Robotics
Laboratory

Christian Rossi

Academic Year 2023-2024

Abstract

The course is composed by a set of lectures on autonomous robotics, ranging from the main architectural patterns in mobile robots and autonomous vehicles to the description of sensing and planning algorithms for autonomous navigation. The course outline is:

- Mobile robots' kinematics.
- Sensors and perception.
- Robot localization and map building.
- Simultaneous Localization and Mapping (SLAM).
- Path planning and collision avoidance.
- Robot development via ROS.

Contents

CHAPTER 1

Introduction

1.1 Middleware origins and usage

Middleware was introduced by d'Agapeyeff in 1968, the concept of a wrapper emerged in the 1980s as a bridge between legacy systems and new applications. Today, it is prevalent across various domains, including robotics. Examples outside of robotics include Android, SOAP, and Web Services.

The concept of Middleware is widely recognized in software engineering. It serves as a computational layer, acting as a bridge between applications and low-level details. It's important to note that Middleware is more than just a collection of APIs and libraries.

Issues The challenge lies in fostering cooperation between hardware and software components. Robotics systems face architectural disparities that affect their integration. Ensuring software reusability and modularity is also a critical concern in this context.

Main features The key attributes of middlewares include:

- *Portability*: offering a unified programming model irrespective of programming language or system architecture.
- *Reliability*: middlewares undergo independent testing, enabling the development of robot controllers without the need to delve into low-level details, while leveraging robust libraries.
- *Manage the complexity*: handling low-level aspects through internal libraries and drivers within the middleware, thereby reducing programming errors and speeding up development time.

1.1.1 Middlewares

Several middleware have been developed in recent years:

- *OROCOS*: originating in December 2000 as an initiative of the EURON mailing list, OROCOS evolved into a European project with three partners: K.U. Leuven (Belgium), LAAS Toulouse (France), and KTH Stockholm (Sweden). The requirements for OROCOS

encompass open source licensing, modularity, flexibility, independence from specific robot industries, compatibility with various devices, software components covering kinematics, dynamics, planning, sensors, and controllers, and a lack of dependency on a singular programming language. The structure includes:

  - Real-Time Toolkit (RTT): offering infrastructure and functionalities tailored for real-time robot systems and component-based applications.
  - Component Library (OCL): supplying ready-to-use components such as drivers, debugging tools, path planners, and task planners.
  - Bayesian Filtering Library (BFL): featuring an application-independent framework encompassing (Extended) Kalman Filter and Particle Filter functionalities.
  - Kinematics and Dynamics Library (KDL): facilitating real-time computations for kinematics and dynamics.

- *ORCA*: the project's objective is to emphasize software reuse in both scientific and industrial applications. Its key properties include enabling software reuse through the definition of commonly-used interfaces, simplifying software reuse via high-level libraries, and encouraging software reuse through regularly updated software repositories. ORCA defines itself as an "unconstrained component-based system."

  The primary distinction between OROCOS and ORCA lies in their communication toolkits. OROCOS utilizes CORBA, whereas ORCA employs ICE. ICE, a contemporary framework created by ZeroC, functions as an open-source commercial communication system. ICE offers two fundamental services: the IceGrid registry (Naming service), which facilitates logical mapping between various components, and the IceStorm service (event service), which forms the foundation for publisher-subscriber architecture.

- *OpenRTM*: RT-Middleware (RTM) serves as a widely adopted platform standard for assembling robot systems by integrating software modules known as robot functional elements (RTCs). These modules include components such as camera, stereo vision, face recognition, microphone, speech recognition, conversational, head and arm, and speech synthesis. OpenRTM-AIST (Advanced Industrial Science and Technology) is built upon CORBA technology to realize the extended specifications of RTC implementation.

- *BRICS*: the objective is to uncover the most effective strategies for developing robotic systems by examining several critical areas: Initially, by thoroughly examining the weaknesses found in current robotic projects. Subsequently, by delving into the integration between hardware and software components within these systems. Thirdly, by advocating for the incorporation of model-driven engineering principles in the development process. Moreover, by creating a tailored Integrated Development Environment (IDE) specifically for robotic projects, known as BRIDE. Finally, by defining benchmarks aimed at evaluating the strength and effectiveness of projects based on BRICS principles.

- *ROS*: introduced by Willow Garage in 2009, the Robot Operating System (ROS) serves as a meta-operating system tailored for robotics, boasting a diverse ecosystem replete with tools and programs. ROS has expanded to encompass a vast global community of users. The developer community stands out as one of ROS's most significant features.

CHAPTER 2

Robot Operating System

2.1 ROS file system

The foundation of the ROS file system revolves around packages, depicted in the diagram below.

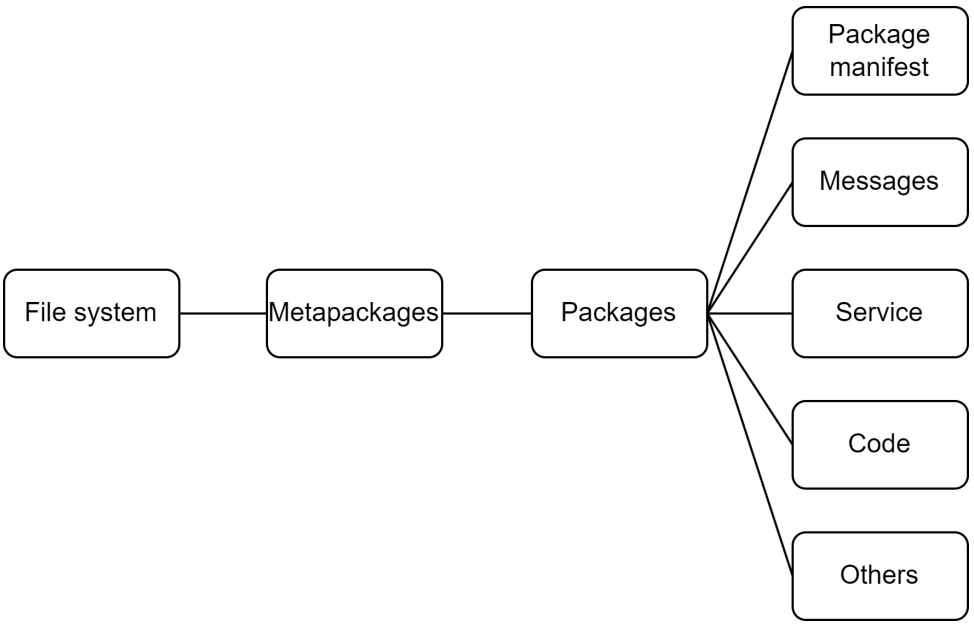

Figure 2.1: ROS file system

Packages serve as the fundamental units within the ROS file system. They are essential references for most ROS commands and encompass nodes, messages, and services. Each package is described by a `package.xml` file and acts as a mandatory container for its contents.

Additionally, there are metapackages, which aggregate logically related elements. Unlike packages, metapackages are not directly utilized when navigating the ROS file system. They contain other packages and are described by a package.xml file as well, but they are not obligatory components.