A middle way for robotics middleware

Elena Ceseracciu Daniele Domenichelli Paul Fitzpatrick Giorgio Metta Lorenzo Natale Ali Paikan

***Abstract*— Robotics is changing. The amount of software available (and needed) is growing. For better or worse, the glue that holds that software together, the middleware, has a big impact on its viability. YARP is a middleware for robotics, with over a decade's continuous use on various humanoid robots. YARP was designed to help code survive changes, to easily experiment with new code and integrate with other systems. In a world of constant transition, with a steady stream of hardware and software upgrades, YARP helps code last long enough to make a real impact, and avoid premature loss of good code through unfortunate dependencies. We revise the features of YARP that support this flexibility describing those situations in which they have been practically useful.**

I. Introduction

There is growing attention in robotics to software middleware, some specific to robotics (Player [1], ROS [2], YARP [3], OROCOS [4], Urbi [5], MIRO [6], LCM [7] and MIRA [8]), others more general (ICE [9], CORBA [10], ØMQ [11]). There are several attempts in the literature to compare middleware based on features and performance [12], [13], [8]. Here, we look at another fundamental question that is rarely addressed (since difficult to measure): the question of character.

YARP ("Yet Another Robot Platform") is a middleware for robotics that has been in continuous use on humanoid robots for more than ten years. In character, it is light-weight and non-prescriptive, designed to be easily extended and play well with others.

We look here at the character traits of YARP and illustrate how this has proved helpful in practical cases.

II. Fragmentation or monoculture?

Middleware is a mixed blessing. In the short term it helps a system scale up, but in the long term it can hold it back. Like operating systems, middleware is "sticky" – developers have to make a special effort to make their software usable in other platforms. In the short term, for an individual developer, this is no big deal, since other platforms are irrelevant. But in the longer term or in larger teams this is a real cost, paid for through collaborations that never happen or that get bogged down.

A possible solution is for everyone to get together and agree to use a common platform. But even with the best of intentions a single platform becomes a trap, as more and more components come to rely on fragile quirks of that platform, increasing the cost of any changes, even positive ones. If a platform cannot play well with its peers, there's no reason to expect it to play well with future versions of itself. That implies one of two things. Either there will be a "big bang" change of infrastructure and all the pain that goes with that, or development of the platform will slow to a crawl.

There is a middle way between fragmentation and monoculture, and it is not particularly revolutionary. It is simply this: be polite and considerate of others. Middleware should not assume that the entity on the other side of a network socket is under its control, since sockets are a key opportunity for interoperability. Middleware should not make too many claims on the main thread in a program, or the program's build process, since two middleware making such claims will conflict. And so on. This is not a call for standardization and elaborate RFPs, just individual developers bearing the world outside their middleware in mind, thus benefiting their community today (by facilitating collaborations) and in the future (by reducing the pain of upgrades, and so making radical improvements more practical).

III. Naming

Middleware typically has some mechanism for converting symbolic names into detailed information on how to access a resource (for example, to convert the name "/camera" to "machine 192.168.1.15, port 10012, protocol mjpeg").

We found that naming can become a barrier preventing interoperability and forcing users to resort to inefficient bridging code. With time YARP extended naming support so that it could interface with other systems. At this aim it now offers the following options:

- The YARP name server accepts arbitrary registrations, so a webcam at a random location can be named and read from directly without bridging.
- YARP clients can be directed to use a non-native name server, given an appropriate plugin. For example, there is a plugin for the roscore name server.
- YARP clients can be directed to use multiple name servers, to better support heterogeneous networks.
- YARP clients can be used without any name server for basic tcp connections where host names and socket port numbers are given directly.

A broad-minded nameserver is a boon for interoperation. Of course the *general* case of naming external resources is full of pitfalls, but most *particular* cases are quite simple.

IV. Identifying Connections

Middleware typically plays a role in inter-process communication. This is a great opportunity for interoperability. Even when interoperability is not an immediate priority, it is worth taking some simple steps to avoid unnecessarily hobbling it.

For example, most every middleware has some kind of regular TCP protocol. Suppose the other side of a socket is

not controlled by your middleware. They may not know for sure what protocol we're speaking. Just as it is super-helpful for files lying around in a file system to have a "fingerprint" or "magic number" comprising a recognizable pattern of bytes early in the file, it is helpful if TCP connections do the same.

For example, HTTP responses begin with "HTTP/", and YARP's native protocols start with variants of "YA....RP" or other identifying strings. This makes these types of messages easy to identify. This may seem trivial, however in contrast, another robotics middleware uses a protocol that begins with two 4-byte integers representing lengths, which in principle could have any value. A quick fingerprint up front could be very been handy for interoperation, including with future versions of that middleware.

V. Speaking many languages

There are situations in which a middleware should plan to speak many protocols. Here are situations end-users hit that middleware developers do not see:

- You need to use a novel kind of network not broadly available.
- You have a network-aware piece of hardware and you would like to talk to it without bridging.
- You badly need to interoperate at the network level with software from a different community.
- You have a problem with some aspect of current behavior, and the middleware developers just shrug.

YARP originally used QNX message passing, so its own "native" protocol is lost in the mists of time. Having a plugin mechanism for protocols allowed developers and users to extend the original set of protocols and adapt YARP depending on their needs.

In YARP there are basic options like tcp, udp, multicast, and shared memory, suitable for different trade-offs between reliability, speed, and bandwidth. In addition it supports other "edge" protocols (often over regular tcp) that are useful for talking to the outside world, such as XMLRPC, HTTP with JSON, MJPEG, plain text.

Finally there are some notable examples in which specific users' needs have been elegantly and efficiently solved with the implementation of custom protocols:

- A special purpose protocol plugin was made to carry raw Bayer pattern image streams and decode them on the receiver side, rather than decoding them closer to the hardware. This saves bandwidth without introducing compression artifacts, and it is fully backward compatible, since as far as user code is concerned it receives color images as normal. When there are multiple receivers, it does increase overall CPU usage, which is the main trade-off. As a side benefit, users have the freedom to choose how colors are reconstructed, and this choice can be made per-receiver. A protocol like this is very useful, but so specific to a class of camera that it would be unlikely to get supported in a middleware without a culture of protocol plugins.
- A set of plugins were added to support YARP-ROS interoperability, via XML/RPC and TCPROS.
- A new protocol was introduced to experiment with the concept of priority arbitration into YARP, so that messages from different origins arriving at the same target could inhibit each other in defined ways. This plugin made use of the fact that plugins can be chained (currently in a rather crude way), so that the new priority mechanism could be overlaid on existing protocols.

VI. Conclusions

Most literature on software middleware for robotics focuses on comparing performance. In our experience, this is not the most important characteristic of a software middleware. In the fast and ever evolving world of robotics, a good middleware must be designed for change and, above all, interoperability. Who should care about interoperability between multiple middleware? All of us. We need at least confidence that the middleware we use can interoperate with a future version of itself, so our users and their collaborators do not end up in a middleware muddle.

VII. Acknowledgements

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7 ICT) under grant agreement No. 270273 (Xperience).

References

[1] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[3] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[4] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3, 2001, pp. 2523 – 2528 vol.3.

[5] G. Technologies, "Urbi," http://http://www.urbiforge.org.

[6] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro - middleware for mobile robot applications," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 493–497, Dec. 2002. [Online]. Available: http://dx.doi.org/10.1109/TRA.2002.802930

[7] A. Huang, E. Olson, and D. Moore, "LCM: Lightweight communications and marshalling," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

[8] E. Einhorn, T. Langner, R. Stricker, C. Martin, and H. Gross, "Mira - middleware for robotic applications," in *International Conference on Intelligent Robots and Systems*, 2012.

[9] Z. Inc., "Internet communications engine," http://zeroc.com/ice.html.

[10] OMG, "Common Object Request Broker Architecture (CORBA/IIOP).v3.1," OMG, Tech. Rep., Jan. 2008. [Online]. Available: http://www.omg.org/spec/CORBA/3.1/

[11] "ØMQ, The Intelligent Transport Layer," http://www.zeromq.org/.

[12] J. F. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Auton. Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[13] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics: A survey," in *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, sept. 2008, pp. 736 –742.