The YARP middleware

Lorenzo Natale
iCub Facility
Istituto Italiano di Tecnologia, Genova, Italy

Towards Humanoid Robots OS
Humanoids 2016 Workshop, Cancun, Mexico, 15th November, 2016
System Integration
Key issues

**Complexity:** distributed processing, heterogeneous systems, noise, real-time

**Asynchronous development**

Variability: various scenarios and platforms

Fast prototyping

Lack of standards

Fluctuation in hardware and algorithms, lots of open questions
Component driven software development

**Computation**

- What we are interested in

**Communication**

- Dependent on the hardware, network topology

**Configuration**

**Coordination**

- Application dependent

**Composition**
YARP approach

• Simplified form of **publish-subscribe**
  – **Observer pattern**: subscribers register their interest directly with publishers, which manage subscriptions and sends events
• Communication is **asynchronous** or synchronous
• Space decoupling
• Connections are **dynamic**
• Remote procedure calls for server-client type of communication

See also:
*Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects*, Stewart et al., 1997
Register /camera, 192.168.1.4:10001
Register /cmd, 192.168.1.4:10002
Query /camera ...
Query /cmd ...

Register /source, 192.168.1.3:10001
Register /pos, 192.168.1.3:10002
Query /camera ...
Query /cmd ...

connect /camera /source
connect /pos /cmd
connect /camera /source2 mcast
Which Middleware

- Robot Operating System (ROS)
- YARP
- OROCOS
- SmartSoft
- CORBA
- ICE
- OMG DDS
- Many others: OpenRDK, Mira...
## YARP/ROS comparison

<table>
<thead>
<tr>
<th><strong>YARP</strong></th>
<th><strong>ROS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run-time reconfiguration</strong> of connections</td>
<td>Strongly typed</td>
</tr>
<tr>
<td><strong>Pluggable protocols and devices</strong></td>
<td>Rich set of libraries and tools</td>
</tr>
<tr>
<td><strong>Multicast</strong> for efficient one-to-many</td>
<td><strong>Eco-system</strong>, very active community</td>
</tr>
<tr>
<td>communication</td>
<td><strong>Packet management</strong></td>
</tr>
<tr>
<td><strong>Multi-platform</strong></td>
<td>BSD license</td>
</tr>
<tr>
<td><strong>QoS, channel prioritization</strong></td>
<td>Ubuntu based</td>
</tr>
<tr>
<td><strong>LGPL/GPL</strong></td>
<td>Restricted set of protocols</td>
</tr>
<tr>
<td><strong>Smaller community</strong></td>
<td>All connections from a topic use the same protocol</td>
</tr>
<tr>
<td><strong>No packet management</strong></td>
<td></td>
</tr>
</tbody>
</table>
YARP main features

- Peer-to-peer, loosely coupled, communication
- Stable code base >10 years old
- Written in C++, bindings for python, Java, Matlab etc..
- Easy install with binaries on many OSes/distributions (Ubuntu, Debian, Windows, MacOs)
- Recently added: channel prioritization (including QoS)
- Custom protocols:
  - Built-in: tcp/udp/mcast
  - Plug-ins: ROS tcp, xml rpc, mjpg etc..
Simulators and datasets

- Using YARP without hardware: dataset player, simulators
- Available in sources and binary releases for Linux and Windows
- URDF models for iCub, Coman, Armar III, Walkman
- Gazebo ([https://github.com/robotology/gazebo-yarp-plugins](https://github.com/robotology/gazebo-yarp-plugins))
- Robotran (symbolic engine)
Repositories

116 members
160 contributors/year
11115 commits/year
243 total contributors

Source: https://www.openhub.net/p/robotology
Managing repositories & build system

• Projects are managed at the level of **individual repositories** and **large builds** (i.e. project repositories)
• Repositories are hosted on Github and on our own GitLab installation
• Single build system (YCM):
  – Agglomerate several projects in larger builds
  – Favor sharing of code (as opposed to binaries)
  – Built on top of CMake (~20 patches contributed to CMake)

https://github.com/robotology/ycm
lib-ace
libxml2-dev
libeigen3-dev
....

Walkman
download_and_compile(yarp)
download_and_compile(Gazebo)
download_and_compile(GazeboYARPPlugins)
download_and_compile(ComanSimulator)
download_and_compile(planner)
download_and_compile(valve)
....
<table>
<thead>
<tr>
<th>SubProjects</th>
<th>Project</th>
<th>Error</th>
<th>Configure</th>
<th>Warning</th>
<th>Pass</th>
<th>Error</th>
<th>Build</th>
<th>Warning</th>
<th>Pass</th>
<th>Not Run</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TinyXML</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>YARP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>kdl_codyco</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>kd_format_io</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iDynTree</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>paramHelp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>COMAN_shared</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>src_shared</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>simple_homeing</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>valve</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>flat_walk</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>hand_control</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>DStabilizer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>coman_yarp_apps</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>hri_geom</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>pykdl_utils</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>sot_velkcon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>UIComarRosPlg</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>basic_example</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>comar_publisher</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Testing

• Unit testing for the YARP middleware on compile farm and github (travis)
• Pull Requests are peer-reviewed using CodeReviewHub
• Robot software is tested using Robotic Testing Framework
  – Specifications
  – Individual components
  – Configuration
  – Bugs
Managing and building
Managing and building
Interfaces

- Robot abstraction layer: interfaces to motors and sensors minimize the impact of changes in the hardware
- Custom interfaces and data types (Thrift IDL)

```thrift
point3d.thirft
struct Point3D {
  1: i32 x;
  2: i32 y;
  3: i32 z;
}

adder.thirft
service Adder {
  /** ... */
  i32 get_answer();
  /** ... */
  bool set_answer(1:i32 val)
  /** ... */
  i32 add (1:i32 x);
}
```
Robot Interfaces

Loops on board

... 
... 
read encoders
read IMU
read FT
... 
get image
... 
set position
Robot Interfaces

External loops
...
... read encoders
... read IMU
... read FT
... get image
... set position

Network
Robot Interface
Network Stub

Loops on board
...
... read encoders
... read IMU
... read FT
... get image
... set position
YARP plugins

• YARP includes a plugin system for **drivers** and **protocols** (carriers)
• Interchangeable carriers allow:
  – interfacing existing software with ports (without bridges)
  – change significantly port behavior
• Examples: ROS, mjpeg, xml rpc, ..., port monitor
Carrier plugins

YARP Camera
/camera

YARP receiver

yarp connect /camera /receiver

MJPG camera
http://65.52.88.202:5159

YARP receiver

yarp connect /65.52.88.202:5159 /receiver mjpeg

ROS Camera
Node: /camera
Topic: /image

YARP receiver

yarp connect /image@/camera /receiver

Camera.msg
More on YARP-ROS (1)

- YARP protocols for rostcp and xmlrpc
- Compatibility with roscore
- YARP can interpret ROS messages, statically or dynamically
- Extended YARP’s API (nodes, publishers, subscribers)

```cpp
#include "Pose.h"
#include "Twist.h"
/* create ROS Node /controller */
yarp::os::Node node("/controller");
/* create a subscriber for Pose.msg */
yarp::os::Subscriber<Pose> pose;
/* subscribe to /turtle1/pose */
pose.topic("/turtle1/pose");
/* create a publisher for Twist.msg */
yarp::os::Publisher<Twist> cmd;
/* publish to /turtle1/cmd_vel */
msg::Twist msg;
/* publish to /turtle1/cmd_vel */
cmd.topic("/turtle1/cmd_vel");
/* read a new value from the topic */
pose.read(p);
/* publish the command */
cmd.write(t);```

- Include Pose.h
- Include Twist.h
- Create a ROS Node /controller
- Create a subscriber for Pose.msg
- Subscribe to /turtle1/pose
- Create a publisher for Twist.msg
- Publish to /turtle1/cmd_vel
- Read a new value from the topic
- Publish the command
YARP machine

C:\yarp read /turtle1/pose@/reader
yarp: Receiving input from /turtle1/pose to /turtle1/pose-@/reader
5.544445 5.544445 0.0 0.0 0.0
5.544445 5.544445 0.0 0.0 0.0
...

ROS machine

$ rosrun turtlesim turtlesim_node
[ INFO] [1444722896.501281004]: Starting turtlesim with node name /turtle1/pose

[type] BEGIN turtlesim/Pose
[type]   float32 x
[type]   float32 y
[type]   float32 theta
[type]   float32 linear_velocity
[type]   float32 angular_velocity
[type] END turtlesim/Pose

ROS+YARP machine

yarpidl_rosmsg -name /typ@yarpidl
ROS gmapping, RVIZ, Gazebo
Port monitor

Paikan, A. et al, Data Flow Port's Monitoring and Arbitration, JOSER 2014
if (C1.certainty > 0.9) accept(C1)
else accept(C2)

if (check(C1))
dispatch(event)

C1=filter(C1)
dispatch(C1)

If (C1)
T1=getTime()
Why Channel Prioritization (QoS)
Determinism is affected by:

- **Thread scheduling** (CPU usage)
- **Packet conflicts** (network usage)
Approach: improve determinism by increasing thread priorities and reducing network bottlenecks using QoS

Paikan et al., A Best-Effort Approach for Run-Time Channel Prioritization in Real-Time Robotic Application IROS 2015

> prop sched policy 1 priority 30
> prop set qos priority HIGH
HOST1

20%/70% load

With priority (20%)

Standard YARP (20%)

With priority (70%)

Standard YARP (70%)

With priority (70%)
Comparison with DDS

Average round-trip time

<table>
<thead>
<tr>
<th>Load</th>
<th>YARP Pkts l/s</th>
<th>RTI Pkts l/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>326</td>
<td>240</td>
</tr>
<tr>
<td>20%</td>
<td>393</td>
<td>398</td>
</tr>
<tr>
<td>40%</td>
<td>696</td>
<td>577</td>
</tr>
<tr>
<td>80%</td>
<td>2074</td>
<td>2400</td>
</tr>
</tbody>
</table>

Standard Deviation of round-trip time

<table>
<thead>
<tr>
<th>Load</th>
<th>YARP Pkts l/s</th>
<th>RTI Pkts l/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>44.1247</td>
<td>148.9</td>
</tr>
<tr>
<td>20%</td>
<td>161.361</td>
<td>261.4</td>
</tr>
<tr>
<td>40%</td>
<td>445.729</td>
<td>492.5</td>
</tr>
<tr>
<td>80%</td>
<td>583.3</td>
<td>699.9</td>
</tr>
</tbody>
</table>
Comparison with DDS

<table>
<thead>
<tr>
<th>RTI QOS (DataWriter)</th>
<th>YARP QOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>history: KEEP_ALL_HISTORY_QOS</td>
<td></td>
</tr>
<tr>
<td>Reliability: DURATION_INFINITE_SEC</td>
<td></td>
</tr>
<tr>
<td>Transport_priority : 36</td>
<td></td>
</tr>
<tr>
<td>Thread.mask: DDS_THREAD_SETTINGS_REALTIME_PRIORITY; thread.priority = 30;</td>
<td></td>
</tr>
<tr>
<td>threadPriority: 30</td>
<td></td>
</tr>
<tr>
<td>packetPriority: HIGH (36)</td>
<td></td>
</tr>
<tr>
<td>threadPolicy: SCHED_FIFO</td>
<td></td>
</tr>
</tbody>
</table>

Average round-trip time (with QoS)

Network arbitrary load using TCP protocol

Standard Deviation of round-trip time (with Qos)

Network arbitrary load using TCP protocol
An application
Acknowledgements

Ali Paikan (YARP, port monitor, channel prioritization)
Daniele Domenichelli (YARP, ROS-YARP)
Alberto Cardellino (YARP, robot interface, ROS-YARP)
Marco Randazzo (robot interface, firmware)
Andrea Ruzzenenti (simulator, slam)
Marco Accame (firmware)
Valentina Gaggero (firmware)
Alessandro Scalzo (firmware)

Ugo Pattacini
Vadim Tikhanoff
Silvio Traversaro
Francesco Romano
Francesco Nori
Giorgio Metta
Thank you