, samples, and actuators." *Journal of Web Semantics* 56 (2019): 1-10.

Jarrar, M. and Meersman, R., “Ontology Engineering - The DOGMA Approach. Advances in Web Semantics,” in *LNCS Vol. 4891, pp. 7-34.*, 2008.

Kent, D., Galluzzo, T., Bosscher P. and Bowman, W., “Robotic Manipulation and Haptic Feedback via High Speed Messaging with the Joint Architecture for Unmanned Systems (JAUS),” OpenJAUS, 2014.

Knublauch, H., Hendler, J. A. and Idehen, K. , “SPIN - overview and motivation. W3C Member Submission.,” 2011.

Lenat, D.B. and Guha, R.V. , “Building large knowledge-based systems; representation and inference in the Cyc project. Addison-Wesley Longman Publishing Co., Inc.,” 1989.

Liu, Wei et al., “Extending OWL for Modeling Event-oriented Ontology,” in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, Krakow, Poland, 2010.

Malizia, A. et al., “SEMA4A: An ontology for emergency notification systems accessibility,” *Expert Systems with Applications,* vol. 37, no. 4, pp. 3380-3391, 2017.

Marques, M.M., “STANAG 4586 –Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability,” 2012.

NATO Standardization Agency , “ STANAG 4609 JAIS (Edition 3) – NATO Digital Motion Imagery Standard,” 2009.

NATO Standardization Agency, “STANAG 4586 (Edition 3) – Standard Interfaces of UAV Control System (UCS) For NATO UAV Interoperability.,” 2012.

Noy, N.F. and McGuinness, D.L., “Ontology development 101: A guide to creating your first ontology.,” 2001.

OASIS , “Emergency Data Exchange Language Resource Messaging (EDXL-RM) 1.0,” 2008.

OASIS, “Common Alerting Protocol Version 1.2,” 2010.

Pan, F. and Hobbs, J.R., “Temporal Aggregates in OWL-Time.,” in *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference*, Clearwater Beach, Florida, USA, 2005.

Perry, Matthew, and John Herring. "OGC GeoSPARQL-A geographic query language for RDF data." *OGC implementation standard* 40 (2012).

Pinto, H.S., Staab, S. and Tempich, C., “DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies.,” in *Proceedings of the 16th European Conf. on Artificial Intelligence (ECAI), pp. 393-397*, 2004.

Poveda-Villalón, M., Gómez-Pérez, A., & Suárez-Figueroa, M. C. ., “(2014). Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation.,” in *International Journal on Semantic Web and Information Systems (IJSWIS), 10(2), 7-34*, 2014.

Riga, M., et al, “EUCISE-OWL: An Ontology-based Representation of the Common Information Sharing Environment (CISE) for the Maritime Domain,” 2019.

ROBORDER Consortium, “Grant Agreement number 740593 ROBORDER,” European Commission, 2017.

Russomanno, David J., Cartik R. Kothari, and Omoju A. Thomas. "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models." *IC-AI*. 2005.

Segers, R., Vossen, P., Rospocher, M., Serafini, L., Laparra, E., & Rigau, G. (2015). Eso: A frame-based ontology for events and implied situations. *Proceedings of MAPLEX*, *2015*.

Serrano, D. “Introduction to JAUS for Unmanned Systems Interoperability,” ” NATO Science & Technology Organization, Cerdanyola del Vallès, Spain, 2015.

Suárez-Figueroa, M., Gómez-Pérez, A. and Villazón-Terrazas, B., “How to write and use the Ontology Requirements Specification Document. On the move to meaningful internet systems: OTM 2009. Part of the Lecture Notes in Computer Science book series,” 2009.

Sure, Y., Staab, S. and Studer, R., “On-To-Knowledge Methodology (OTKM).,” *Handbook on Ontologies,* pp. 117-132, 2004.

Swartout, B., Ramesh, P., Knight, K. and Russ, T. , “Towards distributed use of large-scale ontologies.,” in *Symposium on Ontological Engineering of AAAI, pp. 138-148.*, 1997.

Taylor K., Leidinger L. , “Ontology-Driven Complex Event Processing in Heterogeneous Sensor Networks. In: Antoniou G. et al. (eds) The Semanic Web: Research and Applications.,” *ESWC 2011 Lecture Notes in Computer Science,* vol. 6644, 2011.

Tonti G., Bradshaw J.M., Jeffers R., Montanari R., Suri N., Uszok A., “Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder,” in *The Semantic Web - ISWC 2003*, Berlin, Heidelberg, 2003.

Uschold, M. and King, M., “Towards methodology for building ontologies.,” in *Workshop on Basic Ontological Issues in Knowledge Sharing, held in Conjunction with IJCAI-95.* , Cambridge, UK., 1995.

Uszok, Andrzej et al., “KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement,” in *Policies for Distributed Systems and Networks*, Lake Como, Italy, 2003.

Vrandečić, D, “Ontology Evaluation, Handbook on Ontologies. International Handbooks on Information Systems,” pp. 293-313, 2009.

W3C, “OWL 2 Web Ontology Language Document Overview” (Second Edition). W3C Recommendation 11 December 2012, available online: http://www.w3.org/TR/owl2-overview/