Software componentization for robotics

Mixing middleware, architectures, and several robot types
Giorgio Metta  
Scientific Director  
Italian Institute of Technology

L. Natale, D. Pucci, U. Pattacini  
Italian Institute of Technology
Once upon a time...
The sad fate of most robot software

- Writing software is difficult and time consuming
- Our software tends to die with our projects/students
- Sad! Software collaboration *speeds things up*
- Code sharing could *promote successful components*
Barriers to software collaboration

- Groups developing on different robots face obstacles
  - Differences in sensors, actuators, bodies...
  - Differences in processors, operating systems, libraries, frameworks, languages, compilers...

- Lack of reward for producing reusable code

- Research groups that all use a specific robot (Khepera, Pioneer, AIBO, ...) often form a natural software community
  - But each alone is a small subset of robotics

The popular robots in year 2001
YARP is an open-source (BSD) middleware for humanoid robotics

History
- An MIT / Univ. of Genoa collaboration
- Born on Kismet, grew on COG, under QNX
- With a major overhaul, now used by RobotCub consortium
- Exists as an independent open source project (GitHub)
- C++ source code (mostly)
philosophy

- One processor is never enough
- Modularity
- Minimal interference
- Stopping (the robot) hurts
- Humble approach (thin middleware)
- Exploit diversity

Exploit diversity: portability

- Operating system portability:
  - Adaptive Communication Environment, C++ OS wrapper: e.g. threads, semaphores, sockets

- Development environment portability:
  - CMake

- Language portability:
  - Via Swig: Java (Matlab), Perl, Python, C#
Achieving modularity

- Factor out **details of data flow between programs** from program source code
  - Data flow is very specific to robot platform, experimental setup, network layout, communication protocol, etc.
  - Useful to keep “algorithm” and “plumbing” separate

- Factor out **details of devices used by programs** from program source code
  - The devices can then be replaced over time while code can be used in other systems
  - The pattern: publisher-subscribe.
channel prioritization

carrier plug-ins

Camera
/camera

YARP
receiver

YARP
receiver

yarp connect /camera /receiver

yarp connect /65.52.88.202:5159 /receiver mjpeg

MJPG camera
http://65.52.88.202:5159

ROS
Node: /camera
Topic: /image

yarp connect /image@/camera /receiver

Camera.msg

== Slide 12 ==
custom, efficient, protocols
The **Port Monitor** is a plug-in that can be loaded by any connection point. It has access to in-coming and out-going data; usage:

- Add compression/de-compression algorithms
- Log (e.g. compute statistics or performance indicators)
- Sniff data, also bi-directional
- Avoid explicit man-in-the-middle components

```python
if (C1.certainty > 0.9):
    accept(C1)
C1 = filter(C1)
log(C1)
C1 = compress(c1)
If (C1):
    T1 = getTime()
```

- Filtering
- Logging
- Compression
- Monitoring delays, QoS
**Yarp Device:**
- A plugin which exposes the functionalities of a hardware device through a standardized Yarp C++ Interface.

**Yarp NWS:**
- A Network Wrapper Server (NWS) is a software component (plugin) attached to a physical device. It does not contain any logic. It just exposes the interface to the network.

**Yarp NWC:**
- A Network Wrapper Client (NWC) is a software component which implements the same interface of a real device, but instead of being connected to a physical hardware, it communicates with a Yarp NWS.
The code is well separated, and the functionality of each component is clear.

- Easy to maintain.
- Easy to extend.

- NWS can be used to make Yarp to communicate with different middlewares (which use different network/serialization protocols)
  - Yarp (yarp ports protocol)
  - ROS noetic (ros topics)
  - ROS2 humble (ros2 topics with DDS)
  - IsaacSDK Nvidia

- Multiple NWSs can be used simultaneously to expose the same plugin to multiple middlewares.
Due to a human programming error, the robot fell when transitioning from the driving task to the egress task (the foot throttle controller wasn’t turned off). This caused the robot to the fall and faceplant out of the car onto the asphalt.

Source: http://drc.mit.edu/
Rethink’s Robots Get Massive Software Upgrade, Rodney Brooks “So Excited” (IEEE Spectrum)

Slide credits: Michele Colledanchise
State charts vs. behavior trees (BT)
Behavior trees: a primer

**Sequence**
- Condition: \(\rightarrow\)
- Action:
  - Do1
  - Do2

**Fallback**
- Condition: \(?\)
- Action:
  - If fails
  - Do

**Parallel**
- Action:
  - Do
  - Do

== Slide 20 ==
Reactive behaviors: a simple example
Colledanchise, et al., Formalizing the Execution Context of Behavior Trees for Runtime Verification of Deliberative Policies, IROS 2021
Semantics of BT + Skills + Components

- BT, skills and components modelled as communicating transition systems - **asynchronous** execution (threads)
- Properties specified with SCOPE language (OTHELLO subset)
- Communication follows the **Query Pattern**
- Interfaces are specified using an **interface definition language**
- "Sniff" the messages passed across layers.
- Intercept message by a runtime monitor
- A runtime monitor detects differences between the expected behaviors and the actual one
A robotic museum guide

What’s needed:
- Dialog management
- Human-detection
- Self-localization
- Navigation

Cloud connectivity:
- Through 5G

How long:
- 200 meters, 20+ minutes (70 with questions)
- 110+ tours in two weeks
Hardware architecture
Software “tricks”

- **Port monitors** to implement data compression: images and LIDAR over 5G

- **Behavior trees** to implement the behavior coordination as shown earlier

- **Multiple middleware** systems: ROS for navigation, YARP to control the robot, Google APIs for speech, etc.

- **Flexible plug-ins** and remotization to handle distributed processing with controlled latency
Model-Based System Engineering (MBSE)

MBSE = MBD + System Engineering

- Complex Systems
  - Hierarchal components
  - Functional, logical, physical decompositions
  - Catch errors early, minimize rework

- Standardization
  - Data dictionaries for I/F’s
  - Ports and connections

- Design Optimization
  - Static analysis

- Effective Communication
  - Implementable descriptions
  - Requirements
From CAD design to realistic simulations

Simulink/Simscape
JAXsim

A scalable physics engine for robot learning implemented in pure Python with JAX.

Diego Ferigo, Silvio Traversaro, Daniele Pucci

ami-iit/jaxsim
Collaborative software & the robot apps
Community hub
Development steps

*GitHub Registry / Docker hub*

Web App containing available apps list

Docker Swarm

GitHub Registry / Docker hub
Automatic building & testing

- GitHub push
- trigger
- GitHub
- Self Hosted Building
- GitHub Action Docker Login
- GitHub Action docker-swarm-deployer
docker-compose-actions

Automatic building & testing

- push
- trigger
- GitHub Action Docker Login
- GitHub Action docker-swarm-deployer
docker-compose-actions