A RabbitMQ-based framework to deal with naval sensors systems design complexity

Paul Quentel, Yvon Kermarrec, Ludovic Grivault, Pierre Le Berre, Laurent Savy

A RabbitMQ-based framework to deal with naval sensors systems design complexity

Paul Quentel[1,2,3], Yvon Kermarrec[1,2] Pierre Le Berre[3], Ludovic Grivault[3], and Laurent Savy[3]

[1] IMT Atlantique, Plouzané, France
paul.quentel@imt-atlantique.fr
[2] UMR 6285 Lab-STICC Technopole Brest-Iroise - CS 83818, 29238 Brest Cedex 3 - France
[3] Thales Defence Mission Systems, Brest and Elancourt, France

Abstract. Naval sensors systems are complex due to their nature, the services they need to fulfil under rigorous constraints and the information they need to be aware on their environment. Sensors are among the sources of the information that is collected, exchanged and synthesized and their integration into a sensors systems architecture present numerous challenges. In this context, it is difficult to visualize those systems as a whole, and we lack methods to abstract the complexity level. We will introduce an approach to evaluate different architectural concepts and their impacts on the communication network. This paper presents our methodology involving: (1) an operational simulation, which allows to get closer to real use case; (2) a benchmark involving a middleware to simulate, in an abstract manner, communications between naval platforms, the operational environment, and sensors; (3) a network monitoring tool allowing to test new mechanisms that might be implemented in the final architecture.

Keywords: communication architecture, sensors network, multi-functions sensors,naval sensors systems, RabbitMQ, Prometheus

1 Introduction

The evolution of the naval defence context requires a significant modification of sensors systems architecture in order to overcome future threats. For about ten years, the French Ministry of Defence (MoD) has initiated R&D investigations in order to enhance surface ship combat systems' operational capacities. Nowadays, naval sensors systems cooperate and each platform shares its data with the others through the Combat Management System (CMS). Management of sensors, as well as tracking, are done autonomously and locally at CMS level in each naval platform. Each surface ship keeps a local tactical situation within the CMS thanks to its own sensors and a global tactical situation thanks to exchanges with other platforms through Tactical Data Links.

In 2021, the "Veille Coopérative Navale" (VCN), a new information system, proved its effectiveness in mission. The VCN improves the tactical situation

thanks to exchanges between platforms and the local data fusion based on the radars' raw data. The Combat Cloud is a global meshed network where data are shared between users, platforms and nodes (e.g., sensors, effectors, etc.). The U.S. DoD (Department of Defense) have started to point out the concept named "Network Centric Warfare" in early 2000s [11], the forerunner of the Combat Cloud. The automation of the Combat Cloud still remains a major challenge. In this context, the cloud has to be described by a low latency, a large bandwidth and a high resilience.

In this article, we outline an approach to evaluate architectural concepts, to compare them and to choose the most efficient architecture in terms of resilience and response times and to deal with the communication network. Our team has developed a simulator of theatres of operation. The final aim of this work will be to describe an architecture for the future naval sensors systems.

1.1 General problem

In the context of naval collaborative combat, we need a new naval sensors system architecture that will allow to associate sensors from different platforms to carry out collaborative actions. The new architectures will have to enable collaboration between different networked sensors while taking advantage of data exchange for enhancing the data processing and the threat engagement. The multi-platform collaboration is a key point; it permits to propose new sensor services that will exist only thanks to the exchanges between naval platforms. The most important need is to obtain information about the enemy faster, in order to counter new threats that can undermine the fleet defence in various conditions (e.g., saturation, jamming, communication loss, material destruction, etc.). Numerous architectural issues appear in this context, such as: sensors networking, sensors management, resource optimization, improvement of data processing increasing data workability, improvement of communications bandwidth, etc.

The technical challenges are thus to improve sensors architectures to enable a lower decision time, to improve reliability of sensors systems and to scale up according to a changing number of sensors. By means of scenarios, we aim at comparing between architecture in order to show which one is the most adapted in terms of performance and adaptability against major threats.

1.2 Application and Industrial contexts

Preliminary work, mandated by French MoD, are under way on the cooperative subject, sharing radar plots between frigates is the main concern within the VCN. However, current systems have well known limitations as the quantity of data to be exchanged is growing faster than communication bandwidth. Systems need to exchange ever increasing data volumes without having the necessary bandwidth to do so.

Currently, data, which comes from sensors integrated into the platform or external platforms, is transmitted to the combat system according to a centralized architecture. A decision has to be taken at sensor level on what should

be transmitted not to overload and saturate the communication channel. These raw data could be aggregated and filtered in the architecture, or the channels bandwidth could be increased. Our systems are under hard real-time constraints (i.e., under few milliseconds), which means that the exchanged information must be processed in due time in accordance with the aforementioned constraints on bandwidth and latencies, notably to engage future major threats (e.g., hypervelocity missiles).

Furthermore, there must be an answer to system users' needs during the definition of the architecture. The architecture must be resilient, scalable, modular (i.e., the system must be able to: work in degraded mode, be extended to a larger heterogeneous fleet, include new components). In addition, one major issue of naval sensors systems is to reduce at most the decision-making time [12] known as OODA loop (Observe, Orient, Decide and Act). This problem can be partially answered by proposing a networked sensor system architecture with an increased autonomy and resilience. We have to take hypothesis on future sensors and computing technologies in order to be able to design the architecture.

The paper is organised as follows: we presented the application context and industrial needs of our research in the first section. The second section is about the related work of naval system, to show the potential of our approach. The third section presents our contributions for the design of a new architecture for naval sensors systems. The fourth section presents initial results that we have obtained. Finally, the fifth and last section summarizes this paper and open for the upcoming work.

2 Related work

Numerous teams have investigated architectures for terrestrial, aerial or naval military systems. Those research are directed towards combat cloud concepts or collaborative combat.

An American Navy approach, called CEC (Cooperative Engagement Capability) [1], aims at increasing performances of battle fleet in response to threats. Combat systems share sensor data associated with tracks, quickly and precisely, in order for the group of ships, aircraft and ground units, localized on the battleground, to act as a whole. The CEC enables to exchange radar data between the platforms, so tactical situations are shared and synchronized. In addition, the system allows a coordinated engagement of the target between the platforms. The CEC is interfaced with the platform's CMS and integrated on combat systems of rank Aegis and LHA for example (Destroyer and Landing Helicopter Dock). The concepts presented in CEC are closed to the French ones defined in section 1.

On the aeronautical collaborative combat side, the FCAS (Future Combat Air System) is a common project between French, German and Spanish aerial forces which will rise by 2040. FCAS is a system of systems containing a combat cloud that connects airborne platforms together and allows the collaborative combat.

A PhD research was conducted [8] on one multi-agent based architecture for multi-sensor systems embedded on airborne platform. The proposed architecture takes advantage of software agents which are autonomous entities capable of communicating and taking decisions in a software environment. Each agent corresponds to an object of the theatre of operation, and each agent proposes actions for sensors to perform, then a scheduler plans the actions of each sensor over time. This architecture is limited to a single platform and its own sensors. It is exactly what we expect from our future architecture, to manage sensors, but we extend the targets to multi-platform and multi-domain (i.e., air, land, sea) needs.

The CESMO project (Cooperative Electronic Support Measures Operations) [10] has been raised by NATO (North Atlantic Treaty Organization) because of the lack of standardisation and interfaces in NATO forces. The aim of CESMO is to enable cooperation between ESM sensors, while ensuring low bandwidth usage, in order to obtain better localisation of radars emitters. CESMO is mainly concerned about data sharing between allied platforms and a main platform is taking decision in a centralised way. Data are exploited locally and sensors are not controlled by this system, while for our research, we want a distributed architecture with more autonomous sensors. The optimization of the bandwidth proposed in this project is a major constraint for our research.

The European project CAMELOT (C2 Advanced Multi-domain Environment and Live Observation Technologies) [4] proposes a distributed architecture aiming to control European borderlands against illegal immigration or drugs smuggling. In these regards, a review of existing architectures has been done under criteria chosen for the CAMELOT architecture [16]. The motivations that lead the choices about the new architecture concern the capacity to command and control several UxVs (Unmanned Vehicles) as well as sensors to deliver complex services. Furthermore, standardization allows to integrate more easily new modules or services. This architecture uses a middleware, which permits different modules, services, assets or tasks to interact using a publish-subscribe paradigm. The middleware provides facilities and services such as scalability, modularity, and distribution, to mention a few. A study about the different middlewares according to architecture needs has been made in this report, the project CAMELOT chose to integrate the middleware RabbitMQ.

Some authors [13] pointed out that the current implementation of RabbitMQ defines and configures the queues and the consumers when the application is launched, as a consequence the system does not scale dynamically when more messages are incoming while queues are already full. The paper [13] presents a software tool that monitors messages between cloud components in the context of smart home system, this tool enables auto-scaling when problems are detected. They came to use RabbitMQ for several reasons like multiple messaging protocols, message queuing and delivery acknowledgement, in the proposed architecture, the micro-services send data at publisher/producer side, the consumer nodes consume messages stored in a queue. Furthermore, the paper provides a test environment using Zabbix as monitoring tool, the authors introduced dif-

ferent values for parameters like prefetch or number of consumers. Our aim is different, we want to use RabbitMQ and monitoring tools to simulate the network load while developing new concepts in the context of Naval Combat and the limited access to the bandwidth.

As we can see in this state-of-the-art, most of the works in the domain of collaborative combat are focused on sharing data between platforms in order to have a better view of the battlefield. As far as we know, testing the network of architectural concepts in the military domain has never been done. The proposed framework is a premise of the incoming work that will propose to exchange data between software agents in a distributed architecture, it will be presented in a future paper.

3 Contribution

Our work is composed of three sub-works. The first step was to use a proprietary software called Sonia and to interface it with another software, the STK (Sensor Tasking). The second step was to use and experiment the middleware RabbitMQ [3] and a tool used to test performances [2]. The final step was to develop a benchmark using RabbitMQ. The purpose of this work was to abstract the industrial framework complexity and, by the end, to possibly integrate the middleware RabbitMQ to this framework.

3.1 Framework and Methodology

The software Sonia provides a simulated environment that represents the theatre of operations. In this environment, actors like naval or aerial platforms can be added and the systems can evolve and adapt to the operational context. Each platform can enable or disable their sensors and effectors; it is also possible to make the platform moving in the space and so to create scenarios to replay afterwards. The benefit of this simulator for our work is to retrieve sensor data when a target is being detected and tracked.

The software STK (for Sensor TasKing) is built on an agent-based architecture [17]; it relies on Ludovic Grivault's Doctoral Thesis [8] by extending the concepts to distributed architecture instead of a centralised one. The existing algorithms from the software propose to activate sensors on very short intervals, and then sensors resources are booked. An agent represents an object (e.g., a projectile, an aircraft, a warship, etc.) in the theatre. The internal memory of one agent gathers various information about one object such as its position or its speed. Moreover, all agents take decisions about the tasks that the sensors will perform, thus, there is kind of a loop because the sensors give feedbacks to agents about the object they track and the tactical situation is completed or updated.

Those software applications need to be interfaced, as the platforms from the simulator Sonia could use the intelligence from the agents in order to schedule sensors' tasks. The middleware ZMQ (Zero Message Queue) [9] is used to

exchange messages between those two entities; a platform in the simulator can send sensor tracks towards the STK, after that, agents are created or updated for each track received. The agents can send to the platforms the scheduling of sensors to accomplish the mission.

Currently, the framework is based on a centralized architecture, only one platform contains agents and performs computation to establish the sensor tasking of the entire fleet. Nevertheless, the centralized architecture has few specific limitations for the requirement of the naval domain (e.g., platforms are distant, the network must be resilient to communication loss, etc.) and we face problems of complexity that restrain us when proposing new concepts of architectures.

That is the reason why we have decided to work on a benchmark using RabbitMQ [7] in order to give a level of abstraction to our concepts from a network point of view. The benchmark will evolve from a simple program to a software with new features. The shared data between platforms will be used in the framework to approach the reality in our model, with more realistic message exchange. Later, users will be able to modify parameters and different scenarios will be proposed to show the impacts of the suggested architecture concepts.

3.2 RabbitMQ

RabbitMQ [7] is a message-oriented middleware (MOM), which is used within distributed architectures in order to communicate and cooperate between services. This MOM uses the AMQP standard (Advanced Message Queuing Protocol), the messages are sent asynchronously from a producer to a consumer through queues; messages are sent to receivers thanks to binding and routing keys.

Thanks to its features, RabbitMQ brings important properties for the architecture, such as:

- Modularity: services or other heterogeneous networked entities can be interconnected with the addition of a serialisation tool like JSON;
- Scalability: nodes can be added or removed dynamically and the middleware is designed to deal with thousands of nodes;
- Quality of Service (QoS): queue size can be modified, the Time-To-Live (TTL) of a message can be changed as well as the priority, the messages can be acknowledged;
- Interface and management: there is a management user interface or a metric exporter (e.g., metrics can be messages per second, the number of producers or consumers, etc.) that can be enabled through plug-ins;
- Fault tolerance and failures since nodes or messages will be lost;
- Secured communications.

Furthermore, RabbitMQ proposes some API (Application Programming Interface) in different languages such as C#, Java, PHP or python.

RabbitMQ is not the only existing MOM. The paper [14] compares four MOM for the communication in distributed architectures : AMQP (RabbitMQ),

Kafka, MQTT and ZeroMQ. Multiple features are compared like QoS, security, the standardization of the MOM or the transport protocols used. They consider RabbitMQ as a very balanced MOM with the biggest flexibility and a lot of functionalities, ZeroMQ outstands from all other MOM by its performance but it is harder to be implemented. Furthermore, RabbitMQ provides mechanisms of persistence and message retention. There are plenty of documentation, and and monitoring can be be achieved by integrating Grafana and Prometheus in a RabbitMQ-based architecture.

3.3 A load testing tool : PerfTest

PerfTest [2] is a tool developed in Java for testing the throughput performances of RabbitMQ by generating load. It is possible to play with several parameters such as the number of consumers, the number of producers, the message sending and receiving rates, the queue size or the size (in bytes) of a message. It is also possible to generate random load, as well as to modify message publishing rate on defined time intervals.

The tool allows users to activate a metric exporter that enables exportation of Prometheus metrics, like with the RabbitMQ's plugin. The metrics exported are data related to latencies, non-routed messages, the total of published, rejected or consumed messages, but also metrics about memory or CPU usage.

The tests that we conducted on this tool brought us the following analyses: PerfTest was created for making tests about network load, which is not exactly appropriate for our needs and it has a lack of flexibility. The tool proposes to generate a global latency as well as a global throughput, moreover each message has the same size. Therefore, PerfTest is limited in regards to our objectives. We would like to be able to modify different parameters like the size of messages, latencies between different producers and consumers. In plus, we would like to define which producer is sending messages to which consumer and not to only one consumer. Finally, we would like to be able to do the following: simulate communications or packet losses, delays; add or remove nodes during simulation; modify latencies or throughputs on communication links. Given the reasons listed above, we decided to develop our own RabbitMQ-based framework which fits more our needs.

3.4 RabbitMQ-based framework

The developed framework allows starting many consumers or producers, their number are defined in a configuration file. We chose to use topics as routing rules in the Exchange, which is initially receiving messages before routing them. The producers define routing keys while consumers define binding keys.

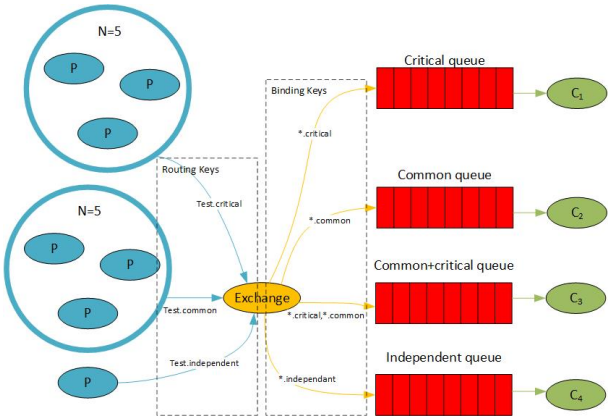

Fig. 1. Example of configuration with 11 Producers, 4 Consumers and 3 different topics

Figure 1 shows a configuration example with different routing and binding keys. For instance, there are five producers that publish messages via the routing key "Test.critical", these messages are sent and duplicated into corresponding queues (i.e., queues with the binding key "*.critical"). Consequently, consumers one and three will receive the same messages from the aforementioned producers.

We started the development of the framework with a very simple program, which sent a message from a single producer to a single consumer. We added features afterwards, the purpose being to get closer to an industrial model. As far as we are concerned, the second step was to be able to send several messages from N producers to M consumers. The third step was to choose the size of the messages and the frequency at which they are sent in order to modify the throughput.

Prometheus and Grafana are the two software applications being used in this framework. Together, they present graphically network data inside dashboards.

Prometheus Prometheus [15] is a network monitoring and alerting software. It registers metrics in a real-time database and provides a query language called PromQL that queries data in the database. Four types of metrics can be used: counters, gauge, histogram and summary. With Prometheus, it is possible to: use regex (Regular Expression) in order to modify outgoing metrics, to collect values during time in a vector and to use aggregation operator (e.g., sum, min, max, etc.) as well as other functions.

Grafana Grafana [5] is an analytical and monitoring software which allows to show graph within a dashboard and whose data is coming from temporal database (in our case, the data is stored in Prometheus server). Grafana allows to scrape data from this database which is updated every few seconds. In addition,

Grafana permits to create personal dashboard to fit user needs and to show time graph, gauge or histogram to highlight the data.

Variables can be used in Grafana, so the same dashboard can be used for many producers or consumers; combined with regex, the dashboard can be auto-configurable that fits well even when the configuration is modified. Thus, the scalability of the framework is guaranteed as modification in RabbitMQ configuration will not impact changes in the dashboards.

4 Results

We worked with RabbitMQ, Prometheus and Grafana. Firstly, we define a configuration that describe consumers, producers and exchanges between them. Then, we export real-time information, called metrics, about consumers and producers to a Prometheus server. Finally, Grafana queries and displays data from this server.

At this stage of the study, our first results are limited to display the metrics according to the defined configuration without analysing nor comparing the results. These results will allow us to compare architectures according to several criteria. For instance, we can take the simple case of a centralized architecture where there would be the leading naval platform represented by one single consumer, and the allied fleet's sensors represented by a variable number of producers. Afterwards, we can compare this architecture with another one where the number of consumers is more important for instance.

As depicted in figure 2, general information coming from RabbitMQ servers are displayed; the configuration used there is following the one presented in figure 1. We can observe the average throughput of all producers (i.e., incoming average), as well as for all consumers (i.e, outgoing average). The consumers' throughput is more important because messages are duplicated and sent to several queues.

Fig. 2. Dashboard with general information from RabbitMQ

In figure 3, a time graph provides throughput information about one consumer. In this example, the information is updated each five seconds and the

instantaneous throughput is shown in green and the average throughput in yellow.

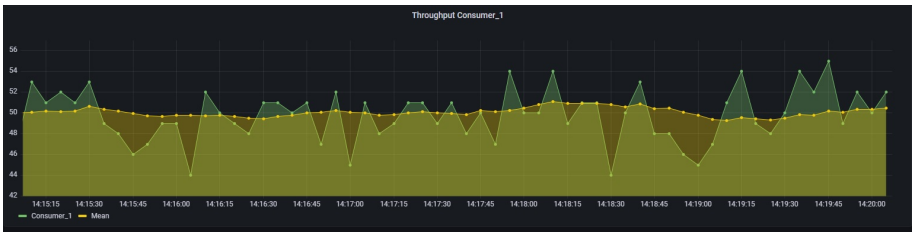

Fig. 3. Graph with the throughput of one consumer

More results were displayed such as the number of messages effectively sent and well-received for each producer. That means we can observe that the producer number one sent X messages to the consumer number one and Y messages to the consumer number two, by making an addition we can know the number of messages sent by the producer. The throughputs of each consumer and producer are displayed in a table. Then, two graphs were displayed thanks to plugins from Grafana, the graphs show producers and consumers as nodes (or vertices), the links between the nodes (or edges) are made using the number of messages received by a consumer from a producer.

5 Conclusion

Our studies aim to abstract naval sensors systems architecture complexity by proposing a method for evaluating architectures using a RabbitMQ-based framework. In these perspectives, we introduced the context of our research and the upcoming issues, then we highlighted the work done in the field. The first contributions were presented:

- The interfacing of two proprietary frameworks;
- First tests of performances using PerfTest tool from RabbitMQ;
- The development of a RabbitMQ-based framework allowing to use many possible configurations between producers and consumers;
- First results by displaying metrics with Prometheus and Grafana, and ability to compare architectures under various criteria.

For the future work, we will integrate our consumers and producers in a multi-agent system (MAS) [15] extended from the concepts presented in 3.1, this is one of the architectures we would like to propose. Our work is placed in a context where many naval platforms have to work cooperatively, so data exchange is a major issue. Different organizations are possible for the agents [6], and some rules (e.g. message sending frequency, optimized packet length,

etc.) have to be followed to optimise the bandwidth use. Depending on how your agents are situated, and how they are communicating, the complexity is increasing while the size of the fleet is growing and we need to point out the limits. Then, RabbitMQ will be used to exchange data between agents which are physically situated on distinct platforms, the agents need to cooperate so they produce and consume messages from one to another. RabbitMQ, combined with Grafana and Prometheus monitoring tools, will show graphical outputs that we will be able to analyse and we will highlight the drawbacks in order to evolve our concepts of agent.

References

1. The cooperative engagement capability. *Johns Hopkins APL Technical Digest, Volume 16, Number 4*, 1995.
2. RabbitMQ perftest. *https://rabbitmq.github.io/rabbitmq-perf-test/stable/htmlsingle/*, 2022.
3. RabbitMQ tutorials. *https://www.rabbitmq.com/getstarted.html*, 2022.
4. TEKEVER ASDS. Specifications. *https://www.camelot-project.eu/results*, 2018.
5. Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications*, pages 187–240. Springer, 2021.
6. Ali Dorri, Salil S Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *Ieee Access*, 6:28573–28593, 2018.
7. David Dossot. *RabbitMQ essentials*. Packt Publishing Ltd, 2014.
8. Ludovic Grivault. *Architecture multi-agent pour la conception et l'ordonnancement de systèmes multi-senseur embarqués sur plateformes aéroportées*. PhD thesis, Sorbonne université, 2018.
9. Pieter Hintjens. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.
10. Frank T Johnsen, Trude Hafsøe, Espen Skjervold, Kjell Rose, Ketil Lund, and Nils A Nordbotten. Multinett II: SOA and XML security experiments with cooperative ESM operations (CESMO). *FFI rapport*, 2344:2009, 2008.
11. Bonnie Worth Johnson and John M Green. Naval network-centric sensor resource management. 2002.
12. Bordellès Jérôme and Ulvoa Mickaël. The armed forces' current and future needs of radio frequencies: A strategic issue for france. 2020.
13. Milica Matić, Sandra Ivanović, Marija Antić, and Istvan Papp. Health monitoring and auto-scaling rabbitMQ queues within the smart home system. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 380–384. IEEE, 2019.
14. P Sommer, Florian Schellroth, M Fischer, and Jan Schlechtendahl. Message-oriented middleware for industrial production systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, 2018.
15. James Turnbull. *Monitoring with Prometheus*. Turnbull Press, 2018.
16. UPVLC. Architecture and data model CAMELOT. *https://www.camelot-project.eu/results*, 2018.
17. Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.