Middleware and Software Frameworks in Robotics

– Applicability to Small Unmanned Vehicles –

Daniel Serrano
Department of Intelligent Systems, ASCAMM Technology Center
Parc Tecnològic del Vallès, Av. Universitat Autònoma, 23
08290 Cerdanyola del Vallès
SPAIN

dserrano@ascamm.com

ABSTRACT

This paper intends to present some of the software architecture and middlewares that have been developed in the last decade for robotics. A robot is a system that presents a set of particular requirements to interact in real-time in a dynamic, uncertain environment. The paper covers the technical details of some of the software frameworks widely established in the robotics community.

1.0 INTRODUCTION

Unmanned vehicle are highly complex systems. They are required to execute several tasks concurrently, while monitoring and managing unexpected events. These systems typically integrate several sensors and actuators. In between, a set of processes work together to solve some of the most common problems in robotics.

For instance, sensors are not perfect; they usually induce noise and therefore errors in the system. This implies that the perception of the environment, and some associated process such as the estimation of the pose of the robot, are subject to uncertainty.

Writing software for robots is difficult, particularly as the scale and scope of robotics continues to grow. Different types of robots can have wildly varying hardware, making code reuse nontrivial. Since the required breadth of expertise is well beyond the capabilities of any single researcher, robotics software architectures must also support large-scale software integration efforts. [2].

All this complexity leads to the need of well-designed robotics architectures as more and more robots are present in different contexts nowadays. The need for common robot architectures and frameworks to share and reuse software and algorithms across different organization and systems seems convenient.

The term robot architecture is often used to refer to two related, but distinct, concepts. Architectural structure refers to how a system is divided into subsystems and how those subsystems interact. The structure of a robot system is often represented informally using traditional boxes and arrows diagrams or more formally using techniques such as unified modeling language (UML). In contrast, architectural style refers to the computational concepts that underlie a given system. For instance, one robot system might use a publish–subscribe message passing style of communication, while another may use a more synchronous client–server approach [1].

2.0 ROBOTICS SOFTWARE FRAMEWORKS

In the recent years, various software frameworks and middleware have become very popular and widely adopted in industry and research communities.

A software framework is a universal, reusable software platform to develop applications, products and solutions. Software frameworks include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together the different components to enable development of a project or solution. A middleware is usually focused on some complex functionality such as robot control or inter-process communication. Both software frameworks and middlewares are development tools that employ reusable software components, following specific design patterns.

They tackle the seamless integration of software algorithms on different hardware platforms. Some of them started as a communication layer but they have became nowadays very complex entities, that provide, from hardware drivers, algorithms and libraries (PLAYER, ORCA, YARP), to complete sets of algorithms and services such as hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes, and package management (ROS).

Some of these frameworks and middleware have become so widely accepted that they are usually proposed as candidates to ensure interoperability. However, looking at the requirements associated to an interoperability standard, the recommendation is that, even though software framework and middleware are really valuable and relevant for unmanned systems development, they are not considered standards and therefore, they do not satisfy the needs in terms of interoperability. They should remain at the platform level (subsystem level) as they definitely speed up the development and adaptation of a platform, and enable software re-usability.

3.0 EXAMPLE OF ROBOTICS SOFTWARE FRAMEWORKS

To meet these challenges, many robotics researchers, including ourselves, have previously created a wide variety of frameworks to manage complexity and facilitate rapid prototyping of software for experiments, resulting in the many robotic software systems currently used in academia and industry [3]. Each of these frameworks was designed for a particular purpose, perhaps in response to perceived weaknesses of other available frameworks, or to place emphasis on aspects which were seen as most important in the design process [2].

The following table shows some of the most relevant software frameworks and tools in robotics:

- ROS (Robotics Operating System)
- Microsoft Robotics Studio
- NXJ (Open source Java for the Lego robot kit)
- Player (robot framework)
- Orocos (C++ framework for component based robot control)
- Rock (Robot Construction kit)
- Orca (robot framework)
- MOOS (robot framework)
- CARMEN (robot simulator)
- Simbad robot simulator
- Gazebo (multi-robot simulator)

3.0 ROBOT OPERATING SYSTEM (ROS)

ROS is an open source robot operating system. It provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

ROS provides a structured communications layer above the host operating systems of a heterogeneous compute cluster. It was design with a philosophy of modular, tools-based software development. It aims at maximizing the reusability of available robot sensor visualizations, sensor fusion and control algorithms. It has gradually managed to gather around a lot of developers and it has become a sort of standard framework for robotic platforms (mostly in research).

ROS defines standard message types for commonly used robot sensor data such as images, inertial measurements, GPS, odometry etc. for communicating between nodes. Thus separate data structures do not need to be explicitly defined for integrating different components. However, these messages have been created on demand and they are continuously evolving as new needs are identified.

4.0 MISSION ORIENTED OPERATING SUITE (MOOS)

Mission Oriented Operating Suite (MOOS) is very popular in the maritime robotics community. It is a C++ cross platform middleware for robotics research, but there are also interfaces from Java and Python. It was created by Paul Newman between 2001-2005, with help of students and researchers at Oxford and at MIT. It is composed by a set of layers:

- Core MOOS provides a very robust, and easy to use, network based communications architecture designed as two libraries (MOOSLib, MOOSGenLib) and a lightweight communications hub (MOOSDB).
- Essential MOOS is a set of applications built upon Core MOOS that provide different functionalities commonly found in unmanned systems (i.e. pLogger, pScheduler, pMOOSBridge).

The entire software package provides a variety of tools and applications, some of them still with a maritime flavour.

MOOS has a star-like topology. Each application within a MOOS community (a MOOSApp) has a connection to a single “MOOS Database” (called MOOSDB) that lies at the heart of the software suite. All communication happens via this central “server” application. This is an example of the contents of a MOOS message:

Variable	Meaning
Name	The name of the data
String Val	Data in string format
Double Val	Numeric double float data
Source	Name of client that sent this data to the MOOSDB
Time	Time at which the data was written
Data Type	Type of data (STRING or DOUBLE)
Message Type	Type of Message (usually NOTIFICATION)
Source Community	The community to which the source process belongs

Figure 1: MOOS message content

There are some projects building upon MOOS and, therefore, extending its capabilities. The most relevant is MOOS-ivp, which provides autonomy on marine robots (and others).

5.0 PLAYER

The Player Project [4] creates a free software tool that enables research in robot and sensor systems. The Player robot server is probably the most widely used robot control interface in the world. Its simulation back ends, Stage and Gazebo, are also very widely used. An Integrated platform for experiments in mobile robotics and sensor networks, Player/Stage provides tools for rapid code development and large-scale testing

Player [5] provides a network interface to a variety of robot and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. Player supports multiple concurrent client connections to devices, creating new possibilities for distributed and collaborative sensing and control. Player supports a wide variety of mobile robots and accessories. Look here for a list of currently supported components.

Stage simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry. Stage devices present a standard Player interface so few or no changes are required to move between simulation and hardware. Many controllers designed in Stage have been demonstrated to work on real robots.

Gazebo is a multi-robot simulator for outdoor environments. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics). Gazebo presents a standard Player interface in addition to its own native interface. Controllers written for the Stage simulator can generally be used with Gazebo without modification (and vice-versa).

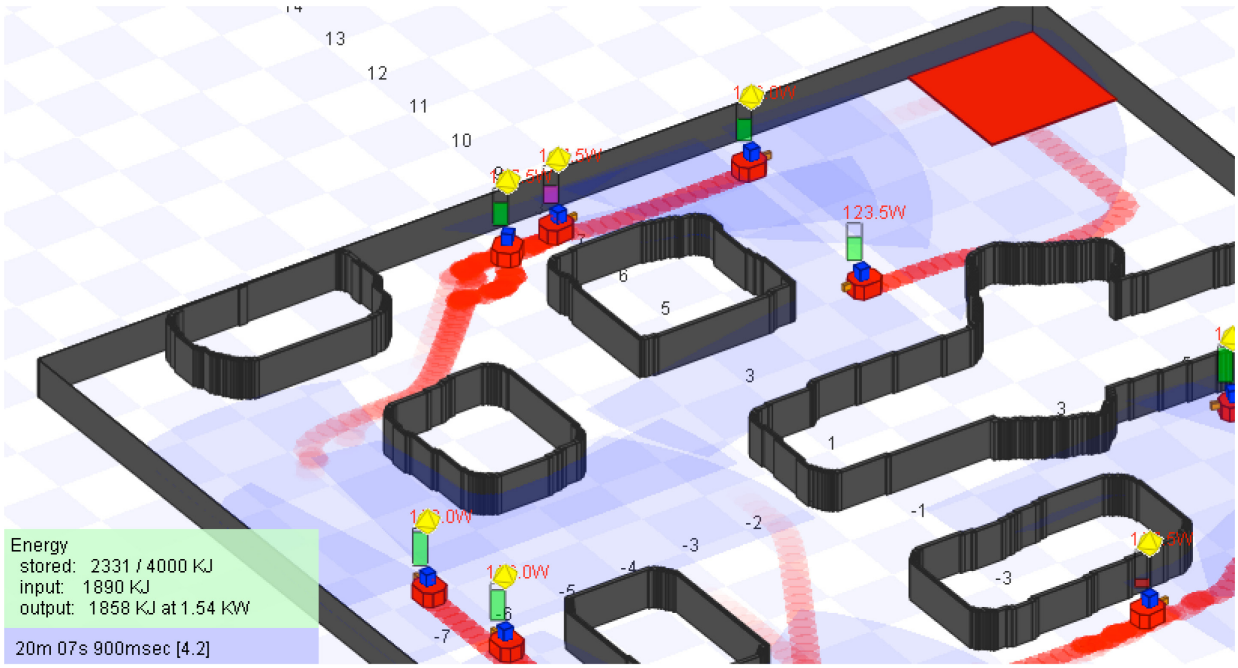

Figure 2: Player/Stage simulation

6.0 YET ANOTHER ROBOT PLATFORM (YARP)

YARP [6] is an open source library that developed to support software development on humanoid robotics. YARP tries to facilitate code exchange between researchers, especially when this speeds up the time it takes to develop a platform and use it for research.

The goal of YARP is to minimize the effort devoted to infrastructure-level software development by facilitating code reuse, modularity and so maximize research-level development and collaboration. Humanoid robotics is a "bleeding edge" field of research, with constant flux in sensors, actuators, and processors. Code reuse and maintenance is therefore a significant challenge.

In short, the main features of YARP include support for inter-process communication, image processing as well as a class hierarchy to ease code reuse across different hardware platforms. YARP is currently used and tested on Windows, Linux and QNX6 which are common operating systems used in robotics.

YARP is written by and for researchers in humanoid robotics, who find themselves with a complicated pile of hardware to control and with an equally complicated pile of software. The heart of YARP is a communications mechanism to make writing and running such processes as easy as possible. Even where mobility is required this is not a limiting factor if tethers or wireless communication are acceptable.

7.0 OROCOS

"Orocos" [7] is the acronym of the Open Robot Control Software project. The project's aim is to develop a general-purpose, free software, and modular framework for robotand machine control. The Orocos project supports 4 C++ libraries: the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the Orocos Component Library.

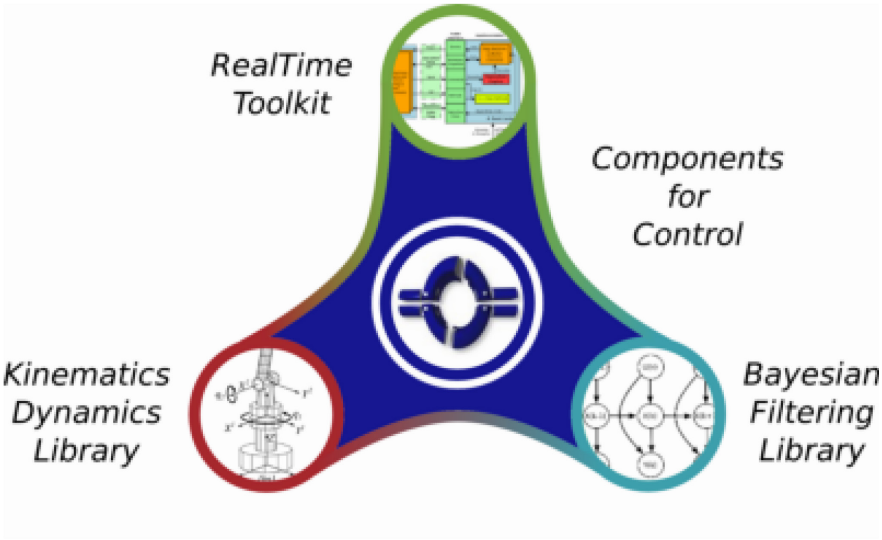

Figure 3: Orocos Libraries

- The Orocos Real-Time Toolkit (RTT) is not an application in itself, but it provides the infrastructure and the functionalities to build robotics applications in C++. The emphasis is on real-time, on-line interactive and component based applications.

- The Orocos Components Library (OCL) provides some ready to use control components. Both Component management and Components for control and hardware access are available.

- The Orocos Kinematics and Dynamics Library (KDL) is a C++ library which allows calculating kinematic chains in real-time.

- The Orocos Bayesian Filtering Library (BFL) provides an application independent framework for inference in Dynamic Bayesian Networks, i.e., recursive information processing and estimation algorithms based on Bayes' rule, such as (Extended) Kalman Filters, Particle Filters (Sequential Monte methods), etc.

Orocos is a free software project; hence its code and documentation are released under Free Software licenses.

REFERENCES

[1] B. Siciliano and O. Kathib, "Handbook of Robotics", Springer-Verlag Berlin Heidelberg 2008

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source robot operating system". In ICRA Workshop on Open Source Software, 2009.

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

[4] Richard T. Vaughan. "Stage: A Multiple Robot Simulator". Technical Report IRIS-00-394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 2000.

[5] Player/Stage website http://playerstage.sourceforge.net/

[6] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. International Journal on Advanced Robotics Systems, 3(1):43–48, 2006.

[7] The OROCOS project website http://www.orocos.org/