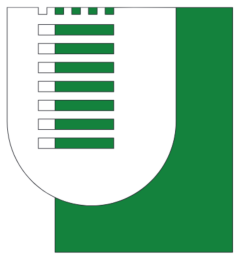
Robot Operating Systems 2

Lecture 1: Middleware Fundamentals

Roberto Masocco
roberto.masocco@uniroma2.it

University of Rome "Tor Vergata"
Department of Civil Engineering and Computer Science Engineering
Intelligent Systems Lab

Month Day, Year



TOR VERGATA
UNIVERSITY OF ROME

School of Engineering

Roadmap

Roadmap

What is middleware?

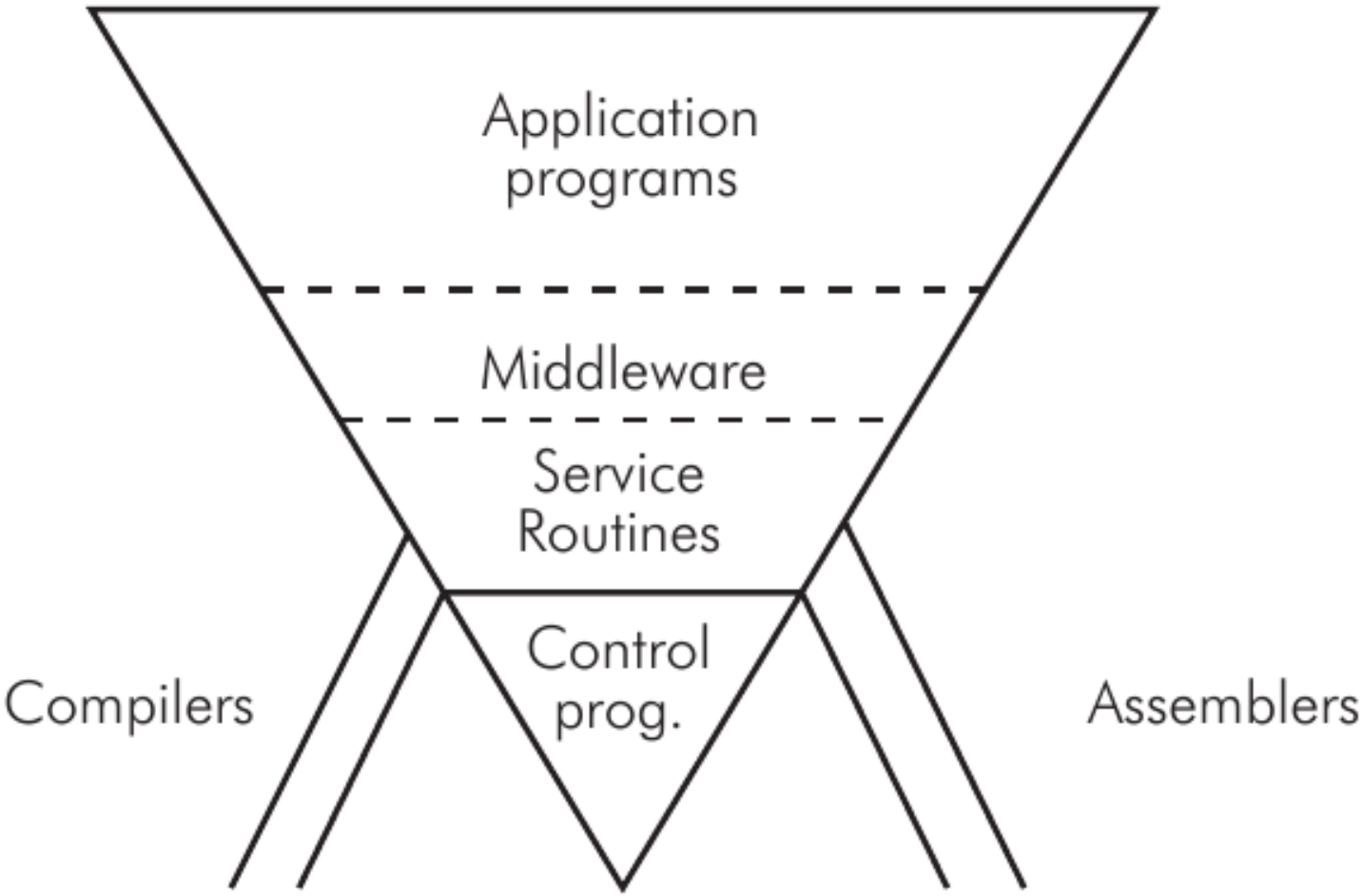


Figure 1: Software organization in a generic computer system

What is middleware?

Definition of middleware

The term **middleware** identifies a kind of software that offers common services and functionalities to applications in addition to what an operating system usually does.

Middlewares are usually implemented as **libraries** that application programmers can use via appropriate **APIs**.

Middleware in robotics

Classic problems arising when developing software for autonomous systems:

- definition of tasks;
- hardware integration;
- software organization and maintenance;
- communication and data exchange (involves both hardware and software!);
- debugging and testing.

Middlewares can offer services to tackle and solve each one!

Data Distribution Service

Definition of DDS

A DDS is a **publish-subscribe middleware** that handles communications between **real-time** systems and software over the network.

DDS implementations follow an open standard that defines:

- serialization and deserialization of data packets;
- automatic discovery of **DDS participants** (over **multicast-IP/UDP**) and transmission of data (over **unicast-IP/UDP**);
- **security protocols** and cryptographic operations;
- enforcing of **Quality of Service** policies to organize transmissions (specifying things like **queue sizes**, **best-effort** or **reliable** transmissions...).

DDSs are currently used in automotive, aerospace, military...

Data Distribution Service

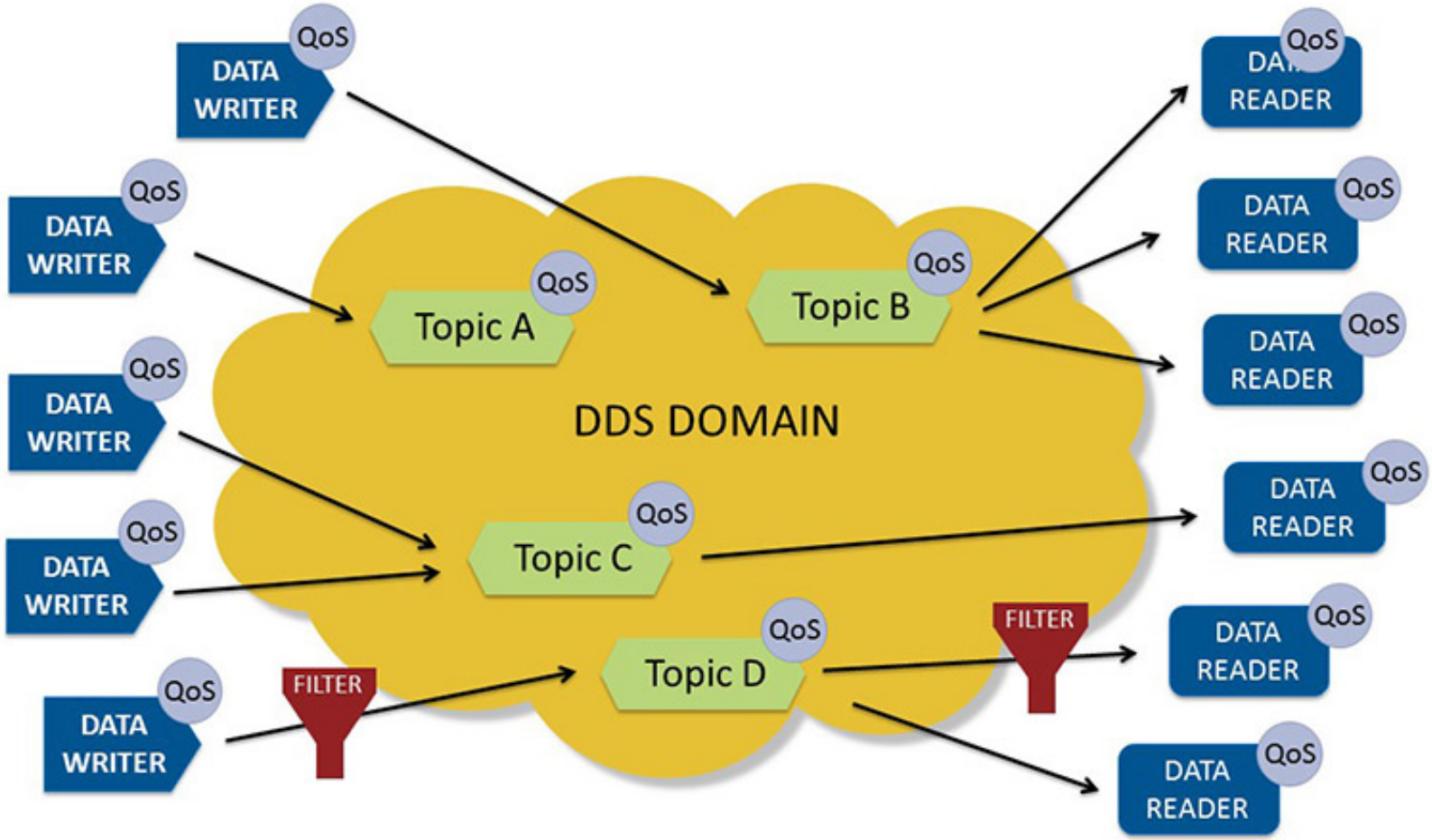


Figure 2: Scheme of a DDS-based network

Data Distribution Service

DDS participants can either **publish to** or **subscribe to** a **topic**.

Definition of DDS topic

A DDS topic is uniquely identified by three things:

- a **name**, i.e. a human-readable character string;
- an **interface**, i.e. a custom packet format that specifies what data is carried over it (e.g. strings, numbers, arrays...);
- a **QoS policy** that specifies how transmissions should be performed.

Changing even only one of the above results in a completely different topic!

Roadmap

What is ROS 2?

ROS 2 is a **DDS-based**, **open-source** middleware for robotic applications. It allows developers to build and manage **distributed control architectures** made of many modules, usually referred to as **nodes**.

Figure 3: ROS 2 logo

What is ROS 2?

ROS 2 currently supports **C++** and **Python** for application programming, and runs natively on **Ubuntu Linux 20.04**.

New versions are periodically released as **distributions**: the current LTS one is **Foxy Fitzroy** and the latest one today is **Galactic Geochelone**; the development version is **Rolling Ridley** and can only be compiled from source.

Figure 3: ROS 2 logo

Main Features

As a middleware, it offers the following services to roboticists:

- **three inter-process communication (IPC) paradigms**, easy to set up and based on the DDS;
- organization of software packages, allowing for **redistribution and code reuse**, thanks to the **colcon** package manager;
- module configuration tools: **node parameters** and **launch files**;
- integrated **logging subsystem** (involves both console and log files);
- CLI **introspection tools** for debugging and testing;
- integration with **simulators** (e.g. Gazebo) and **data visualizers** (e.g. RViz);

Main Features

As a middleware, it offers the following services to roboticists:

- **three inter-process communication (IPC) paradigms**, easy to set up and based on the DDS;
- organization of software packages, allowing for **redistribution and code reuse**, thanks to the **colcon** package manager;
- module configuration tools: **node parameters** and **launch files**;
- integrated **logging subsystem** (involves both console and log files);
- CLI **introspection tools** for debugging and testing;
- integration with **simulators** (e.g. Gazebo) and **data visualizers** (e.g. RViz);
- and much more...

Flaws

ROS 2 biggest flaws (as of today)

The main concerns arise when developing low-level stuff:

- the DDS layer is almost completely abstracted, so some network configurations are impossible;
- the internal job scheduling algorithm (the **executor**) is **not suited for hard real-time applications**.

Flaws

ROS 2 biggest flaws (as of today)

The main concerns arise when developing low-level stuff:

- the DDS layer is almost completely abstracted, so some network configurations are impossible;
- the internal job scheduling algorithm (the **executor**) is **not suited for hard real-time applications**.

What to do when development gets to a really low level?

- Use **micro-ROS**: ROS 2 on microcontrollers and different communication interfaces.
- Hand off stuff to **microcontrollers**.
- Use something else.

Job Executor

TODO:

- event-based explanation
- node jobs+executor+callback scheme
- "to be addressed in new upcoming releases+micro-ROS"

Roadmap

Frame Name

Frame contents