TRANSNAV the International Journal on Marine Navigation and Safety of Sea Transportation
http://www.transnav.eu
Volume 15
Number 1
March 2021
DOI: 10.12716/1001.15.01.15

A Web-oriented Architecture for Deploying Multiple Unmanned Vehicles as a Service

C.N. Au, C. Delea, V.E. Schneider, J, Oeffner & C. Jahn
Fraunhofer Center for Maritime Logistics and Services, Hamburg, Germany

ABSTRACT: Providing a robotic-assisted service in scenarios involving multiple Unmanned Vehicles (UVs) in possible beyond-visual-Line-Of-Sight (LoS) operations, safety and security are critical concerns. We develop a web-oriented, human-in-the-loop infrastructure to explore how the service provider can secure their system, enforce instant access control over dynamic operator-robot connections, and ensure the integrity, availability, and traceability of communicated data. Our proposed minimal viable solution requires an authentication server to verify user identity, a back server with a database to handle user requests and state-transition events, and a RabbitMQ (RMQ) server to trace the origin of data.

1 INTRODUCTION

The security aspects of robotic research generally receive much less attention, which can be partly reflected by the lower number of search results related to "robotic security" as shown in Table 1. As robotic systems become more powerful and easily accessible to the public, the lack of security awareness provides attackers with growing opportunities to cause massive damage to lives and properties. The situation is further exacerbated if we incorporate Robots-as-a-Service (RaaS) concept and introduce more user types and complexities to the system. Our work hence attempts to provide a viable solution that fulfils both the need for a secure system and the need for online delivered robotic service.

Table 1. Search terms entered to Google Scholar and the respective number of returned results.

Search terms	Number of results in thousands
robotics	2150
robotic	1770
robotic system	1520
robotic control	1480
robotic artificial intelligence	440
robotic security	153

Teleoperated UVs have been existing for over half a century and are traditionally deployed in extreme environments such as nuclear plants, deep-sea, and in space [13]. With the latest advancement in capabilities of UVs, teleoperation is getting more involved in more diverse use-cases to assist human operators [3, 12]. As the number of scenarios where UVs could be utilised grows, we envision the emergence of a market spectrum that offers various kinds of robotic services to customers [21]. These services will likely be accessible through the internet via standard browsers to allow for easy access of a wide range of possible users [19]. On the other hand, up-scaling these

systems will raise challenges related to performance, security, and safety regulations [26]. The security aspect of robotics research is often neglected, giving attackers chances to take over and potentially cause physical damage [9]. The Robot Operating System (ROS), for example, though widely used in robotics research, is not secure enough to be accessed publicly unless a series of security hardening is done [10].

As an effort to overcome these problems, we propose a web-based, service layer that is built upon mature open-source software. There are currently multiple implementations still under active development. An early version of our service layer is implemented in the Robotic Vessels as-a-Service (RoboVaaS) project, which aims to create a direct communication channel between the port customers and professionals that perform five oceanographic services, such as quay-wall inspection and environmental data collection [23]. In the European Union (EU) project Shipping Contributions to Inland Pollution Push for the Enforcement of Regulations (SCIPPER) [24], the service layer is named Environmental Shipping Monitoring Center (ESMC) and focuses on collecting data from environmental sensors and provides a web interface for visualising emission information. In the Horizon 2020 funded project SEarch, identificAtion and Collection of marine Litter with Autonomous Robots (SeaClear), the service layer interfaces with a team of heterogeneous robots (air- and waterborne) and assists with marine litter clean-up operations. In another EU project Risk-aware Autonomous Port Inspection Drone (RAPID), the service layer is officially called Command and Control Center which instantiates a customer service-request web interface and the operator mission-support interface. In general, these implementations are designed to be working closely as an integrated environment with managers, engineers, operators, and clients, who expect to track progress and analyse results of services involving off-shore electronic or robotic systems without the need to understand intricate details. On the other hand, the online delivered robotic services are not focusing on creating external interfaces for directly commanding robotic systems.

2 APPORACH

We adopted a microservice architecture and split the whole system into several computational components which communicate with each other through a set of well-defined and transparent Application Programming Interfaces (APIs) over Hypertext Transfer Protocol (HTTP) protocol. In this way, each microservice can focus on its core functionalities and offload computation to other responsible services. For example, the encryption and decryption of authentication token are handled solely by the authentication server. When the back server processes a user request, it can simply call the decode API of the auth server to retrieve the user's identity information from the token, without the need to implement another authentication algorithm. Moreover, as explained in [8], by decoupling the overall system in microservices, various cloud-based services can be exploited to fulfil the requirements of the different users involved in operating complex robotic systems. The reliability overall is also enhanced since the crash of a particular microservice does not result in a complete system meltdown.

We use a centralised approach to manage overall system states and access control policies of data channels. Although peer-to-peer communications in a decentralised setup have less overhead and fewer dependencies, we believe the service provider should possess ownership of data channels and have the power to immediately alter the system state in case of malicious activities or emergencies. By accessing the broker services the service provider can also verify the source and time of any communicated data.

To reduce duplication of effort, we strive to reuse as much mature and open-sourced technologies as possible. For example, service logic is mainly written in Javascript and run in NodeJS servers. User data and state information are stored in a PostgreSQL database. This approach allows us to focus on tackling logic that addresses problems specific to a particular use case.

Interfacing the service layer with other robotic systems requires services to translate data. For example, Rosbridge [6] is required to convert data between the ROS message format and the AMQP or WebRTC protocols. Another issue associated with ROS middle-ware or Data Distribution Service (DDS) is that they do not offer mechanisms to force an established communication to hang up. Due to these incompatibilities, our approach is to create a bridge node and treat it as a single monolithic vehicle.

3 SYSTEM FEATURES

The aspired service layer serves to provide business logic and to handle data according to a large set of different use-cases for various sectors and applications. Our implementation supports the following functionalities or features:

– Accessibility through internet: The service layer acts as the frontier facing end-users (or clients) and should be publicly accessible as a website. It should follow the latest security standards while leveraging the capabilities of modern browsers. For example, communication channels should conform to secure protocols such as Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), to prevent eavesdropping of network packets and to ensure the integrity of data. To mitigate vulnerabilities related to Cross-Origin Resource Sharing (CORS), we deploy a front server that is in essence an Nginx-based reverse proxy server, to tunnel communications between client and servers. In this way, the front server acts as a single point of contact (SPOC) and requests intended for the back server and authentication server are distinguished by different Uniform Resource Locator (URL) prefixes.
– Optimised resources allocation: When the number of users, UVs, and operators increase, efficient allocation of resources becomes a complex task. To resolve potential conflicts, tools such as mission

planners or booking systems have to be made available. While these tools could propose optimal solutions to users, the back server always has the final say on whether to proceed with a request.

- Distinguish stakeholders: While a typical robotic system mainly focuses on interfacing with operators and robots, the service layer also interacts with external users such as clients, guests, and authorities. Such a big variance of audience results in vastly different data needs and should be addressed accordingly. The data is traced to the producer by checking the binding status of RMQ's queues and exchanges.
- Horizontally Scaled: The architecture should be able to scale up to support a large number of customers, operators, robots, and ongoing missions simultaneously. It should include mechanisms to enable switching of robot control from one operator to another, so to maximise the up-time and flexibility of the service. In our implementation, the RMQ service supports expansion into a node cluster to support heavier traffic. Similarly, the NodeJS back server can be duplicated so that all requests are load-balanced through the front server. The database act as a single source of truth (SSOT) and is responsible to resolve any conflicts resulting from multiple concurrent transactions.
- Supplying lean information: Following the principles of Lean Product Development [16], the service layer should eliminate unnecessary data transfer and focus on delivering data that the user values. This is especially important given the high cost of bandwidth when operating across the web. A responsive interface and smooth user experience should be achieved through data post-processing techniques such as filtering, indexing, sub-sampling, and compressing.
- Reporting live data: Live data, such as robotic system position and video footage from on-board cameras not only enhance transparency and user experience but also provides managers and regulatory bodies with timely information necessary for monitoring and decision making in case of exceptional or hazardous events.
- Supports human in the loop: Although UVs are increasing their level of autonomy, human judgment is still necessary to perform some safety-critical procedures or to deal with situations when the necessary context is difficult to be captured and reasoned by a computer [5]. For instance, the service layer should prompt the responsible user for responses when processing safety procedures such as risk assessments, flight plan modifications, and traffic rerouting.
- Complements teleoperation: Modern browsers are capable of utilising sensors of mobile devices and interact with the human in complex ways. With an abundant supply of JavaScript libraries and their native support for WebRTC protocol, browsers are becoming an attractive choice to be used as an operator interface. Such an interface can be directly incorporated into the service layer as a tool for remote operators.

4 SYSTEM DESCRIPTION

The service layer is a bundle of components that can be tailored to handle a wide range of use cases. By using Service-Orientated Architecture (SOA) the native software solutions of the electronic and robotic systems can conveniently report their operations and leverage the data analyses to dedicated services. Following the Separation of Concern Principle [7], the service layer is separated into modules of distinct functionalities to enhance maintainability and readability. This section walks through the roles of these components and how they can be incorporated into a complete system.

4.1 *Component Overview*

The core of the service layer comprises a mission management system and a user management system. Whereas user management is handled by the authentication server, the logic concerning state changes and mission management is implemented in the back server. To communicate data among the robotic systems and various stakeholders, data brokers such as RMQ and Janus Gateway [1] are used. Network logs are dumped into a data lake for future analysis. The major components are summarised in Table 2.

Table 2. A list of typical system components in the service layer.

Component	Description
Database	Structured file systems that support SQL query and store important information such as the current system state, post-processed mission data, and metadata of compressed binary files.
Authentication server	A server that provides REST API related to user management such as user creation and verification of single-sign-on tokens.
Back server	A server that handles high-level requests from users and authorizes the subsequent system state transitions.
Front server	A proxy server that accepts all external requests and forwards them to the target servers. This mitigates vulnerabilities related to Cross-Origin Resource Sharing (CORS).
File server	A server that stores and serves files encoded in binary formats.
Data brokers	Servers that handle many-to-many communications by relaying data between users.
Browsers	Provides graphic-user-interfaces to enable user interacting with front-server.
Data-lake	Directories which store unstructured raw records such as network logs and sensor dumps.

4.2 *System Integration*

The service layer can be integrated with a robotic system, as depicted in Figure 1. This system is proposed to handle situations where a swarm of UVs needs to perform inspection work in an off-shore situation. The whole system can be classified into three parts that are spatially distinct from each other: Off-shore, on-shore, and remote. In the off-shore network, a team of UVs shares a local wireless

network with local operators. While the communication within the local network is stable, internet access becomes limited and unstable. The service layer resides in the on-shore network and can interact with off-shore machines. The off-shore system and the service layer can interface at a bridge node that translates information to achieve compatibility. The remote network usually contains users whose presence is not critical to the ongoing mission, such as the client and authorities. If the infrastructure supports enough bandwidth, it is also possible for an operator to command a vehicle remotely.

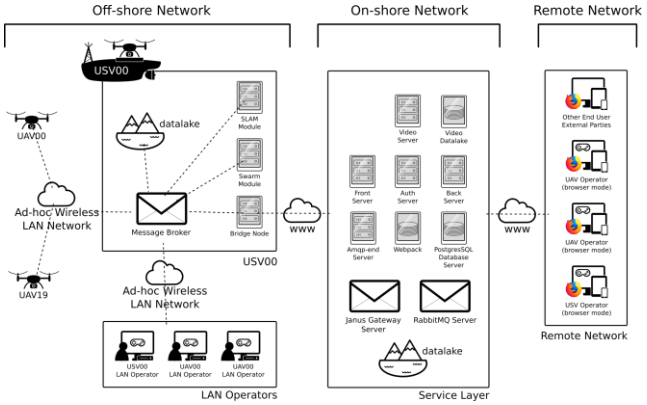

Figure 1. An example layout where the UVs perform a bridge inspection mission while the service layer is deployed on-shore. Note that a robotic network is deployed on a USV (Unmanned surface vehicle) offshore and can directly be accessed by operators in close vicinity even when the connection to the service layer is lost. Only essential data is exchanged between the offshore and onshore network, such as requests to create the mission, beacon signal and front camera video from UVs, and requests to be permitted by air traffic control authorities.

4.3 *User Management*

As an indispensable part of the service layer, the user management system implements a simple role-based access control mechanism [22] to enable proper sharing of resources among all users. The service layer provides a representational state transfer (REST) API and a web-based graphical interface to handle requests from all users. To access the system, users must first register themselves and obtain an assigned role from the administrator. All subsequent requests will then be handled according to the user role of the requester. A user always has one of the following roles:

- Admin: Administrative users who can edit the user role and approved use-cases for all other users.
- Client: End users who request services. They can view and comment on missions created by them.
- Planner: Users, who can assign missions to supervisors and plans the itinerary of a mission.
- Supervisor: Users who conduct a mission and are responsible to authorise and manage connections between operators and UVs. They can oversee the status of robots during the mission and can modify itineraries when necessary.
- Operator: Users who have direct control over robots.
- Robot: Accounts that represent robots. Only registered and approved robots can access and interact with the service layer.
- Guest: Users with limited read-access to specific missions.
- Inactive: The default role of a new account. No interactions are allowed.

Apart from the essential admin, client, operator, and robot roles, the rest of the list should be customised to suit different project needs. As the system consists of many separate modules whose logic requires authentication, an authentication server is used to issue and verify SSO tokens [20]. Users only need to input a password once to obtain a secret token, and then attach this token to all subsequent requests. Modules handling the requests then ask the authentication server to decode the token.

4.4 *Mission System*

The service layer encapsulates and presents information to users in terms of mission IDs. Each mission usually contains meta-information listed in Table 3.

Table 3. The information a mission usually contains.

Item	Description
Use-case	The business use-case requested by the client.
Geofence	A single connected space that all UVs should stay within.
Points-of-interest	Objects or locations of end-user's interest.
Itinerary	A list of tasks that specifies the time, location and actions to be carried out.
Risk assessment	Details on risk assessment.
Sensor data	Collected sensor data.All sensor data of business value must reference a unique and immutable mission ID.
Mission state	A mission must always be in one and only one of the states specified.

The life-cycle of each mission is captured by the mission state, which also provides users with a clear insight into the overall progress. The possible mission states are as follows:

- Planning: The initial state after creation.
- Filed: The risk assessment is submitted.
- Approved: The mission is ready to proceed to operation.
- In-progress: The supervisor checked in the mission.
- Interrupted: Some unplanned events occurred or extra action is required, e.g. beacon signals lost, traffic collision warning.
- Reviewing: All operators have checked out the mission.
- Closed: The client accepted or rejected the results of a service.

The list could be different for different use-cases and should ideally be presented as a finite state machine for more robust planning. For instance, in the RoboVaaS project, the mission system does not have a filed, approved, and interrupted state, since the system does not include Unmanned Aerial Vehicles (UAVs) and has more relaxed safety procedures.

4.5 *Data Brokers*

To enforce a dynamic access control policy on all data links, robots and operators are ideally bridged via centralised data brokers [18]. For example, in the RoboVaaS project, we use RMQ as the sole data broker that relays sensor data and command signals between operators and Unmanned Surface Vehicles (USVs) [23]. Changes of data link are invoked by channel change events. For example, to access the USV operators first issues a mount request to the service layer, which will then establish a communication link only if both the operator and USV are not booked in another mission or service. Mounted operators can then check in a mission so that the service layer associates the sensor data with the specified mission ID. As the broker is centrally managed by the service layer, it provides the possibility for a supervisor to force changes on communication channels to handle undesired situations, such as when a connection is dropped from a remote operator.

To ensure data traceability and integrity, we set up the RMQ server in a way that users can only access exchanges and queues beginning with their usernames. Therefore an operator cannot access the robot without first asking permission from the service layer. After a channel is established, the back server can trace sensor data back to the operator and UV by checking the binding status. This mechanism provides two advantages. Firstly the UV does not need to bother including their user ID in each sensor data, and secondly, attackers cannot spoof their identity by sending false user ID.

Since the web layer can be deployed on the public network, the Advanced Message Queuing Protocol (AMQP) is used to handle TCP traffic also due to its strong security features [17]. For UDP-based packet, Janus Gateway [1] will be used to relay WebRTC-based [14] traffic between browsers and the service layer.

The events that cause a change of routing of data are channel change events. These events should be handled with extra caution as they affect the data broker, hence ultimately who can control a UV. The two types of common events are:

- Mount and Unmount events: These events cause the creation or destruction of communication between an operator and a UV.
- Check-in and Check-out events: These events cause the creation or destruction of communication between a UV and the database.

Upon processing mount and check-in events the back server should make sure the resource requested is available, the requester has sufficient rights and the changes are correctly executed.

4.6 *Software Stack*

By using mature software libraries shown in Table 4 we could focus on implementing the logic related to specific use-cases and take advantage of the latest security features. For example, the load balancing feature of Nginx facilitates horizontal scaling in case we want to add more NodeJS servers to support more concurrent users [4]; OpenLayers draws dynamic and interactive maps on webpages [11]; The D3.js library provides a range of tools for visualising big data [2]. To ensure good code readabilities and to keep the development time short, the back server and authentication server are implemented using NodeJS. Unit-tests and end-to-end tests are written in Python and conducted by the Selenium Webdriver. We use PostgreSQL as the database to memorise user data, all system states, and sub-sampled sensor data. Concerning video data, the Gstreamer library [25] is used to build a pipeline, in which video data streams are encoded on the client-side using the H.264 compression standard [15] and then sent to the Janus Gateway server.

Table 4. List of major software used.

Component	Technology
Auth server	NodeJS
Front server	Nginx
Back server	NodeJS
Database	PostgreSQL
TCP-based broker	RabbitMQ
UDP-based broker	Janus Gateway
Browser	ReactJS, HTML5
Robot	Linux, Gstreamer

4.7 *Graphical User Interface*

The graphical user interface (GUI) is web-based and can be loaded to modern browsers as a web page. The interface allows user to navigate around and interact with the following elements:

- Login Page: Where the user authenticates by providing the username and password. As shown in Figure 2, the login page should stay as minimal as possible so sensitive information does not easily leak to unintended users.

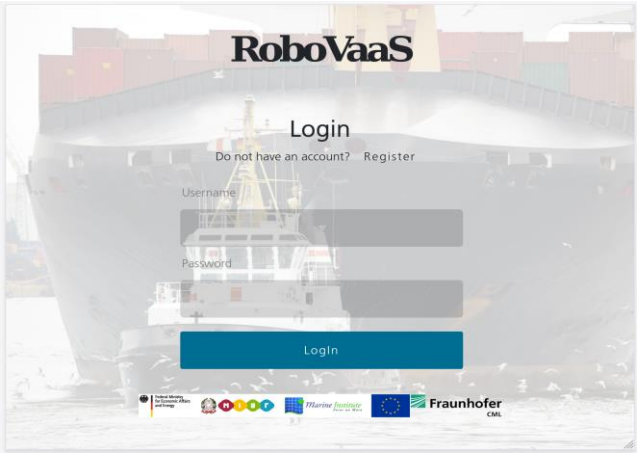

Figure 2. An example login page.

- Dashboard: This section acts as the main page and appears right after a successful login. It should provide the user with a clear and succinct overview of overall progress. In the example dashboard for operator shown in Figure 3, a list of active missions, the robot currently mounted, and the credentials for accessing the RMQ server, are shown.

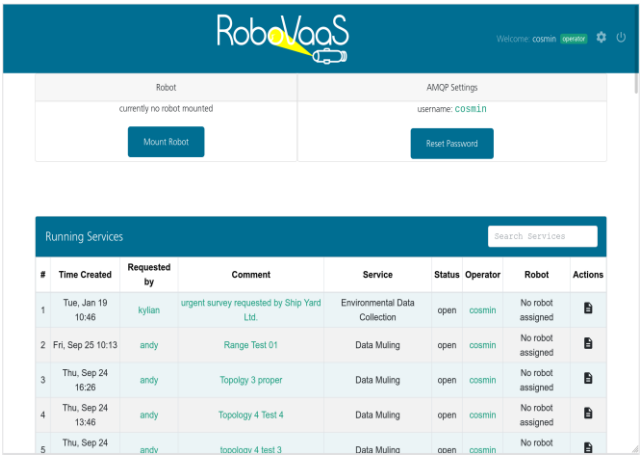

Figure 3. An example dashboard.

– Accounts Page: All the user information is displayed here. By default, all users can view their account information. Besides, some account types can view other accounts. For example, the accounts page seen by an admin user is shown in Figure 4. The page not only displays all other registered accounts but also allows the admin to further edit the user roles and allowed use-cases. A supervisor, on the other hand, can only see the information of all registered robots.

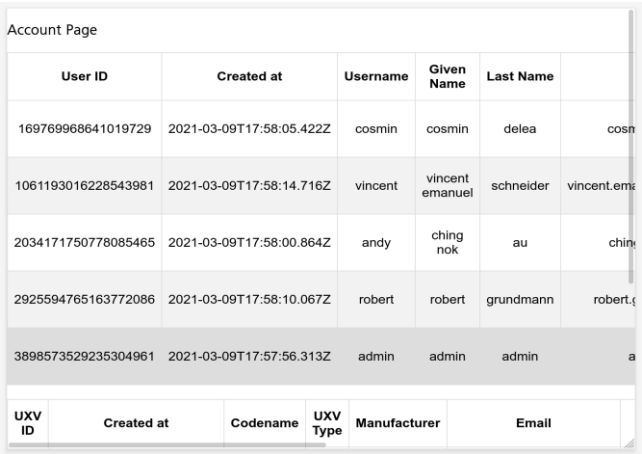

Figure 4. An example accounts page.

– New Mission Page: To create a mission, a client should first fill in details such as the desired use-case, points of interest, and geofence, on the new mission page. Then a summary page is displayed and the user can confirm and submit the request officially.
– Mission List: The mission list is a table containing basic information of missions accessible by a user, such as the name of the author and current mission state.
– Mission Detail Page: All the detailed mission information such as sensor data, live video stream, and discussion histories, is displayed on this page. In the example shown in Figure 5, some bathymetric sensor data collected during a RoboVaaS mission is visualised on a 2D map. The data is clustered and color encoded for easier perception. This page is also supposed to incorporate advanced functions such as report generation, file management, and data analysis.

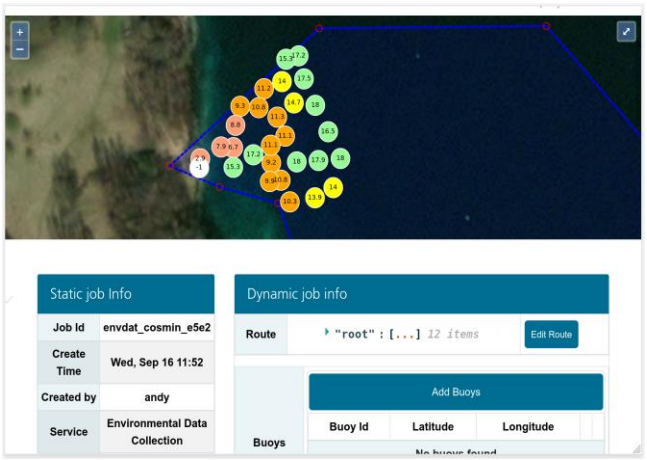

Figure 5. An example mission detail page.

5 PROGRAM FLOW

This section describes the communication flow between system components. Figure 6 illustrates how the front server act as a proxy and relays an authentication request between a user and the auth server. As the returned token is only decryptable by the auth server, other servers need to ask the auth server to verify a user's identity when they receive a new request containing a token.

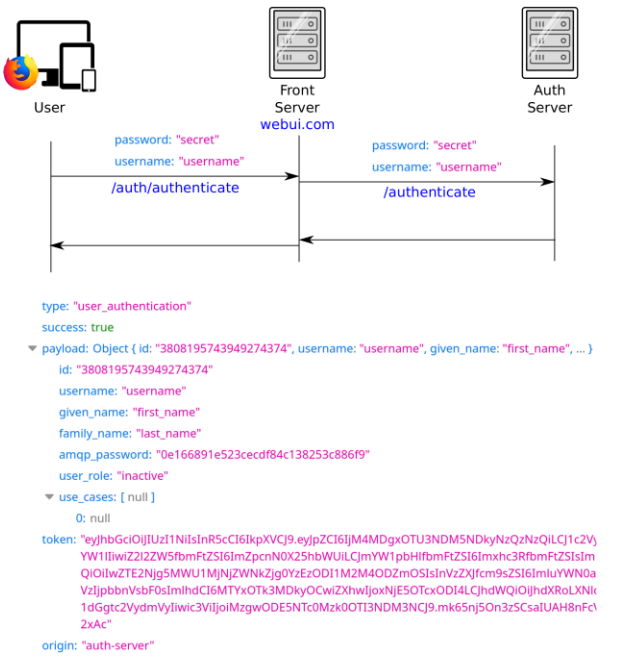

Figure 6. SSO sequence diagram. An example user sends to the front server an HTTP request, which has a JSON-formatted body with the credential. The front server then proxies the request to the auth server, which verifies the credentials against the record in the database. If it is valid, the auth server creates a payload, encrypts it into a token, and sends them back to the user.

Figure 7 illustrates how the service layer can be coupled with a data broker. Data brokers usually have their own ecosystems and hence an independent user management systems. To reduce the overhead for users having to manage two accounts, most account management is performed by the service layer on

behalf of the user. Take RMQ as an example, the username of the data broker account is always identical with the service layer's username. The password of the RMQ account is randomly generated by the service layer, stored in the database, and is sent to users when they successfully authenticate. To avoid a user tampering with resources, by default they can only influence exchanges and queues whose names start with the username.

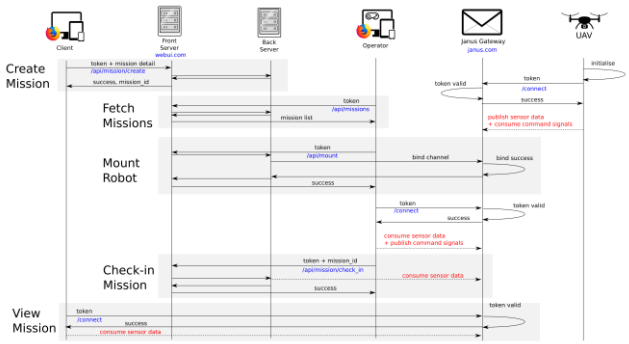

Figure 7. Sequence diagram to illustrate steps to broadcast live data via RMQ service. Unlike the service layer token that expires, The sensor module and user can always access RMQ service as long as they remember their RMQ credentials. In this example we defined an RMQ exchange called "beacon" to broadcast live sensor data from everyone. To emit a beacon message, the sensor simply publish a message with RMQ header "{'type':'beacon','codename':'sensor name'}". To avoid spoofing, messages without a conforming header will be discarded.

Figure 8 depicts a minimal example of how users collaborate on a mission. If the service requires, extra steps and more users might be involved. For instance, RoboVaaS requires a broker user to assign an operator to a mission before the operator could view and check in to this mission. If more than one UAV are involved in a mission, the mission can only be listed after receiving permission from an approver.

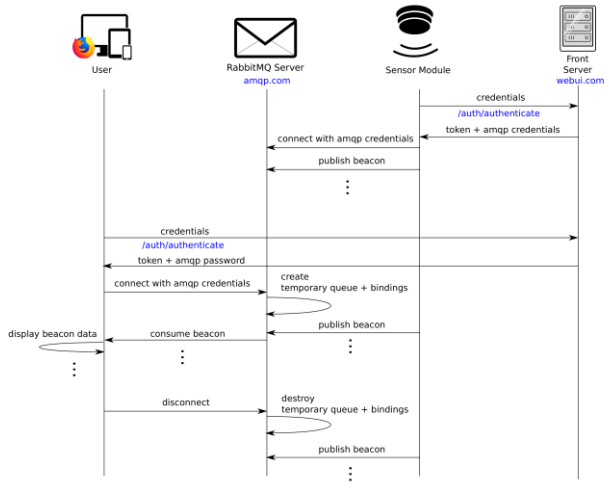

Figure 8. Sequence diagram to illustrate steps to store sensor data from a drone to database. First, a mission id is generated when a client requests a service. Depending on the service, concerning parties can view the mission and edit it accordingly. An operator can access and control the robot once they obtained mount permission from the back server, but the generated data is still not associated with any mission before a check-in event. After the operator successfully checked in a specific mission, the back server starts to read sensor values and store them under the mission id. The client who owns the mission will also receive data via the broker.

6 EXPERIMENTAL RESULTS

The service layer is deployed and tested with real and simulation data. The simulation environment is used to simulate the multi-body dynamics of the UVs used in project RoboVaaS and to connect the higher-level control system to the business logic enforced by the service layer. The output generated by the partner applications that command the UVs is sent over Transmission Control Protocol (TCP)-based connections over RMQ broker. The operation frequency is 10 Hz, the size of control messages not exceeding 9.6 kbit, while the video stream consumed up to 5 Mbit/s.

To transmit simulated data an internally wired network was used, while a wireless network was deployed for connecting the robotic system to shore. The service layer was deployed on a Dell Latitude 5411 laptop running Ubuntu 18.04 (Desktop). Besides handling the live data for the web-browser clients, the service layer needed to enforce the business logic needed for the operators to safely reserve and command an USV.

The service layer is capable of handling multiple actors (operators and UVs) as long as each one has a unique user and follows the handshake procedures explained in the previous sections. The design of the control system for commanding multiple UVs is out of this work's scope, but complying with the logic doesn't influence the robotic system's performance, as the handshake can be implemented in the higher-level control sequences and solely impacting the parameters of the external interfaces.

7 CONCLUSIONS AND FUTURE WORK

While measuring the overall performance of a web-based application relative to the classical standalone approach towards real-time systems is non-trivial, this work reveals the main components, workflow, and test results of a prototype solution. The results of the sea trials for the overall system confirm reaching Technology Readiness Level (TRL) 5-6. For reaching TRL 6-7, further tests in industrial and/or equipment are needed.

To make robotic services more readily available to the public in a user-friendly way, we introduced the design of a web- based service architecture to handle user interactions, manage mission life cycles, and administer communication channels between multiple operators and UVs. We also showed various use-cases in which the web layer can be deployed to provide on-demand services. With the promising results in terms of scalability, future works will try to encompass more diverse use cases that need various types of data, especially the ones that have a higher payload, such as point-cloud data.

Secondly, because the service layer is system-agnostic, multiple robotic systems with different control systems will be added. The main focus will remain on waterborne systems, but the service layer will encompass UAVs. A web-based solution over multi-agent systems developed with the extremely popular robotics framework ROS is envisioned for proving the capabilities of the service layer.

Lastly, the interaction of the web-browser clients will be enhanced by more features that will allow even more actions to be performed from the remote. We envision online data analysis and more dynamic user-operator interaction for enhancing the quality of the delivered service and the collected data.

ACKNOWLEDGEMENT

The tool was developed as part of the RoboVaaS and SCIPPER project. The RoboVaaS project received funding from the German Federal Ministry of Economic Affairs and Energy under support code 03SX463A, Universities and Research (MIUR), and ERA-NET Cofund MarTERA (contract 728053). The SCIPPER project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement Nr.814893

REFERENCES

1. Amirante, A., Castaldi, T., Miniero, L., Romano, S.P.: Janus: a general purpose WebRTC gateway. In: Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications. pp. 1–8 (2014).
2. Bao, F., Chen, J.: Visual framework for big data in d3. js. In: 2014 Ieee Workshop on Electronics, Computer and Applications. pp. 47–50 IEEE (2014). https://doi.org/10.1109/IWECA.2014.6845553.
3. Castillejo-Calle, A., Millan-Romera, J.A., Perez-Leon, H., Andrade-Pineda, J.L., Maza, I., Ollero, A.: A multi-UAS system for the inspection of photovoltaic plants based on the ROS-MAGNA framework. In: 2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS). pp. 266–270 IEEE (2019). https://doi.org/10.1109/REDUAS47371.2019.8999697.
4. Chi, X., Liu, B., Niu, Q., Wu, Q.: Web load balance and cache optimization design based nginx under high-concurrency environment. In: 2012 Third International Conference on Digital Manufacturing & Automation. pp. 1029–1032 IEEE (2012). https://doi.org/10.1109/ICDMA.2012.241.
5. Cranor, L.F.: A framework for reasoning about the human in the loop. Presented at the Usability, Psychology, and Security , San Francisco, CA, USA April 14 (2008).
6. Crick, C., Jay, G., Osentoski, S., Pitzer, B., Jenkins, O.C.: Rosbridge: ROS for Non-ROS Users. In: Christensen, H.I. and Khatib, O. (eds.) Robotics Research : The 15th International Symposium ISRR. pp. 493–504 Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-29363-9_28.
7. De Win, B., Piessens, F., Joosen, W., Verhanneman, T.: On the importance of the separation-of-concerns principle in secure software engineering. In: Workshop on the Application of Engineering Principles to System Security Design. pp. 1–10 Citeseer (2002).
8. Delea, C., Coccolo, E., Covarrubias, S.F., Campagnaro, F., Favaro, F., Francescon, R., Schneider, V., Oeffner, J., Zorzi, M.: Communication Infrastructure and Cloud Computing in Robotic Vessel as-a-Service Application. In: Global Oceans 2020: Singapore – U.S. Gulf Coast. pp. 1–7 (2020). https://doi.org/10.1109/IEEECONF38699.2020.9389285.
9. DeMarinis, N., Tellex, S., Kemerlis, V.P., Konidaris, G., Fonseca, R.: Scanning the Internet for ROS: A View of Security in Robotics Research. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 8514–8521 (2019). https://doi.org/10.1109/ICRA.2019.8794451.
10. Dieber, B., Breiling, B., Taurer, S., Kacianka, S., Rass, S., Schartner, P.: Security for the Robot Operating System. Robotics and Autonomous Systems. 98, 192–203 (2017). https://doi.org/10.1016/j.robot.2017.09.017.
11. Gratier, T., Spencer, P., Hazzard, E.: OpenLayers 3: Beginner's Guide. Packt Publishing Ltd (2015).
12. Kiribayashi, S., Yakushigawa, K., Nagatani, K.: Design and development of tether-powered multirotor micro unmanned aerial vehicle system for remote-controlled construction machine. In: Field and Service Robotics. pp. 637–648 Springer (2018). https://doi.org/10.1007/978-3-319-67361-5_41.
13. Lichiardopol, S.: A survey on teleoperation. Technische Universitat Eindhoven, DCT report. 20, 40–60 (2007).
14. Loreto, S., Romano, S.P.: Real-time communication with WebRTC: peer-to-peer in the browser. O'Reilly Media, Inc. (2014).
15. Marpe, D., Wiegand, T., Sullivan, G.J.: The H. 264/MPEG4 advanced video coding standard and its applications. IEEE communications magazine. 44, 8, 134–143 (2006). https://doi.org/10.1109/MCOM.2006.1678121.
16. Morgan, J., Liker, J.K.: The Toyota product development system: integrating people, process, and technology. CRC Press (2020).
17. Naik, N.: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: 2017 IEEE international systems engineering symposium (ISSE). pp. 1–7 IEEE (2017). https://doi.org/10.1109/SysEng.2017.8088251.
18. Oeffner, J.: A modular testbed using centralised data exchange for Autonomous Navigation Systems. Fraunhofer IML (2016).
19. O'reilly, T.: What is Web 2.0: Design patterns and business models for the next generation of software. Communications & strategies. 1, 17 (2007).
20. Radha, V., Reddy, D.H.: A survey on single sign-on techniques. Procedia Technology. 4, 134–139 (2012). https://doi.org/10.1016/j.protcy.2012.05.019.
21. Rappa, M.A.: The utility business model and the future of computing services. IBM systems journal. 43, 1, 32–42 (2004). https://doi.org/10.1147/sj.431.0032.
22. Sandhu, R.S.: Role-based access control. In: Advances in computers. pp. 237–286 Elsevier (1998).
23. Schneider, V.E., Delea, C., Oeffner, J., Sarpong, B., Burmeister, H.-C., Jahn, C.: Robotic service concepts for the port of tomorrow: Developed via a small-scale demonstration testbed. In: 2020 European Navigation Conference (ENC). pp. 1–8 IEEE (2020). https://doi.org/10.23919/ENC48637.2020.9317486.
24. Simonen, P., Dal Maso, M., Kangasniemi, O.: THE SCIPPER PROJECT: Shipping Contributions to Inland Pollution Push for the Enforcement of Regulations. (2020).
25. Taymans, W., Baker, S., Wingo, A., Bultje, R.S., Kost, S.: Gstreamer application development manual (1.2. 3). Publicado en la Web. (2013).
26. Zhang, X., Liu, Y., Zhang, Y., Guan, X., Delahaye, D., Tang, L.: Safety assessment and risk estimation for unmanned aerial vehicles operating in national airspace system. Journal of Advanced Transportation. 2018, (2018). https://doi.org/10.1155/2018/4731585.