Middleware
Introduction from the robotics point of view

Václav Hlaváč
Czech Technical University in Prague (ČVUT)
Czech Institute of Informatics, Robotics, and Cybernetics (CIIRC)
Prague 6, Zikova 4, Czech Republic
hlavac@ciirc.cvut.cz
http://people.ciirc.cvut.cz/hlavac/

Courtesy: Vladimír Petrík, Libor Wagner
Middleware in computer science

- Middleware is software providing services to applications beyond those available from the operating system.
- Middleware makes it easier for software developers to perform communication and input/output.
- Most commonly used in the context of distributed applications.
- More specifically: dash in “client-server”.
- Also used in a sense of: a software driver, an abstraction layer hiding details of hardware and software from an application.
1. **Message oriented middleware**: asynchronous store and forward application messaging.

2. **Object middleware**: object request brokers, manages communication between objects.

3. **RPC middleware**: synchronous interaction, usually within an application.

4. **Database middleware**: direct access to data structures allowing interaction with DB directly.

5. **Transaction middleware**: transaction processing as well as web application servers.

6. **Portals**: enterprise portal servers allowing access from user’s desktop to back end systems and services.
CORBA, my first middleware

- **Common Object Request Broker Architecture (CORBA)** is a standard enabling software components written in multiple computer languages and running on multiple computers to work together.

- Used in our ActIPret project (2001-2004) to control a robot with various vision sensors.

- It was too heavy and slow.

- There was a need to write a lightweight middleware allowing real-time interaction.
Middleware in robotics

- Glue software to connect software and hardware components together.
- Often, communication between components is considered to be middleware.
- The look is from the software developer perspective.
- In addition, all application-independent helping composition of subsystems into larger systems are often included too.
- Middleware should be invisible.
Four minimal primitive concepts

- **Communication**: components must exchange information (data, events, commands,...), and how this exchange is done is an important property of the composite system.

- **Computation**: each component performs certain computations that are necessary to provide the functionality that is expected from the system.

- **Configuration**: components should be usable in more than one possible configuration (i.e., concrete settings for each of their variable parameters).

- **Coordination**: at the system level. Involves: decision making, scheduling, (de)activating subsystems and/or their interconnections, etc.
Robotic middleware examples

- OpenRDK
- Urbi – for complex organization of components, French company GOSTAI
- MIRO, based on CORBA,
- OpenNI – middleware for 3D sensing,

Middleware V. Hlaváč has some experience with
- RSB - U of Bielefeld
- ROS
Three issues to be tackled when developing robot software:

1. Sequential programming ill-suited to asynchronous world.
2. Must manage significant complexity.
3. Details of a specific robot hardware have to be abstracted.
Ad 1. Avoiding seq. programming

**Callback:**

- Function that’s called whenever data is available for processing.
- Asynchronous: callback can happen anytime.

**Examples:**

- An image is read from the camera.
- A bumper tells that the robot hit something.
Ad 2. Tackling complexity

Code organization

- Separate processes: cameras, odometry, map creation ...
  Can be separated out and interact through an interface.

- Interfaces: SW processes; in ROS “nodes” communicate about shared “topics”.

- Publish/subscribe: each piece of sw receives only messages it requests.

May 2016  V. Hlaváč, A3M33IRO Intelligent Robotics
Ad 3. Hardware abstraction

Hardware-Independent Software

- Face Detection
- Obstacle Detection
- Obstacle
- Map Building

Device-Specific Drivers

- Camera
- Laser Scanner
- Image Message
- PointCloud
- PointCloud

Interface
ROS Robot Operating System

Solves all three above discussed issues (callbacks, interface, hardware).

- Initiated by Willow Garage.
- Meta-operating system.
- Message passing.
- Debugging tools.
- Visualization tools.
- Software Management (compiling, packaging).

- Libraries.
- Hardware agnostic.
- Free + open source.
- Suitable for large scale research.
- Emphasis on distributed computing.
- Ubuntu Linux is the only supported platform.
- Support for other platforms, including Windows, is experimental.
ROS design goals

- **Peer-to-peer**: ROS components, potentially on different hosts, are connected in peer-to-peer topology.
- **Tool-based**: Microkernel design, with large number of small tools, used to build, run and analyze ROS components.
- **Multi-lingual**: ROS components can be written in various languages. Commonly: Python, C++, Lisp.
- **Thin**: Drivers and algorithms are encouraged to be written in separated libraries.
- **Open-Source**: ROS is distributed under terms of the BSD license.
ROS concepts

- **Node:**
  A single computation unit (component).

- **Message:**
  Data structure used by nodes to communicate.

- **Topic:**
  Broadcast communication between nodes (publisher-subscriber architecture).

- **Service:**
  Synchronous communication between nodes (client-server architecture).

- **Package:**
  A software unit solving a specific task, e.g. navigation.
Node

- A single process that performs a particular computation.
- ROS supported application is often composed of a large number of nodes.
- Nodes communicate with each other by passing messages through topic or service.
- Connection between two nodes is accomplished through roscore, which acts as a name server. There is only one roscore that secures the communication.
Example: initiating topic communication
ROS tools and packages

- rviz: visualisation tool.
- rosbag: allows recording all communication between nodes and play it back later.
- rqt_graph: visualizes the graph of ROS application.
- rosparam: can manipulate data on the ROS parameter server.
ActionLib

- For preemptable tasks / Longtime running tasks.
- Communication build on top of ROS messages.
- Action is specified by three messages: goal, feedback and result.
- For example Goto: Planning, Filtering, Trajectory execution (can be interrupted).
Transformation (tf)

- Transformations in a tree structure.
- Any transformation is relative to any coordinate frame.
- Buffered in time, i.e. it can be asked what was the transformation in the past instant.
Unified Robot Description Format
- XML format
- links (origin, visual and collision model, color)
- joints (origin, type)

Xacro
- XML macro language
- more readable XML
- macros, parameters, maths
<xacro:macro name="table_leg" params="*origin parent suffix color height">
    <link name="${suffix}"/>
    <xacro:default inertial/>
    <visual>
        <geometry>
            <box size="0.05 0.05 ${height-0.050}"/>
        </geometry>
        <material name="${color}"/>
        <origin xyz="0 0 -${(height+0.05)/2}"/>
    </visual>
</link>

<joint name="${suffix}joint_ftable_leg" type="fixed">
    <insert_block name="origin"/>
    <parent link="${parent}"/>
    <child link="${suffix}"/>
</joint>
</xacro:macro>
URDF, rviz visualization
SMACH – State MACHine

- Implements higher-Level behaviors via finite automata.
- Hierarchical (every state machine can be the state in another state machine).
- Concurrence containers (parallel).
Gazebo

- Multi-robot dynamic simulator for outdoor environments.
- Started in 2002, independent of ROS.
- There is an interface to ROS.
- Plenty of sensors (laser, camera, Kinect, GPS).