Towards a Service-Oriented U-Space Architecture for Autonomous Drone Operations

Miguel Campusano, Kjeld Jensen, Ulrik Pagh Schultz
SDU UAS, MMMI, University of Southern Denmark
Corresponding author: mica@mmmi.sdu.dk

***Abstract*—While the complexity of development and cost of Unmanned Aerial Systems (UAS) decrease, the use and market of UASs continuously increase. The benefits of UASs can however only be fully realized when they are allowed to autonomously fly Beyond Visual Line of Sight (BVLOS). To allow safe and efficient autonomous BVLOS flights, the European initiative SESAR came out with the concept of U-space to integrate UASs into the already populated airspace. There is however currently no open high-level software architecture that realizes the concept of U-space for autonomous BVLOS operations. In this work, we present an initial proposal of such a system, a service-oriented architecture named ROS-DOTS. We follow a modular design that supports the integration of services that can be modified and replaced independently. The benefit of our modular design is the use of high-level abstraction models for UAS operations, using modular services that are developed independently between each other.**

To show the potential of ROS-DOTS, we present a use case where several services in the architecture and a commercial UAS traffic management system work together to generate a flight plan for a UAS operation, that then can be retrieved from our system and uploaded into a UAS.

***Index Terms*—Unmanned Aerial System, U-space, Service-Oriented Architecture, Autonomous Drone Operations**

I. Introduction

The market for Unmanned Aerial Systems (UAS) is continuously growing as improvements in flight characteristics, ease of deployment, and maintenance costs enable applications in a wide variety of domains [1]. The full commercial potential can however only be realized with UASs flying autonomously Beyond Visual Line Of Sight (BVLOS) [2]. BVLOS implies several safety concerns, and measures must be taken to safely and efficiently integrate UAS operations into the existing airspace traffic. Such measures make the development of UASs more complex.

To address these challenges, the U-space concept was born, as a European initiative to provide high-level services for UASs. The SESAR Joint Undertaking drafted a vision on how to realize U-space [3]. While architectures such as DOMUS and SAFIR are built to demonstrate the U-space concept, there is no available open software architectures that links high-level U-space services to autonomous UASs, thereby providing essential services for BVLOS operations, such as planning and dynamic replanning of missions. In this paper we describe our vision for an extensible U-space service-oriented architecture for autonomous UAS operations.

Our previous effort. In a previous work we designed and implemented a Domain Specific Language (DSL) to program UAS operations, named Drone Operation Template Specification language (DOTS) [4]. With DOTS, the purpose of a mission can be expressed using several UASs such that generated flight plans for each UAS can be replanned when new constraints are specified (*e.g.,* new restricted flying areas, priority actors, weather uncertainty, etc). Our goal is to use DOTS to plan UAS operations in the context of the HealthDrone project [5]; planning the delivery of blood samples and medical goods using autonomous UASs. The existing implementation of DOTS is however lacking key software engineering qualities, such as modularity and modifiability, making it inflexible for use in the HealthDrone project. Moreover, the services that DOTS provides are highly coupled to the DSL, making it difficult to use them in other projects.

Contributions. In this work we describe ROS-DOTS, a modular architecture to support U-space services for a wide range of autonomous UAS operations using DOTS, such as logistics and precision agriculture. This architecture is experimentally validated by supporting the HealthDrone project in its integration of UASs into the airspace. Our architecture focuses on modularity, because U-space architecture will grow in terms of the services that are provided and in terms of the capabilities of the UASs and the regulations of the airspace. The services provided by the architecture will need to evolve to accommodate new regulations, new types of UASs, new types of operations, etc.

II. State of the Art

UAS Traffic Management (UTM) systems organize the use of the airspace for UASs in a uniform manner [6]. While this works for generic operations, some UASs need access to services that take into account the domain of the problem. For example, in the case of the Unifly UTM [7], in order to accept an operation, the operator must provide the path of the UAS. However, whenever a conflict is detected after the operation is submitted (*e.g.,* a no-fly-zone is added), the operator herself is in charge to update and re-upload a new valid path, taking into account the problem domain.

The SESAR Joint Undertaking defines a U-space as a set of services that allows a secure and efficient access of UASs into the airspace [8]. These services can be provided by third actors that need to cooperate between each other. The U-space architecture specifications also describes several solution examples: GOF USPACE, Swiss U-space, DOMUS and SAFIR [3]. Every architecture is described as a service provided with

ATM and UTM integrations, effectively extending the features of ATM/UTM systems, similarly to ROS-DOTS. However, they are not open to be modified by adding new services. We expect to open the source code of ROS-DOTS in the future, not only making it available for its use, but also allowing developers to integrate their own services in our U-space architecture.

Systems such as FLYAQ [9] provide ways to describe missions and generate flight plans for drones using its own DSL. The work of Besada *et al.* [10] is described as a drones-as-a-service architecture, where operators define UAS missions using a visual interface. These systems can be seen as providing one type of service: mission planning for UASs. Instead, ROS-DOTS provides several services that allow autonomous UAS operations, in particular mission planning for a fleet of UASs. Moreover, ROS-DOTS connects with UTM systems for reporting these flight plans, a key feature to allow BVLOS operations that is not presented in the mentioned systems.

A high-level service-oriented U-space architecture like ROS-DOTS must be complemented with flight controller frameworks, such as Paparazzi, PX4 and Aerostack [11]–[13]. They provide services to support UAS flights, such as uploading of flight paths as waypoints, remote control, ground control user interfaces and even simulations. However, unlike ROS-DOTS, necessary U-space services for safe UAS integrations into the airspace are lacking into these frameworks, such as mission planning with external constraints (*e.g.,* no-fly-zones, priority actors such as rescue helicopters) and UTM integrations. The services provided by ROS-DOTS can be used by these frameworks to increase the capabilities of UAS operations.

III. U-SPACE UAS OPERATIONS

In this work we presents ROS-DOTS, a U-space service architecture for UAS operations. Before presenting the design of the architecture, we illustrate how it can be used to perform UAS operations. To this end, we use the example of transporting medical goods in the HealthDrone project [5]. In particular, an operator uses a UAS flying BVLOS to transport medicine from a hospital to a nursing home.

Before the operation.

Before performing a UAS operation, the operator needs a UAS. The UAS needs to be registered in a UTM system before it can be used in any operation.

Moreover, there is a special fly zone over the city where the operation is going to be performed. This zone is a restricted area which no UAS can fly over, a zone called no-fly-zone.

Defining the operation.

To define the operation, the operator uses the DOTS DSL. The operation specifies a UAS, the hospital and the nursing home. It also stipulates how the UAS should fly. In this example, the UAS moves from the hospital to the nursing home. This DOTS template can be seen in Listing 1.

Finally, the operator uploads this template to the ROS-DOTS system. At this point, ROS-DOTS only saves the template. A template is an abstract representation of an operation represented with parameters (*i.e.,* in this case `uav`, `nursing_home` and `hospital`). Without the value of the parameters, the template cannot be translated into a flight plan.

```
utm healthdrone:
  import action straight<location>
  operation visit_nursing_home:
    uav: drone
    nursing_home: navigation_point
    hospital: navigation_point
    implementation:
        uav += straight(hospital)
        uav += straight(nursing_home)
```

Listing 1. A template written in DOTS, representing an operation that a UAS visits a nursing home from a hospital.

Planning the operation.

With the operation template uploaded into ROS-DOTS, the operator now needs to fill the parameters to generate a flight plan. The operator assigns a UAS that is already registered in a UTM. It also indicates the coordinates to the hospital where the UAS receives the medicine and the nursing home where the UAS should deliver the medicine.

When the operator uploads the data, the ROS-DOTS system returns the flight plan of the UAS. The flight plan not only specifies the points where the UAS should fly, but also the time when it should fly. In this example, the operator is in charge to start the flight plan of the UAS, and one key aspect of the plan is the time the UAS should be flying. Moreover, this flight plan can be retrieved for later use. The operator does not need to save the flight plan, but she can always retrieve the flight plan to upload it to the UAS to start the operation.

Finally, the flight plan is notified to a UTM. In ROS-DOTS, whenever a flight plan is generated, it is automatically uploaded to a UTM, removing this burden from the operator.

Changes before the start of the operation.

The operator is in charge of starting the operation of the UAS using the generated flight plan. However, this flight plan may change due to changes in the conditions when the flight plan was generated. For example, ROS-DOTS allows for uploading of new no-fly-zones. The new no-fly-zones added to the system may conflict with the already planned flight plans, and they need to be replanned to accommodate to the new conditions. This is automatically done by ROS-DOTS. Moreover, the flight plans are also automatically updated in the UTM.

Conducting the operation.

With the operation planned, the operator prepares the UAS and its system to start the operation. Before starting it, the operator retrieves the flight plan again (because of constantly changing conditions, the flight plan may have changed too) and upload the plan to the UAS. The operation has now started and the operator is in control of the UAS.

Figure 1 shows the configuration of the previous example. The hospital and the nursing home are marked in the map. The no-fly-zone A is added before planning the operation. This area is first avoided by the planner, as shown by the line PATH A. Then, a new no-fly-zone area B is added, and the

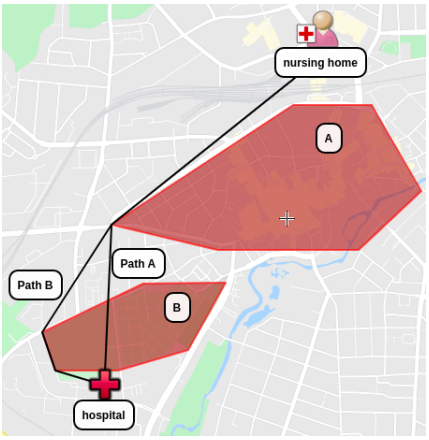

Fig. 1. Map configuration of the example presented in Listing 1.

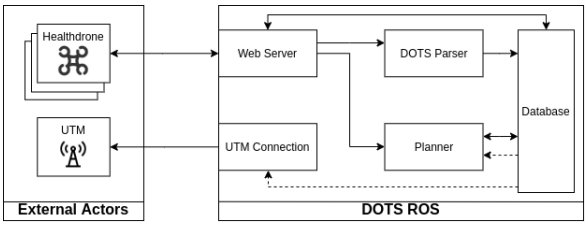

Fig. 2. ROS-DOTS architecture. Continuous lines represent direct calls. Dotted lines represent indirect calls.

system automatically update the original flight plan for the one represented by the line PATH B

IV. ROS-DOTS

In the previous section we showed an example of a U-space UAS operation: transporting medicine from a hospital to a nursing home using a UAS. For that example we use ROS-DOTS as a service-oriented architecture to provide U-space services. In this section we present the design of ROS-DOTS and the services it provides. All the services and how they are connected can be seen in Figure 2.

The main goal of ROS-DOTS is to provide several UAS services to allow for UAS operations in a U-space context. The UAS operations are represented using the DOTS DSL (as shown in Section III).

Moreover, there is a strict regulation for UASs doing BVLOS flights. In particular, it is not normally possible to do fully autonomous flights now but, as is being demonstrated in HealhDrone, this will change in the future when UAS technology is more mature and when there are more experiences of doing BVLOS flights. To support for this uncertainty in requirements, the modular design of ROS-DOTS supports services that allow for fully autonomous and non-autonomous UAS operations. This kind of flexibility is similar on how U-space is planned to be rolled out [8].

A. Service-Oriented Architecture

To allow our system to change and grow accordingly to the UAS systems and regulations, while still supporting a U-space style architecture, we designed ROS-DOTS as a service-oriented architecture that has support for extensible and interchangeable services. We focus our efforts on the following assumptions: (i) services may become obsolete when the system gets more mature; (ii) some services will need to be added without interfering with others; (iii) some services may be exchanged with other, more sophisticated services.

We built ROS-DOTS from the ground up using our previous experience on DOTS [4]. While DOTS provides services for UAS operations, like a planner service, the system uses a code generation strategy to build services for each program, making every service highly coupled to each other and the program. This decision brings several undesirable consequences, such as not being able to manage different templates and flight plans, and not being able to add new services without being certain to not affect the rest of the architecture. The modularity of ROS-DOTS solves the software quality problems of DOTS, allowing us to have a modular system that can grow in terms of the services that provides and that can be used with by other UASs.

B. Service Interoperability

Services may be called by any other services in two ways: direct or indirect calls. A direct call follows the Request/Response pattern, where a service calls another and waits for a response. In the example presented in Section III, this pattern is presented in several calls, including when a developer uploads a DOTS template and when she uploads variables and receive the flight plan as a response.

An indirect call follows the Publish/Subscribe pattern. When a service is interested in certain data, it subscribes to the data channel, while other services publish these data, without directly calling the interested services. In Section III, when a developer uploads the parameters, before receiving the flight plan (request/response), the flight plan is saved in the system. When this happens, the system publishes to all interested services that there is a new flight plan. This allows for a particular service to react and publish the new flight plan to the external UTM.

C. ROS Integration

Similar to the case of Aerostack [13], we decided to use ROS [14] as a base framework for ROS-DOTS. Using ROS, we expect to more easily share ROS-DOTS to the robotic and UAS communities. Every ROS-DOTS service is represented as ROS nodes (*i.e.,* ROS components), that run independently between each other. Each service can be grouped and executed using a launch file (*i.e.,* a configuration file). This set up allows for every service to run in parallel. Due to this design, ROS-DOTS is highly distribute. Furthermore, ROS topics, services and messages (*i.e.,* ROS communication protocol between nodes) is used to exchange data between services. The endpoints of every service depend on each service and they are represented with ROS topics and services depending on the type of call (direct or indirect). The contract of each endpoint are represented using ROS messages.

D. Services

We implemented several key services for U-space UAS operations that show the potential of this architecture in the example presented in Section III.

1) Internal Database: In Section III we showed how an operator may retrieve the flight plans generated by the system at any moment, and how no-fly-zones are taken into consideration when generating flight plans. This is due to an internal database service for ROS-DOTS, where all necessary data is saved and can be retrieved by other services. This service supports call to save and retrieve all data necessary for the system to work: DOTS templates, flight plans and no-fly-zones. Moreover, whenever a data is saved, it publishes back to the system for any interested services.

2) DOTS Parser: In a previous work, we presented DOTS, a DSL to express high-level UAS operations [4]. In this work, we use DOTS as a base to generate flight plans for UASs, and a service is needed to parse DOTS programs. The *DOTS Parser* service receives a DOTS template represented as a XML and converts it to a metamodel representation of the template as an Abstract Syntax Tree (AST), which is returned to the caller and saved into the database service.

3) Planner: The planner service takes a DOTS template and its arguments to plan flights for every UAS specified in the operation. This service also takes into consideration no-fly-zones and the already planned flights. The details on how the planner works are outside the scope of this work. In the future we expect to update this service with a more efficient planner. Nevertheless, the service-oriented architecture allows to plug-in any planner that follows the same interface. This service is also subscribed to the added no-fly-zones. In this case, whenever there is a new no-fly-zone, the planner replans every conflicted flight plan.

4) UTM Connection: Every flight plan should be notified to a UTM service to allow UASs in the air. A service is in charge of connecting with a UTM whenever new flight plans are generated. As was the case in our previous effort [4], in ROS-DOTS we connect to the Unifly UTM [7]. To notify the newly added flight plans, this service is subscribed to the database service, which published to the system the saved flight plans.

5) Web Server: This service is used for UAS operators to access the services inside ROS-DOTS. This service provides endpoints to call 3 internal services: the parser, the operation planner and the internal database. For the internal database, this server service provides endpoint to upload and retrieve no-fly-zones and flight plans. This service allows ROS-DOTS to provide its services to UAS.

V. Conclusion and Future Work

In this paper we presented our architecture to support autonomous UAS operations under the concept of U-space, ROS-DOTS. This architecture uses the DOTS DSL to specify high-level UAS operations. The strong focus on modularity of the architecture allows services to grow independently. We present our architecture with a U-space UAS operation example: generating a flight plan for a UAS transporting medicine from a hospital to a nursing care center.

While the service provided by ROS-DOTS are key in a U-space architecture for UAS operations, they are not enough for fully autonomous operations. More services to monitor flying UASs and to generate more secure flight plans are required. Monitoring flying UASs is key to have a safe airspace, UASs however still may collide if they do not change their flight paths dynamically. To do that, we expect to extend ROS-DOTS with at least two more services: Dynamic replanning, and Detect and Avoid (DAA). The current replanning of an operation only occurs when the operation has not started yet, it is also necessary to replan on-going operations if external conditions changed (*e.g.,* new no-fly-zones, prioritized aircrafts). The service for DAA should give recommendations to operators when dangerous situations are about to happen, such as a UAS in a collision path to an aircraft.

Acknowledgment: This work is part of the HealthDrone project funded by Innovation Fund Denmark.

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2909530

[2] P. Butterworth-Hayes and T. Mahon, "The Market for UAV Traffic Management Services 2021–2025." [Online]. Available: https://www.unmannedairspace.info/uav-traffic-management-services/

[3] Single European Sky ATM Research Joint Undertaking, "Initial view on Principles for the U-space architecture," 2019. [Online]. Available: https://www.sesarju.eu/node/3402

[4] M. Campusano, N. Heltner, N. Mølby, K. Jensen, and U. P. Schultz, "Towards Declarative Specification of Multi-Drone BVLOS Missions for UTM," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2020, pp. 430–431.

[5] HealthDrone, 2020. [Online]. Available: https://healthdrone.dk

[6] Global UTM Association, "UAS Traffic Management Architecture," 2017. [Online]. Available: https://gutma.org/utm-architecture/

[7] "Unifly." [Online]. Available: https://www.unifly.aero/

[8] Single European Sky ATM Research Joint Undertaking, "U-space Blueprint," 2017. [Online]. Available: https://doi.org/10.2829/335092

[9] D. D. Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli, "Automatic generation of detailed flight plans from high-level mission descriptions," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, 2016, pp. 45–55.

[10] J. A. Besada, A. M. Bernardos, L. Bergesio, D. Vaquero, I. Campaña, and J. R. Casar, "Drones-as-a-service: A management architecture to provide mission planning, resource brokerage and operation support for fleets of drones," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 931–936.

[11] G. Hattenberger, M. Bronz, and M. Gorraz, "Using the paparazzi UAV system for scientific research," in *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, 2014.

[12] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[13] J. L. Sanchez-Lopez, M. Molina, H. Bavle, C. Sampedro, R. A. S. Fernández, and P. Campoy, "A multi-layered component-based approach for the development of aerial robotic systems: the aerostack framework," *J. of Intelligent & Robotic Systems*, vol. 88, no. 2-4, pp. 683–709, 2017.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.