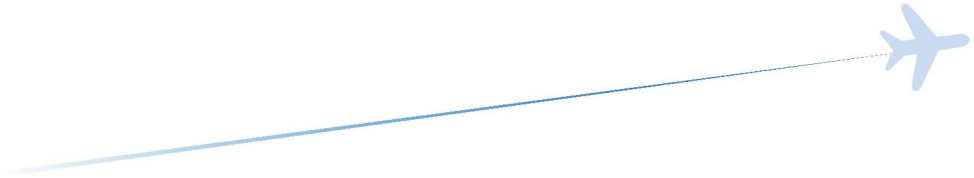

IMPETUS platform and micro-services model

DeliverableID	**D4.1**
ProjectAcronym	**IMPETUS**
Grant:	**763807**
Call:	**H2020-SESAR-2016-1**
Topic:	**RPAS 02 - Drone Information Management**
Consortium coordinator:	**CRIDA A.I.E.**
Edition date:	**09 September 2019**
Edition:	**00.01.00**
Template Edition:	**02.00.00**

Founding Members
EUROPEAN UNION EUROCONTROL
SESAR JOINT UNDERTAKING

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Nicolás Peña	BR&TE PMST representative	25/08/2019
Javier Espinosa	INECO PMST representative	25/08/2019
Chris Foster	ALTITUDE ANGEL PMST representative	25/08/2019
Anna-Lisa Mautes	JEPPESEN PMST representative	25/08/2019

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Pablo Sánchez-Escalonilla	CRIDA PMST representative and manager	30/08/2019

Approved for submission to the SJU By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
Pablo Sánchez-Escalonilla	CRIDA PMST representative and manager	31/08/2019
Nicolás Peña	BR&TE PMST representative	31/08/2019
Javier Espinosa	INECO PMST representative	31/08/2019
Anna-Lisa Mautes	JEPPESEN PMST representative	31/08/2019
Chris Forster	ALTITUDE ANGEL PMST representative	31/08/2019
Michael Christian Büddefeld	TUDA PMST representative	31/08/2019
Aleksej Majkic	TUDA PMST representative	31/08/2019

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date

Document History

Edition	Date	Status	Author	Justification
Structure Proposal		Accepted	Chris Foster	Creation of the document
00.00.01		Draft	All	Integration of all contributions
00.01.00		Final	Pablo Sánchez-Escalonilla	Peer review and review of executive summary

IMPETUS

INFORMATION MANAGEMENT PORTAL TO ENABLE THE INTEGRATION OF UNMANNED SYSTEMS

This deliverable is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 763807 under European Union's Horizon 2020 research and innovation programme.

Abstract

IMPETUS D4.1 summarises the set of services developed by consortium members in work package 4 that will be utilised during the experimental testing outlined in D5.2.

Table of Contents

List of Tables

List of Figures

Executive Summary

This report summaries the models implemented by the consortium partners as part of work package 4:

- Model for testing drone-specific weather service. This model will allow executing Exercise#1 which illustrates how a better knowledge of the uncertainty in the meteorological prediction will improve the robustness of trajectory-based decision making processes underlying Mission Plan, Flight Planning and Traffic Management services;

- Model for testing flight planning management service. This model will allow executing Exercise#2 which investigates how the Flight Planning Management Service can be used as a controlling entity for flight plan submission and strategic deconfliction. The exercise will show how to guarantee the successful and safe adaptation of the initial submitted flight plan when the Flight Planning Management Service interacts with the Local Weather service, the Aeronautical Information Management service and the Mission Plan Management Service;

- Model for testing flight planning management service. This model will allow executing Exercise#3 which explores how dynamic information, especially surveillance data and the positions of obstacles, is gathered, integrated and provided to all the actors involved in the operation;

- Model for testing dynamic capacity management services. This model will allow executing Exercise#4 which explores the services which are needed to dynamically managed the airspace in the execution phase, considering both the standard separation criteria proposed by CORUS and new separation criteria that take diverse drone capabilities into account. Furthermore, the exercise will assess how to deal with dynamic changes in airspace restrictions by determining the safest path for the affected drones to take.

1 Introduction

1.1 Purpose of the document

The purpose of this document is to allow consortium members to declare the models that have been created within work package 4 that will be used during the experimental testing.

1.2 Intended readership

This document is intended to be used by IMPETUS members and SJU (included the Commission Services).

1.3 Acronyms and Terminology

It is necessary to highlight that UTM acronym is used in this document both for the general notion of a drone traffic management system and for the specific system which will be designed in the USA.

Table 1: Acronyms

Abbreviation	Description
AWS	Amazon Web Services
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
MQ	Message Queue
QPID	Queuing Protocol Identification
REST	Representational State Transfer
SSL	Secure Sockets Layer
STOMP	Simple Text-orientated Messaging Protocol
SWIM	System Wide Information Management
UTM	Unmanned traffic management (general term)
UTM	Unmanned Aircraft System Traffic Management (USA)

2 Availability Note Forms

2.1 Model for testing drone-specific weather service

Availability Note by MEMBER(S):	BR&T-E
Availability Note Date:	31/07/2019
Person Responsible Item's Availability:	Nicolas Peña Ortiz (BR&T-E)
Deliverable Code:	D4.1
Item's Name and Version:	Local Weather Service. Version 1.0
Details of Availability	
Step and Maturity Level addressed by the Item implementation:	V01 according to E-OCVM
Documents and their versions used for the Item's development:	• D02.02 Drone Information Services; • D03.01 Architecture and technical requirements.
Item's Verification related documentation:	D05.01 Experimental Plan
Date and location of the Item's Verification:	BR&T-E Madrid Laboratory, July 2019
Verification Overall Result:	The Local Weather service prototype has been successfully developed. This includes both the core modules of the service and the collection of Docker Containers and middleware elements forming part of its interface. It has been tested generating ensembles of 7 iso-probable predictions every 12 hours with two different look-ahead times (24 hours and 72 hours) for an area in in the north west of Spain with a surface of 50Km * 50 Km.
Information on the Content:	
PROTOTYPE TECHNOLOGIES The local weather probabilistic service is composed by many pieces of technology using always cloud friendly technologies. Technology-wise there are two different blocks: the core weather service and the infrastructure to offer it to the clients. The core weather service is written in C++ making use of a parallel execution library that allows it to scale the number of nodes it is running on to adapt it to the amount of calculations required (size of volume to cover, frequency of predictions, number of requested iso-probable predictions per run, etc.). While it is deployed on-premises, all data flows and instances required are compatible with both AWS and Azure deployments. A minor disadvantage of the present configuration is that the core weather service component would require some modifications to be deployed in a hybrid cloud environment due	

to the intense communication between different processes.

All elements of the infrastructure to offer the service to the clients are deployed in the form of Linux-based micro-services that run in individual dock containers, making them very easy to deploy in both AWS and Azure. They are also ready for utilization in hybrid cloud environments. These are also coded in C++. The selection of C++ for every development was decided to optimize resource utilization and performance of the service. It is our intent to further develop this prototype to turn it into a key part of a future commercial product.

Data and Services flows between the different components are all implemented following the guidelines of the EUROCONTROL Specifications for SWIM Technical Infrastructure (TI) Yellow Profile, using always technologies and standards listed there. In our opinion, this profile and its intended uses are perfect for many of the UTM services to be deployed, including the local weather service. Specifically, our implementation uses QPID proton AMQP C++ library for the publish/subscribe flows (both topics and queues) and REST interfaces using a high performance C++ implementation of both the server and client parts for the request/replay flows.

All data flows support authentication and encryption through SSL Certificates.

PROTOTYPE ARCHITECTURE

The Local Service has many components in this microservice-based prototype:

- **Core Weather Service:** This is the scalable engine that performs the actual weather predictions. It is completely isolated from direct contact with the clients and its scaling is controlled by the Clients Weather Service Gateway Microservice. The Core Weather service identifies overlapping requests and minimizes the computation required to run all pending simulations;

- **Clients Weather Service Gateway Microservice**: This is the entry point of any petition of weather information to the service. It offers a REST interface that the clients use to authenticate and provide the request parameters (area, time window, variables, etc.). The gateway passes the job parameters to the Core Weather Service scheduler and instantiates a Client Agent to manage all communications and data flows between the client and the service;

- **Client Agent Microservice**: There is one instance of this Microservice per client. It reads all the AMQP topics coming from the Core weather service that are relevant to the client (their cells are covered by the volume of interest for the client), transforms and adapts all incoming data to satisfy the parameters of the specific client and directs the resulting data flow to an AMQP queue that the client is subscribed to. It also offers a REST interface that allows the client to control the flow of data (PAUSE/RESTART);

- **Sensor Data Gateway Microservice**: This is the entry point of any petition of providing real-time weather data. In offers a REST interface that the clients use to authenticate and start proving reports. Once they are authenticated, they receive a queue ID that they can use to provide reports to their assigned Sensor Data Agent;

- **Sensor Data Agent Microservice**: There is one instance of this microservice per sensor providing data i.e. one per client providing real-time weather reports. It reads from the AMQP queue in order to route those messages that are well formed to the appropriate sensor cell topics (explained below);

- **Node Manager Microservice**: This microservice is not visible by the clients but used internally by the Gateways Microservice to instantiate and terminate instances of Docker Containers of Client Agents, Data Brokers and Sensor Data Receivers as they are required. There is one instance per hardware server on the local cluster. When deploying in a cloud environment, this is substituted by a micro-service that uses the Azure API or AWS API to instantiate the container when instructed. In both cases the API offered to the Gateway is the same and it is based on REST services;

- **Message Brokers**: There are four instances of Message Brokers in the implementation of the weather service that, while identical in implementation, serve different parts of the data flow:

    - Weather Cells Broker: it is topic-based and serves to communicate the client agents with the Core Weather services. There is one topic per cell covered by the model. The model feds it periodically and the agent is subscribed to all the topics that contain data from areas they need to compose the data flow that the client requires;

    - Sensor Cells Broker: it is topic-based and serves to communicate the sensor data agents with the Core Weather services. There is one topic per cell potentially receiving sensor readings. Different processes of the Core Weather Service are subscribed to this topic and consume all data from them that applies to each run;

    - Client Outgoing Data Broker: it is queue-based and contains one queue per client. The corresponding agent feeds its queue according to the flow instructions given by the client;

    - Sensor Incoming Data Broker: it is queue-based and contains one queue per client. The corresponding agent processes all reports coming from its client, parses the data and places them in the corresponding Sensor Cell Topic.

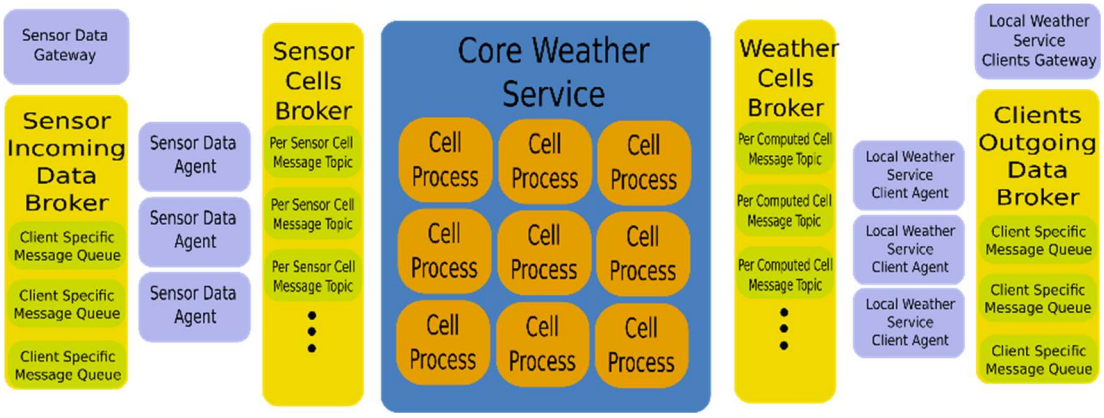

Figure 1- Service Prototype Model Exercise 1

Remarks, Notes, Guidelines
N/A

2.2 Model for testing flight planning management service

Availability Note by MEMBER(S):	Jeppesen
Availability Note Date:	25/07/2019
Person Responsible Item's Availability:	Anna-Lisa Mautes (Jeppesen)
Deliverable Code:	D4.1
Item's Name and Version:	Flight planning management model. Version 1.0
Details of Availability	
Step and Maturity Level addressed by the Item implementation:	V01 according to E-OCVM
Documents and their versions used for the Item's development:	• D02.02 Drone Information Services; • D03.01 Architecture and technical requirements.
Item's Verification related documentation:	D05.01 Experimental Plan
Date and location of the Item's Verification:	Jeppesen's premises, July 2019
Verification Overall Result:	The Flight Planning Management Service prototype has been successfully developed and tested using synthetic flight plan data, with approximately 500 flight plans located in an urban area over a 60 day timeframe, for checking conflicts with other flight plans stored in the database and notifying the client with an average response time of less than 0.1307 seconds.
Information on the Content:	

PROTOTYPE ARCHITECTURE

The main technologies used for the implementation of the service prototype are Spring Boot and Rabbit MQ using JAVA 8. The service prototype has been further deployed on the Windows Azure cloud environment. Figure 1 shows an overview of the architecture of the service implemented. There are four main components in this microservice-based prototype:

- **Flight Plan Microservice**: It processes flight plans that are requested for validation from clients and updates changes in their associated status. Additionally, it stores validated flight plans in a centralized database. It is implemented in Java Spring Boot 2.1.5, guaranteeing reliable synchronous communication through a REST API and MongoDB which enables fast access to the data;
- **Conflict Manager Microservice**: It validates flight plan trajectories against existing trajectories in the considered spatial and temporal frame, considering a specific separation minima and a defined prioritization system. It is implemented in Java Spring Boot 2.1.5;
- **Message Broker**: It queues flight trajectories requests for validation and prepares status

messages after the completion of the validation process. For this purpose, a Rabbit MQ Docker container from the docker public repository was utilized;

- **Notification Microservice**: It notifies subscribed clients about the result of the validation process and about emerging updates in the status of the pre-approved flight plans. It is implemented in Java Spring Boot 2.1.5. Asynchronous communication is done with STOMP over a Web Socket protocol.

Each service runs in a Docker container. The complete run configuration is maintained in a Docker-compose script.

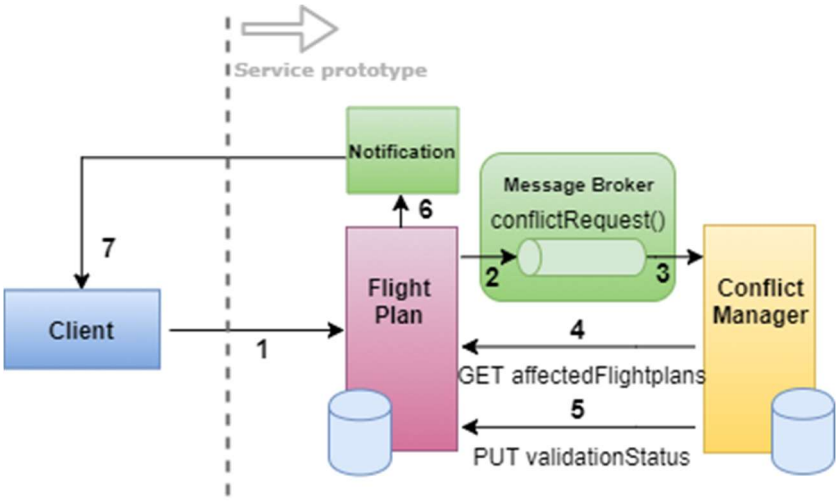

Figure 2- Service Prototype Model Exercise 2

DATA FLOW

Figure 2 also indicates the sequence order of the data flow in the service prototype model. The numbers labelling the arrows in the figure indicate the order in which the drone flight plan data flows in the nominal work flow process.

The external data input consists of drone flight plans including 4D flight trajectory (3D geometry and timing), operational context and drone operator information. The generated flight plans are made available as JSON files. Furthermore, the flight trajectories are modelled with Jeppesen's trajectory modelling algorithm.

Firstly, the generated flight plans are submitted through standard HTTP method to the Flight Plan Microservice (Flow 1). This microservice performs a syntax check-up to the received flight plan and assigns a flight plan ID, which is communicated to the client using the same communication protocol. Next, the extracted flight trajectory is forwarded to the Message Broker (Flow 2) where the message is queued until the Conflict Manager Microservice consumes the message (Flow 3). The Conflict Manager Microservice extracts the spatial and temporal boundary constraints of the flight trajectory and retrieves affected trajectories from the flight plan database (Flow 4). After concluding the validation process, the Conflict Manager Microservice forwards a status message to the Flight Plan Microservice (Flow 5). This microservice forwards the status message to the Notification Microservice (Flow 6) which publishes the message back to the client through a web socket connection (Flow 7).

Remarks, Notes, Guidelines
N/A

2.3 Model for testing monitoring and traffic information service

Availability Note by MEMBER(S):	INECO
Availability Note Date:	30/06/2019
Person Responsible Item's Availability:	Javier Espinosa Aranda (INECO)
Deliverable Code:	D4.1
Item's Name and Version:	IMPETUS platform and micro-service model
Details of Availability	
Step and Maturity Level addressed by the Item implementation:	V01 according to E-OCVM
Documents and their versions used for the Item's development:	• D02.02 Drone Information Services; • D03.01 Architecture and technical requirements.
Item's Verification related documentation:	D05.01 Experimental Plan
Date and location of the Item's Verification:	INECO's premises, June 2019
Verification Overall Result:	Verification successfully passed.
Information on the Content:	

PROTOTYPE ARCHITECTURE

This model aims at proving the benefits of a microservice-based architecture in terms of its performance and capacity to process large volumes of information from multiple sources and providing reliable information at short time (with a short delay) to end users. A set of requirements related to these specific functionalities have been identified and implemented using a microservice model-based architecture, containing the following modules:

- **Communications:** this module comprises the functionality of collecting the information provided by a drone operation based on MAVlink protocol (telemetry and MAVproxy as an interface with the message listener using LTE communications), and shows the final outputs of the platform sending them to a web client;

- **Message Listener**: It is in charge of detecting and collecting new messages, processing them to adapt the content to a specific format and sending them to a gateway to be distributed in the different microservices included in the platform;

- **Information layer**: This layer hands the information around the different modules implemented in the platform. The messages that are going to be distributed are tagged with a label, determining if they consist on a Position message or an Alert message;

- **Processors**: these are the micro-services included in this model. They are supported by an architecture that is capable of monitoring them, checking their status and executing different actions to ensure the performance of the system e.g. launching new instances of the services, balancing the load between them, etc. The key services implemented are:

    - **Mapper**: It processes the inputs from the operation and provides the position and time stamp associated;
    - **Traffic information management platform**: It is in charge of comprising all the inputs from the variety of operations, calculating their separation and providing alerts;
    - **User Interface:** It represents the previous information in a user-friendly environment that must be located in the operation location.

The proposed architecture is shown in the following picture:

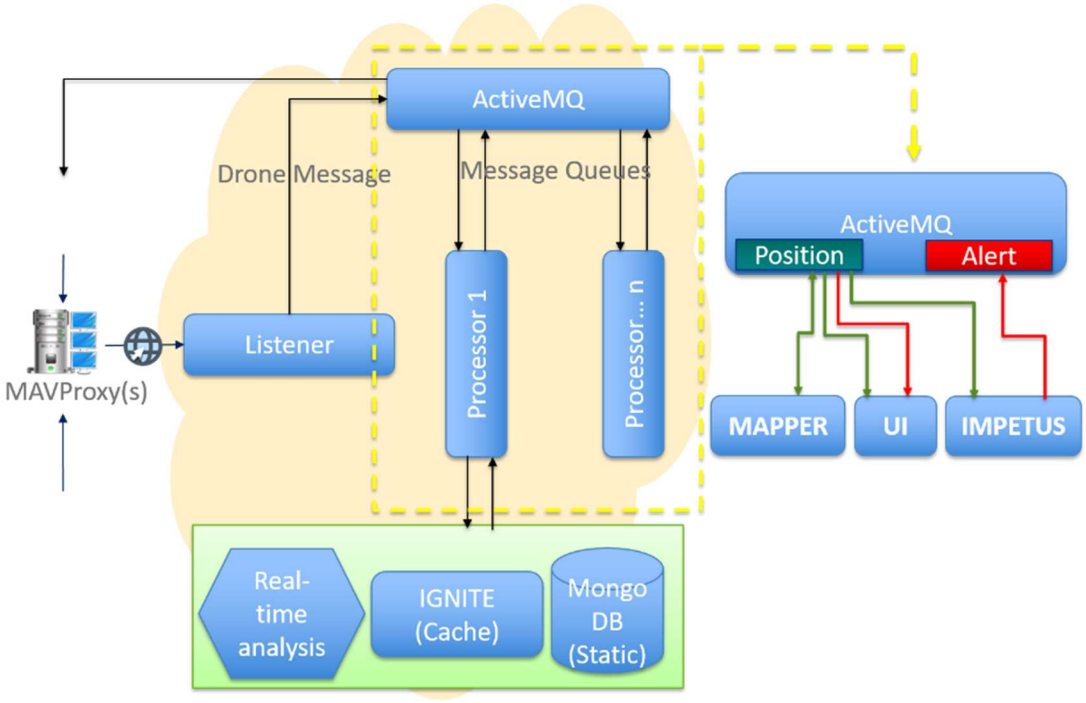

Figure 3- Service Prototype Model Exercise 3

The development of this architecture is based on open source applications e.g. IGNITE as a cache, MongoDB for static information, ActiveMQ as main handler of the information, and Openlayers as the GIS used in the User Interface.

Remarks, Notes, Guidelines

This exercise has been limited in scope to emphasize the benefits of a microservice-based architecture, addressing the analysis of the performance of these types of architecture and mimicking the expected behaviour of the U-Space environment, in which the communications between the Operator and the Orchestrator shall be ensured (using internet connection in this case) and the main platform to be used by the Authorities is integrated using cloud-computing systems.

2.4 Model for testing dynamic capacity management services

Availability Note by MEMBER(S):	Altitude Angel
Availability Note Date:	25/07/2019
Person Responsible Item's Availability:	Chris Forster (Altitude Angel)
Deliverable Code:	D4.1
Item's Name and Version:	IMPETUS platform and micro-service model
Details of Availability	
Step and Maturity Level addressed by the Item implementation:	V01 according to E-OCVM
Documents and their versions used for the Item's development:	• D02.02 Drone Information Services; • D03.01 Architecture and technical requirements.
Item's Verification related documentation:	D05.01 Experimental Plan
Date and location of the Item's Verification:	Altitude Angel's premises, July 2019
Verification Overall Result:	The marketplace discovery service was successfully implemented. The Tactical Deconfliction Service has been successfully developed and tested using synthetic environmental information and air traffic data.
Information on the Content:	

PROTOTYPE ARCHITECTURE

The main technologies used for the implementation of the service prototype are .NET (C#), SQL and Azure services. The service prototype has been further deployed on the Windows Azure cloud environment. Figure 4 shows an overview of the architecture of the service implemented.

The two services developed have a number of in their respective microservice-based prototype:

Marketplace Discover Service

- **Service Registration**: This service enables the micro-service provider to programmatically register their component within the market place. They must critically provide a category of micro-services e.g. U-Space Service Supplier, Supplementary Data Supplier… and additional service information relevant to a service consumer such that they can make a decision as to whether the service meets their high level requirement;
- **Service Discovery**: The service enables the consumer of the micro-service to discover (both programmatically or though UI) micro-services available that meet their requirement;

Tactical Deconflictor

- **Situational Picture**: This component enables the deconflictor service to build an up to date situation representation of the area in which the microservices is managing deconfliction, taking into account all available and relevant environmental and flight data;
- **Analysis**: This component enables the deconflictor to analyse the situational position to determine conflicts within a management area;
- **Command**: This component enables to issue commands to the drone or drone pilot to resolve any conflict and ensure separation.

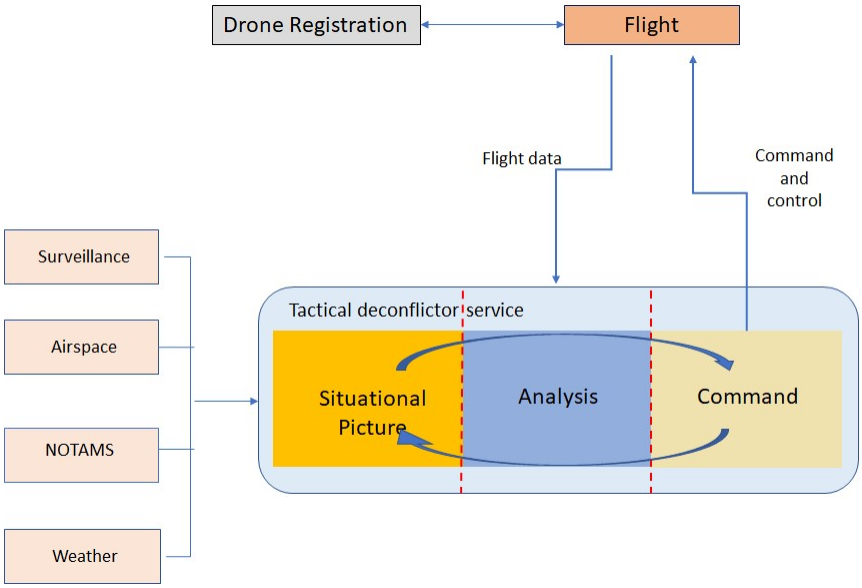

Figure 4- Service Prototype Model Exercise 4

Remarks, Notes, Guidelines

N/A

3 References

[1] D03.01 Architecture and technical requirements.

[2] D05.01 Experimental Plan.


BOEING


JEPPESEN


CRIDA
I·D·i · ATM


ineco

S


ALTITUDE
A·N·G·E·L


C-ASTRAL
AEROSPACE Ltd.


TECHNISCHE
UNIVERSITÄT
DARMSTADT