SIMPLIFIED DEVELOPMENT FOR MULTIPLE ROBOTIC PROJETS USING JAUS

Simon Bolduc[*] and François Campeau[†]

Every unmanned system needs to be remotely monitored and controlled to efficiently operate. Developing a system specific solution that will answer all needs and be flexible enough to accommodate modification to a component on the unmanned system requires a lot of time and efforts. This problem can be solved by using a joint architecture satisfying any unmanned systems. The SONIA Autonomous Underwater Vehicle (AUV) team from École de technologie supérieure (ÉTS) in Montréal, Canada has developed a framework solution using the Joint Architecture for Unmanned Systems (JAUS) SAE standards. This paper presents the framework composed of a communication library and a telemetry visualization system. It also explains how it was successfully integrated into two distinct unmanned systems developed here at ÉTS, with a detailed explanation of the integration in SONIA AUV project. The library describes the architecture and communication between systems and telemetry visualization system. The telemetry enables users to monitor and control any type of unmanned system using the JAUS standards.

INTRODUCTION

Even if the goal of an autonomous vehicle is to accomplish tasks without any input from an operator, telemetry is often required to be able to easily monitor the behavior of the robot. This monitoring of the vehicle greatly facilitates the debugging and optimization of the platform. Moreover, the telemetry visualization system frequently allows to more easily control and operate the vehicle. Consequently, this is a common need for multiple robotic projects that is normally specific for each project.

For the SONIA AUV project, from École de technologie supérieure, many telemetry visualization systems have been built and this tool is now crucial for the operators of the autonomous submarine. They can adapt the mission to the obstacle course, visualize the sensors and mission log, set waypoints and control the actuators. The original communication protocol was created by the team so the reusability of the software was very poor. Also, the last version was directly coupled with the control system. When the control system was changed, everything needed to be rewritten on the telemetry visualization system side. Therefore, the team decided to use a standard communication protocol and completely decouple the communication from the control system.

At École de technologie supérieure, three autonomous vehicle projects exist, a ground vehicle, a quadrotor and a submarine. All those teams have common needs and one of them is a telemetry visualization system to operate their vehicle. The teams decided to collaborate and create OctETS (Open Collaboration Tools École de Technologie Supérieure), an environment where it is easy to share code and ideas. This environment is based on an open-source model and it includes code

[*] Software team leader, SONIA AUV, 1100 Notre-Dame West, Montreal, Quebec, Canada

[†] Mission manager, SONIA AUV, 1100 Notre-Dame West, Montreal, Quebec, Canada

repositories, a bug tracker and a build server. A major component of this collaboration environment was the creation of a common communication library and a common telemetry visualization system which can be used by every team to operate their vehicle.

Even though each team is controlling different kinds of robots, it was decided that the adoption of a generic library and telemetry visualization system would help in the development of the multiple projects. Sharing the development effort and the debugging of the applications helped us create a stronger platform. It also encouraged the creation of reusable software. It could be used for multiple years with minimum development efforts to add new functionality. The generic nature of the project also meant that it needed to be extensible to support the specific needs of each type of robot without sacrificing flexibility.

It was decided that Joint Architecture for Unmanned System (JAUS), would be the foundation for the common communication framework. A generic library was developed containing all the required support for JAUS. Moreover, JAUS has been chosen to permit the interoperability between the unmanned systems. JAUS also has been a requirement of various competition the teams have participated in.

This paper presents an overview of the JAUS integration and the JAUS Telemetry. Towards the end is demonstrated and explained SONIA's specific integration.

RELATED WORK

Multiple open-source and proprietary implementations of the JAUS standards already exist. Among others, there is OpenJAUS, and JAUS++. Both of them are coded in C++. It was decided that the teams would create a new JAUS implementation in Java because it is a well known programming language at ETS and all the teams' control systems are written in this language. It would have been possible to use a Java Native Interface (JNI) to integrate one of these existing non Java libraries to the different systems. This idea did not please the teams for maintainability reasons. There is also the JAUS Tool Set that can be used to design services and then generate the source code. The code generated did not follow any of the style standards the teams had in place. So the teams chose to write the custom services by themselves.

SYSTEM OVERVIEW

As stated before, the system contains a JAUS Library that is an implementation of the standard that allows sending and reception of JAUS messages and a JAUS telemetry visualization system to monitor and control the unmanned system. The library is used by every subsystem that needs to communicate with other JAUS based subsystem such as the telemetry visualization system or an unmanned system.

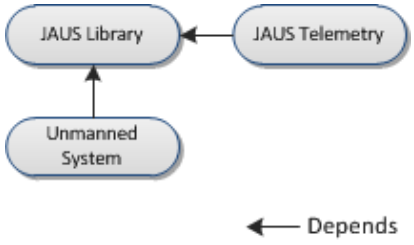

Figure 1. Dependencies overview.

When the collaborative development of the system started, three major quality attributes were defined to build the system: maintainability, extensibility and reusability. The maintainability requirement is primary to all robotic clubs since there is a lot of staff turnover and due to the lack of time; the members generally produce poor documentation. Consequently, the code needed to be easy to understand and easy to modify. Then, the extensibility of the project is very important. Every team needs to be able to adapt the code and extend it to better suit their needs. The adaptation to a different project needs to be simple and easy or the system will not be used at all. Finally, it is also very important for the system to be reusable; so that any unmanned system could use it without being limited by the framework.

JAUS LIBRARY

The first aspect that was tackled in the creation of the JAUS Library was how the subsystems, nodes and components would be represented. In some frameworks everything is a component with a specialization. Even though subsystems, nodes and components all contain services and handle messages, it was decided to create them as independent classes due to their different role in the system. Another aspect was how these elements would be contained in the runtime environment. The JAUS architecture is very flexible on what consists of a separate process and what should run as a thread in a given application. It was decided that subsystems would run in their own process, every node would have their own process and components would run in separate threads under the node's process. This allows the subsystem to be the single communication entry point of the vehicle and have many nodes working together towards the goal of controlling the vehicle. Any given unmanned system has a communication entry point, normally a computer, and then has one or many nodes that could be spread around other computers or other types of electronic devices. In the case of SONIA, the subsystem and the main node, AUV6, both run on the main computer. The idea behind this separation of processes was to give us the possibility to transfer services on custom electronics board to take advantage of real time computation, which cannot be achieved on a regular computer.

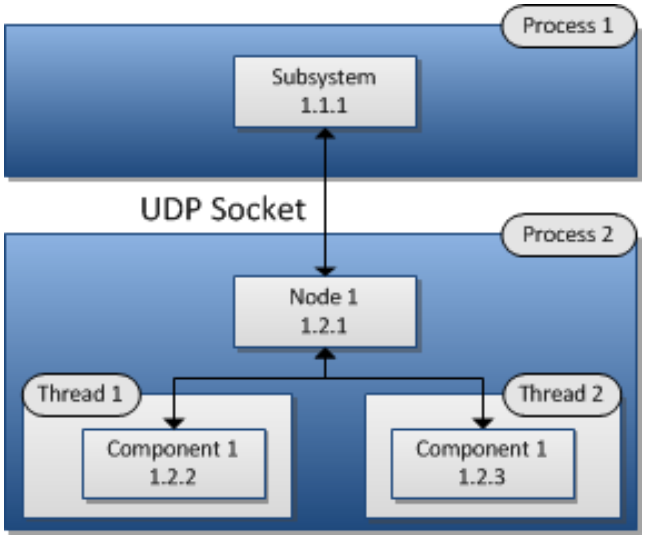

Figure 2. Process architecture.

The fact that subsystems run in different processes than nodes caused us some real headaches. Subsystems were not able to dynamically know about the different nodes that reports to it. Moreover, communication between subsystems and nodes became more complicated. In order to solve these issues nodes open a UDP Socket. To make sure the subsystem would still be able to communicate on the default JAUS port, each node opens its socket on a random port, preventing any port conflicts. This solution did not solve the dynamic discovery issue since the subsystem is not aware of the destination port of its nodes. In order to answer this problem, every node knows the IP address of their parent subsystem and sends out a Report Heartbeat message on the default JAUS port(1, figure 3). Once this message is received by the subsystem, the node is added to the subsystem routing table (2, figure 3). The subsystem is then able to periodically poll the nodes to make sure that they are still up and running (3, figure 3).

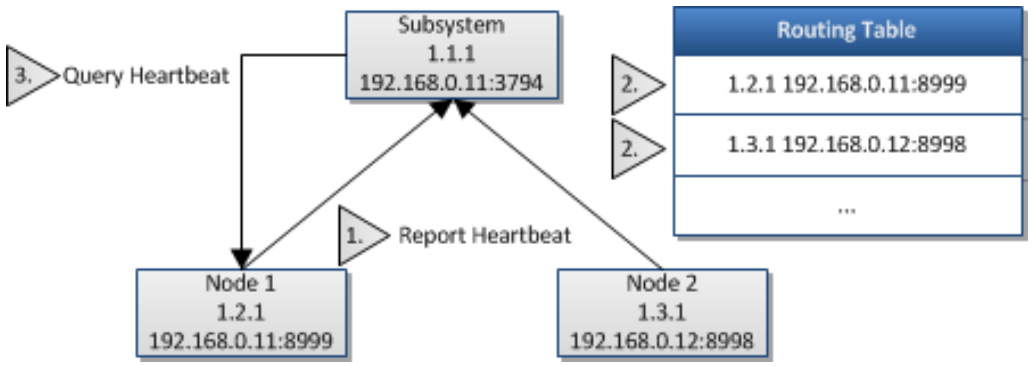

Figure 3. Node subscription process.

Every JAUS element offers services. These services are represented by classes each implementing their own state machine. Contrary to every other service, the transport service was coded differently. Since it is the base of every message handler, it was directly integrated in the components, nodes and subsystems. The message routing mechanism was split into two separate sections. At first, the message enters the element directly. If it is destined for itself it is directed to the service manager. If the message is for another element, it is routed to the right one either through a socket, or directly to the component in the case of a node. That way, it is easier to manage the flow and the order in which the messages had to go through the services. The transport service is effectively treating a message last when sending but has to be the first to treat a message when receiving. As for the other services, the message trickles down from the top most service then through the list of dependencies. To achieve this, a chain of command was implemented in the service manager. Every message goes through the chain of command until a service handles the message. To ensure consistency, every time a service is added to the service manager all its dependencies are added after it in the chain of command. This way the user of the JAUS framework does not need to know about the dependencies and eases the learning curve.

The framework offers a vast array of delegates to allow an easy integration in every control systems projects. Every service offers one or more delegates to allow decoupling of the framework with the actual control system. The unmanned system implements the delegates and registers them to the framework allowing every aspect of the control system to be monitored and controlled by the telemetry visualization system or any other JAUS system. Adding this delegation made sure that every team could use different architecture and simply have to bind to

the JAUS library framework. This was a key feature for widespread acceptance of the framework by all the teams composing OctETS.

JAUS TELEMETRY VISUALIZATION SYSTEM

The JAUS telemetry visualization system provides a generic framework for the teams to easily develop widgets to monitor and control their vehicle. As stated before, the telemetry visualization system is using the library to communicate with any number of unmanned systems.

The telemetry visualization system is composed of a node and a component that establish communication with the unmanned systems. In order to facilitate debugging of the control system, the telemetry visualization system was implemented as a node. Doing so allows the control system and the telemetry visualization system to run on the same computer and avoid port conflicts. Moreover, when the user open a new widget, a new component is created for this widget, called a widget communicator, with a custom widget service. The widget service receives messages from the component and then, passes it to a list of message handlers. Normally, each widget has a message handler whose responsibility is to retrieve the information from the message and set it to the graphical components of the widget. There is also a generic message handler for all widgets to handle the core messages such as event control.

Every widget has its own specific widget communicator that manages the subscriptions of the widget. A subscription holds the address, the query message and the event type. Since a widget is directly bound to a JAUS component, it can easily send messages through the normal JAUS message flow. This allows widgets to effectively control behaviors of the unmanned system.

Some widgets are available in the common telemetry visualization system but these are very generic. They are mostly used to debug the telemetry visualization system. There is a console widget displaying telemetric debug information. A widget that shows a tree of all the components of the system. Another widget to forge JAUS message to send to any component, mainly used for debugging purposes. A generic alarm widget displaying alerts to the operator. Another widget displaying every value handled by JAUS. Each value is displayed in a table. This table is very useful when someone wants to monitor a specific value and no widget displays it.

To configure the telemetry visualization system to their own needs users choose which widgets they want to use and where they want them positioned on the interface. To save time, the operator can save their perspective, a set of widgets, and then the application will load it automatically at program start up.

A major requirement for the JAUS telemetry visualization system is flexibility of the system. The software application must be usable for any type of unmanned system that supports JAUS. Also, the common project had to be as generic as possible because submarine specific widgets should not mix with ground vehicle widgets due to maintainability issues. All the platform specific code is managed by each team. When a JAUS telemetry visualization system is instantiated, it is possible to specify every widget to load by using Java's reflection features. This way, the generic telemetry visualization system is not coupled with any specific code.

To ease the development and debugging of the telemetry visualization system a mock JAUS system was developed, this system also uses the JAUS library. This software is used to simulate values and confirm that the telemetry visualization system displays these values properly.

Developing the telemetry visualization system with the real control system would have been very complicated. First, the real control system is always under development. It is also bound directly to the hardware. The mock system helped development and made integration with the real control system seamless.

SONIA INTEGRATION

For the 2011 competition, the SONIA team decided to code a new control system. The team needed to rewrite all the tools used to control the submarine because they were all directly coupled with the control system. As soon as the JAUS library was functional, the integration with the SONIA control system started.

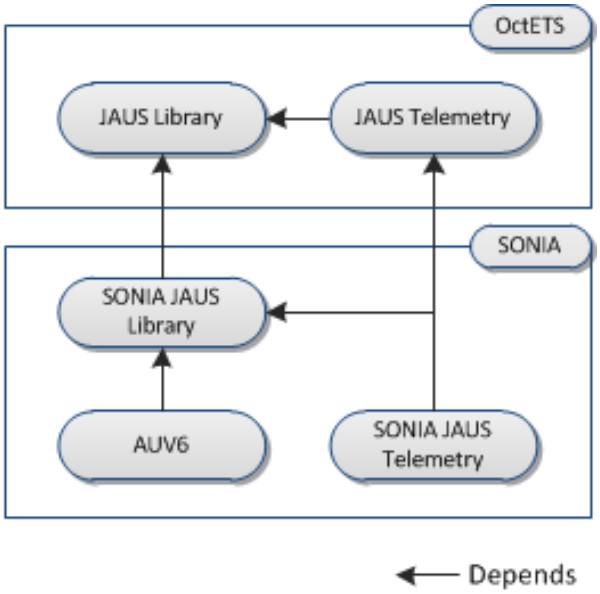

Figure 4. SONIA dependencies overview.

SONIA JAUS Library

Every SONIA specific service and message implementation is gathered in the SONIA specific JAUS library. This extends the common library to add new features, specific to SONIA. Operators want to monitor all the sensors' data. All this information needs to be sent to the telemetry visualization system. They also want to control all the actuators. To do so, every sensor and actuator have their own service that handles its own data. For example, there is a service for the doppler velocity log, the inertial measurement unit, the power management board, the torpedo launcher and several others.

For the SONIA project it was necessary to override some features from the JAUS library such as the control management. For the needs of the team, management service added too much complexity. Every widget needed to ask for control before setting a value. It was mainly problematic when two widgets could modify the same value but only one has the right to do so. The control management was disabled and was not used in the current implementation.

Also, the Local Waypoint Driver service was modified in the SONIA implementation. First of all, a more reusable waypoint tolerance with different values for x, y, z and heading. Moreover, the team disabled the travel speed record because it is not used.

SONIA JAUS Telemetry

The SONIA JAUS telemetry project gathers all the SONIA widgets and injects them in the common telemetry visualization system. Example widgets include a mission editor, an attitude and depth indicator, a control for all the submarine's actuators, output of the mission logs, waypoint control and indicators for many of the submarine's important values. There is also a joystick widget to easily control the submarine in pool tests or demonstrations.

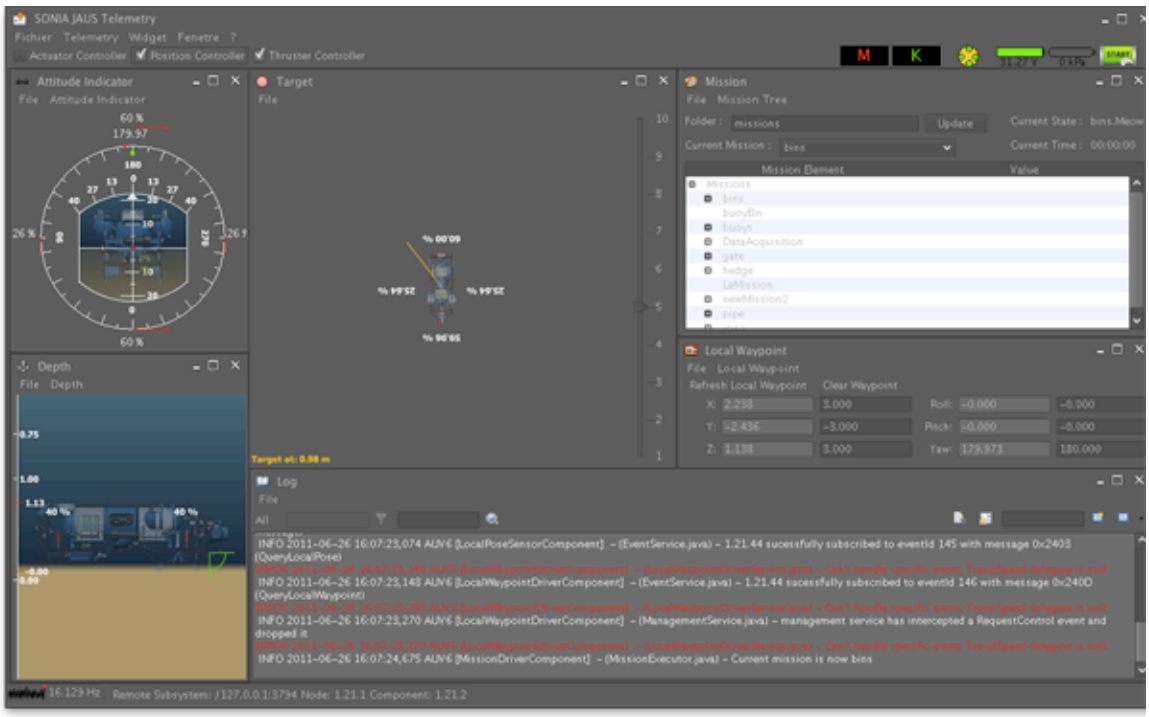

Figure 5. SONIA JAUS Telemetry.

Of course our implementation of the telemetry visualization system still requires some work. For example, when two operators are using the telemetry visualization system to test the submarine some problem can occur. First of all, several telemetries can be used at the same time to control the same control system. When two operators connect to the control system with the same JAUS address, all the event registrations break. Another problem is when a telemetry visualization system is terminated without unregistering their events; the server will continue sending messages. To fix the latter, the liveness service could be used as a keep alive mechanism for the communications.

AUV6 Interface

A full integration of the JAUS architecture in our control system meant that internal communication would have to be done by messages. Past experience showed that this type of communication is hard to debug since it is not always possible to track down the origin of

messages. Team SONIA decided to only use the JAUS framework to handle communication between the telemetry visualization system and the control system. The control system was designed around a linear control loop paradigm. Every part of the system is linked to a JAUS service via one or more delegates. Allowing monitoring and control of the entire system with minimal modification to the design of the older control system. This way, the control system had complete access to all the information without having to go through the mechanism of requesting control from JAUS. Since SONIA is an autonomous vehicle, it needs to be able to control itself without any external communication. In other words, the control system constantly needs to have full rights over all the sensors and actuators available.

RESULT

Overall, the project was a success. Every ETS robotics team was able to use the same baseline for their vehicle telemetry visualization system and also to support interoperability between their unmanned systems. Consequently, the reusability requirement was achieved. The chosen architecture worked well as it presented no conflict between the teams and the code base was not too complicated to manage for the developers. Moreover, with the service delegates, it is easy to integrate the library to an existing control system. It only needs to instantiate the JAUS architecture and provide the data. With this implementation, it is also easy for a team to add new components to the system.

As for the telemetry visualization system, the widget modularity gives great results as the teams can easily exchange widgets. Also, the modularity simplified the learning curve for new widgets developers as they do not need to know the entire system to add functionality to the software. Finally, the usability of the system is good; the operators were able to efficiently maneuver the different types of vehicle. It has been used to control and monitor a submarine since the end of January without major problems and the team should be able to keep this solution for the years to come.

FUTURE WORK

Every time the mission switch is set to on, the SONIA control system saves the camera streams, sensor values and mission logs. This data is then synchronized and replayed using a tool called the log replay. This is used to debug the tasks accomplished by the submarine after a mission. By using JAUS as a communication abstraction it would enable the telemetry visualization system to be used as a graphical user interface for the log replay. Some widgets and services would be added to be able to control the logs and to stream the cameras. This log replay telemetry would connect to a mock AUV control system, reading its data from files instead of directly from sensors. To simplify the implementation, JPEG compression would be used on every image to be able to send a complete image per UDP packet.

During the integration of JAUS with the SONIA control system, a lot of boilerplate code has been written in order to send the vehicle information to the telemetry visualization system. This code includes the creation of new custom services, messages and records. This took some valuable time from team members to write and test this code. To reduce the cost of creating new services, a code generation tool could be implemented to automate this task. This tool could have a graphical user interface to make it usable by anyone on the team.

CONCLUSION

In conclusion, this paper demonstrated that the integration of the JAUS architecture in multiple robotics projects with very different purposes is possible. The choice of going with an established protocol saved the teams a lot of development time and allowed the framework to be available for everyone to use and integrated before their various competitions. The creation of the framework helped to integrate the protocol in every team's project, by offering an interface easy to use. The different delegates rendered the integration to various architectures very easy. This allowed every team to work on different architectures depending on personal preferences and competition requirements. Also, the telemetry visualization system offers a very extensible interface to add different widgets depending on the different projects. Finally, the most important reason OctETS was founded upon, sharing of a common library, was a success. Every autonomous robotics teams at ETS are now using JAUS and the JAUS telemetry visualization system with their own custom widgets. Furthermore, if new robotics teams were to emerge at ETS, they would have a solid and tested base software suite to build upon. This would allow them to concentrate on their own control systems and not have to worry about monitoring the robot itself.

ACKNOWLEDGEMENTS

First of all, we would like to recognize the generous support of the École de Technologie Supérieure and all our sponsors, without them we could not achieve so much and push the limits of our submarine. Also, we would like to thank Kevin Larose, Olivier Allaire, Pier-Luc Caron St-Pierre, Marc-André Courtois, Eric Arseneau and the SONIA AUV team that helped us with our projects and the writing of this paper. Finally, thanks to Frédéric Morin, Guillaume Dorion Racine, Jérome Gagnon and Michael Mimault from Dronolab for their contribution to the project.

REFERENCES

- JAUS Tool Set http://www.jaustoolset.org/
- Open JAUS http://www.openjaus.com/
- JAUS++ http://active-ist.sourceforge.net/
- AS5684 JAUS Service Interface Definition Language http://standards.sae.org/as5684a
- AS5669 JAUS / SDP Transport Specification http://standards.sae.org/as5669a
- AS5710 JAUS Core Service Set http://standards.sae.org/as5710
- AS6009 JAUS Mobility Service Set http://standards.sae.org/as6009