Journal of Engineering and Applied Sciences 13 (Special Issue 12): 9336-9340, 2018
ISSN: 1816-949X

Autonomous Marine Vehicles for Open-Source Middleware Nested Communication

R.N. Raju
GMDSS, AMET University, Chennai, India

Abstract: Software systems for robotics increasingly require support for robust interprocess communication with standard interfaces which has given rise to the use of "middleware" software projects. However, Autonomous Underwater Vehicles (AUVs) have a significantly different inter vehicle communication regime than other branches of robotics due to the physical realities of the ocean as a communication medium. Goby3 is a new middleware, the first specifically designed to address inter vehicle inter process and inter thread communication on AUVs in a unified manner. Goby3 is based on C++11 and is minimally restrictive on the types that can be published and subscribed using it. A reference implementation is given that uses C++ shared pointers for interthread, ZeroMQ for interprocess and Goby-Acomms for inter vehicle communication. This application is shown to provide similar or better performance to existing middlewares.

Key words: ZeroMQ, application, middlewares, communication, Goby3, AUVs

INTRODUCTION

Developing and maintaining software systems for Autonomous Underwater Vehicles (AUVs) and autonomous surface craft is rapidly becoming one of the most complex tasks for successful development of these platforms as hardware components (sensors, actuators, computing elements) reach a plateau of maturity and commoditization. Managing this complexity can be accomplished through several means: abstraction and standardization of interfaces, modularization of software components and leveraging of existing open source resources. To assist in these goals, various software projects have become widely used in the marine robotics community. These projects are referred to as "middlewares" for their intermediary role in between the operating system resources (especially, those about communication) and the robotic application software. Several examples of middleware that have been used on marine vehicles include MOOS (Benjamin *et al*., 2009), the Robot Operating System (ROS)(Quigley *et al*., 2009) and the Lightweight Communications and Marshalling (LCM) project (Huang *et al*., 2010).

From the prospective of middleware, the marine environment poses a significant unique challenge: the extremely low throughput typically available for inter-vehicle communications, since, acoustic modems and low throughput electromagnetic-based systems, (e.g., satellite modems) are often the only practical choice. None of the existing middleware's addresses this specific challenge and approaches to intervehicle communication tend to be decoupled from intervehicle communication. At the same time, users are increasingly fielding multiple AUVs due to reduced vehicle cost and increased need for individual coverage.

Thus, Version 3 of the Goby underwater autonomy project (Goby3) offers a new middleware specifically designed for allowing nested autonomy (Benjamin *et al*., 2010) where de-decisions are made as close to the data source as possible to avoid excessive data traffic. However, as needed, messages can be requested to a scope further from the source. To increase its general applicability, the design of Goby3 can support any choice of transport mechanisms used, (e.g., TCP/IP for interposes) or data marshaling schemes, (e.g., google protocol buffers, DCCL, JSON, msg pack). However, a reference implementation that makes use of various high-quality open source libraries is presented for immediate use by the community.

This study also, reviews, a study of optical, surface morphological and electrical properties of manganese oxide nanoparticles (Roy *et al*., 2016). A comparative study of saline and non-saline water in the application of tomato yield by using photonic sensor.

MATERIALS AND METHODS

Existing middleware used in the marine community: To the degree any middleware is run on an aquatic robot, the most common choices are ROS or MOOS and to a lesser extent LCM. All of this middleware provide an interprocess communication mechanism and a suggested or required marshaling scheme for converting native system types, (e.g., C++ classes) into

bytes and vice-versa at the receiver. ROS also, provides an interthread communication mechanism ("mode lets").

The ROS and MOOS transport mechanisms are built on the Transmission Control Protocol (TCP) and thus, provide reliability at the layer of carriage for published packets. LCM uses the User Datagram Protocol (UDP) multicast functionality and thus, provides no security guarantees (but increased performance for high through- put applications).

ROS and LCM use a conceptually similar Interface Description Language (IDL) that allows the user to define data structures in a language neutral format from a collection of primitive types (integers of various sizes, floating point values, strings, etc.). These message def-initials are then compiled by a tool provided by the middleware into one or more language-specific pre-sensations, (e.g., C++ class, Python class) that can be used in the user's code. The standalone protocol buffers also, provides a similar (though richer) functionality but without any transport mechanism. MOOS uses a single class for all data transferred, the C++ "CMOOSMsg" which is a thin wrapper around either a string, a double-precision floating point value or an array of bytes. Thus, users of MOOS develop their own marshallings schemes or adopt a standalone library such as protocol buffers.

Interoperability between applications written in dif-weren't middlewares is inefficient as it requires writing code that both ferries data between two different transport mechanisms and converts between similar yet incompatible data representations (marshaling schemes). Goby3 aims to improve this situation by carrying objects of any types that can be serialized to bytes in a cross-platform compatible manner. This includes types from existing middlewares, (e.g., LCM types, ROS msgs) and standalone projects, (e.g., protocol buffers, msg pack).

Nested communications: Nested communications (which is a subset of nested autonomy (Schmidt *et al*., 2016) is a concept that splits the collection of possible communicating entities into subgroups where each subgroup shares a common order of magnitude with regards to data throughput. For example, processes on a single vehicle will likely deliver at similar speeds, regardless of whether they reside on a single computer or multiple computers, given the rate of copper and fiber-based ethernet. However, processes between vehicles will communicate at a vastly different rate if underwater acoustic wireless connections only link the two vehicles.

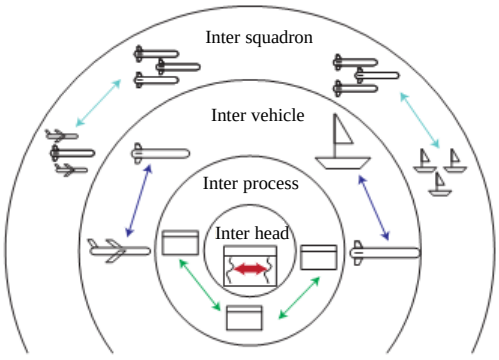

Fig. 1: Illustration of four possible scores

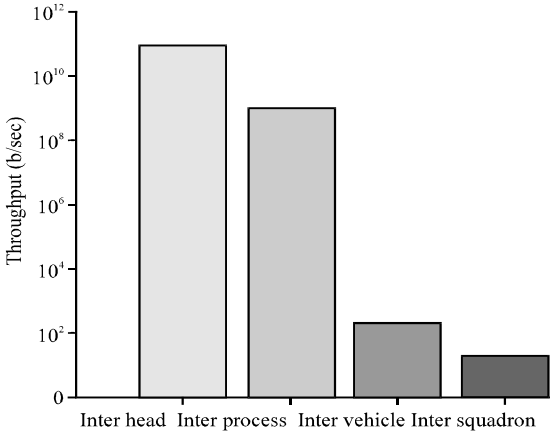

Fig. 2: The order of magnitude data transfer speed

The innermost scope is that which delivers the fastest, out to the slowest outermost scope. All messages sent to outer scopes are automatically forwarded to all inner scopes. An illustration of four possible scores is given in Fig. 1 and the order of magnitude data transfer speeds are shown in Fig. 2.

The publish/subscribe model using nested communications: The publish/subscribe paradigm is common to many of the middlewares, since, its asynchronous nature lends itself well to systems that have many heterogeneous parts operating on different real-time constraints. In the nested implementation of publishing/subscribe in Goby3, an entity publishes its value (of some type) to a "group" at the given nested scope. On the first publication, it is advertised to any existing nodes that are subscribed to that group and type. If no subscribers exist, the published values are not transmitted anywhere. Multiple types can be sent in the same group but customers will only receive (in the form of a callback function) the type (s) they have explicitly subscribed for. This allows both publications and

subscriptions to be strictly typed and not require any parsing or serialization of messages directly by the enduser.

Subscribers can request a variable of a given type or types from a group. Subscriptions will be forwarded into the innermost scope that is fully qualified and if the variable is being published at that scope, all future publications will be escalated to the subscriber's range. This allows data to stay as local as possible until needed by an outer scope, saving bandwidth while maintaining operational flexibility.

Each layer of the nested communications is simple-mounted through two C++ classes, the forwarder, (e.g., InterVehicleForwarder) class and the Portal class, (e.g., InterProcessPortal). The forwarder class is used by entities one scope inside (threads in the case of InterProcessForwarder, processes in the case of InterVehicleForwarder, etc.) which do not directly talk to the transport mecha is at that layer. As the name implies, the for warder passes publications and receives subscribed data from the Portal class via the inner transport layer, (e.g., interthread in the case of InterProcessForwarder). The Portal class communicates on the wire with another instant-actions of the Portal and only one Portal exists for each entity at that scope, (e.g., one InterProcessPortal for each process, one InterVehiclePortal for each vehicle).

The publish/subscribe interface for the end-user application is essentially the same for Portals and Forwarders, except portals need to be configured with the transport related parameters and Forwarders do not, since, they use the internal scope transport mechanism.

The exception to this is the interthread layer which has only one implementation class (the InterProcessTransporter) which is shared by all the threads and is essentially the same as a Portal level design, (since, a Forwarder would be meaningless as there is no further inner scope to forward data through). An example of the interaction between these classes is given in Fig. 3.

Reference implementation: The Goby3 reference implementation is entirely in C++ as defined by the 2011 standard (C++11). C++11 provides numerous new features that are necessary to create this middleware without significant reliance on outside projects such as boost. The major new features used by Goby3 are C++ std:: threads, lambda expressions, std:: function and smart pointers (std:: shared ptr). Goby3 includes a reference implementation that uses three scores and similar transport mechanisms:

Interthread: Zero-copy communication between threads using C++11 shared pointers. Since, no data are copied (just the pointers), the types used at the interthread layer do not need to be serializable into a byte stream. In our experience, multithreading can be error prone and confuse for newcomers to AUV Software. The Goby3 interthread layer allows the transfer of any C++ objects between threads in a publish/subscribe manner that shares the same paradigm and software interface as the outer layers. This allows application designers to write reliable and memory safe multithreaded applications using the same familiar publish/subscribe model without understanding or debugging custom thread data sharing concepts.

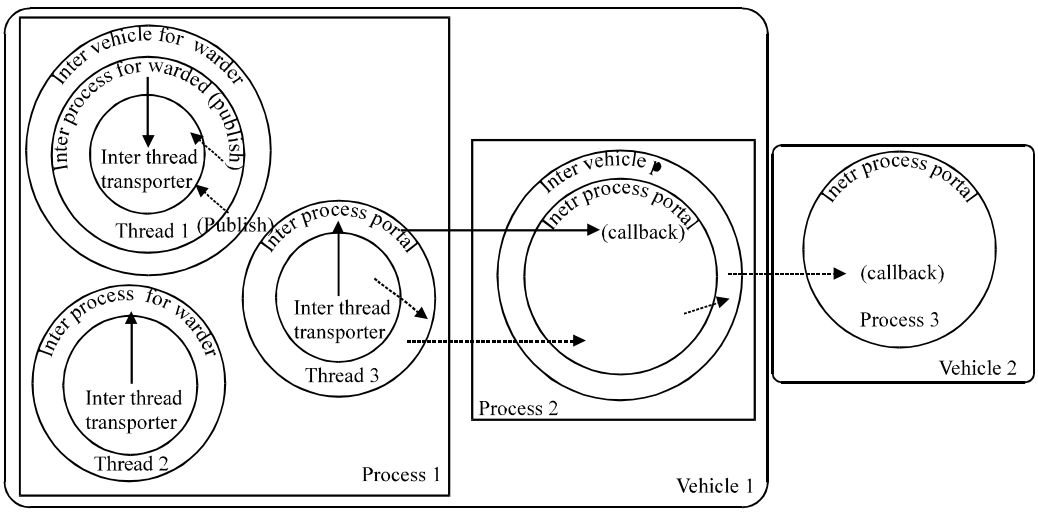

Fig. 3: Interaction between these classes

Interprocess: TCP/IP or UNIX socket communication using ZeroMQ (Hintjens, 2013). This layer of Goby3 uses the ZeroMQ transport layer to allow either single-computer interprocess communications via UNIX sockets or multi-computer, (e.g., connected by a gigabit copper ethernet) interprocess networks using TCP. The assumption is that these processes are all resident on a single vehicle or another node (mooring or topside on the research vessel).

Intervehicle: Acoustic, satellite or other "slow-link" communications using the Goby-Acomms Li-library (Vijayamari *et al*., 2016).

The supported data types (or marshaling schemes) for each of the three scopes in the reference implementation are:

Interthread: Any C++ class.

Interprocess: Any serializable C++ type, (e.g., pro-tool buffers, the Dynamic Compact Control Lan-gauge Version 3 (DCCL3) or msgpack).

Intervehicle: The Dynamic Compact Control Lan-gauge Version 3 (DCCL3)(Roy *et al*., 2016). DCCL3 is an interface description language (based on protocol buffers) and extensible suite of marshaling algorithms specifically designed for extremely low throughput links such as acoustic modems.

RESULTS AND DISCUSSION

The success of some of the primary aims of Goby3 will only be borne out with use by the wider community:bringing the ease of publishing/subscribe to interthread and intervehicle communications and reducing the interopability of different systems by relaxing the marshaling scheme requirements that existing middlewares have.

However, the performance of Goby3's reference in-implementation needs to be acceptable to merit the wider use that will be necessary to assess and realize the aims above. Thus, benchmark testing of the Goby3 reference implementation was performed against the MOOS, LCM and ROS middlewares. Two sizes of messages were tested: a small one on the order of tens of bytes with the exact size depending on the details of the middleware's marshaling scheme and a large one equal to about 1 MB. Ten thousand to one million messages of each size were published and subsequently received by a subscriber. The mean time to publish, transfer and receive each message was calculated. The results of this testing are plotted for the scopes supported by each middleware. This figure shows that the performance of Goby3 is similar or better to that of the comparable middlewares. Some of the minor performance deficit relative to ROS is due to the flexibility of Goby3 (which allows any serializable type whereas ROS only allows ROS msgs).

CONCLUSION

Existing middlewares do not address the "slow link" problem that is very common for marine intervehicle communications. In the researcher's experience, solutions to intervehicle communication tend to be "add-ons" to the leading middleware used for interprocess communication and thus, tend to be difficult to extend or modify when new data needs to be shared between vehicles or the operator topside. Goby3 is designed to provide a standard interface to ease this mismatch.

Existing middlewares tightly couple a required transport layer with a required marshaling scheme. Goby3 relaxes the marshaling system requirement as much as is reasonable, allowing easier development between applications and research groups which "talk" different data were marshaling "languages." Also, the core design of Goby3 does not mandate any particular transport layers so a different choice, (e.g., UDP for interprocess) could be implemented by the user while still using other parts, (e.g., intervehicle and interthread) from the reference implementation. This modularity aims to provide flexibility at the same time as providing working, field quality level code that is ready to use.

Goby3 is open source software (distributed under the LGPL license) and at the time of this writing is in what is considered an "alpha" stage of development. Infrastructure (such as middlewares) are essential pieces of AUV software but difficult to find the time interest or money to create well. Thus, the more shared work that can be done in the AUV community on this topic, the better. Feedback and contributions at this stage are greatly appreciated. The project page for software issue tracking, etc. is https://github.com/GobySoft/goby.

REFERENCES

Benjamin, M.R., H. Schmidt, P.M. Newman and J.J. Leonard, 2010. Nested autonomy for unmanned marine vehicles with MOOS-IvP. J. Field Rob., 27: 834-875.

Benjamin, M.R., J.J. Leonard, H. Schmidt and P.M. Newman, 2009. An Overview of Moos-Ivp and a Brief Users Guide to the Ivp Helm Autonomy Software. Open Archives Initiative, Rijeka, Croatia,.

Hintjens, P., 2013. ZeroMQ: Messaging for Many Applications. O'Reilly Media, Sebastopol, California, ISBN:978-1-449-33406-2, Pages: 328.

Huang, A.S., E. Olson and D.C. Moore, 2010. LCM: Lightweight communications and marshalling. Proceedings of the 2010 IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS), October 18-22, 2010, I EEE, Taipei, Taiwan, ISBN:978-1-4244-6674-0, pp: 4057-4062.

Quigley, M., K. Conley, B. Gerkey, J. Faust and T. Foote *et al*., 2009. ROS: An open-source Robot Operating System. ICRA. Workshop Open Sour. Software, 3: 1-5.

Roy, S.K., M. Harshitha and P. Sharan, 2016. A comparative study of saline and non-saline water in application of tomato yield by using photonic sensor. Proceedings of the 3rd International Conference on Computing for Sustainable Global Development (INDIACom), March 16-18, 2016, IEEE, New Delhi, India, ISBN:978-9-3805-4421-2, pp: 2733-2735.

Schmidt, H., M.R. Benjamin, S.M. Petillo and R. Lum, 2016. Nested Autonomy for Distributed Ocean Sensing. In: Springer Handbook of Ocean Engineering, Manhar, R.D. and I.X. Nikolaos (Eds.). Springer, Berlin, Germany, ISBN:978-3-319-16648-3, pp: 459-480.

Vijayamari, A., K. Sadayandi, S. Sagadevan and P. Singh, 2016. A study of optical, surface morphological and electrical properties of manganese oxide nanoparticles. J. Mater. Sci. Mater. Electron., 3: 2739-2746.