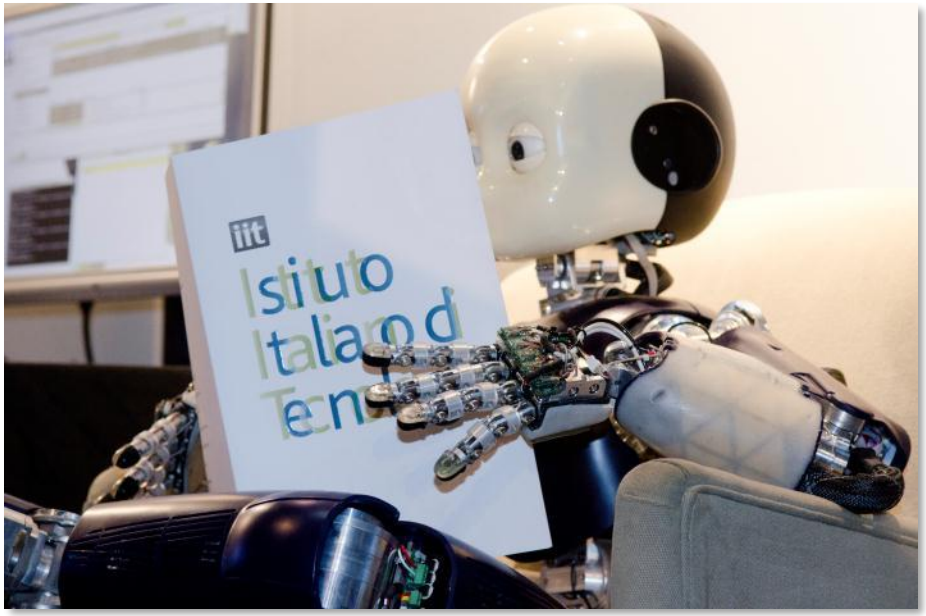
ISTITUTO ITALIANO
DI TECNOLOGIA

Middleware and Robotic Software Programming

Lorenzo Natale

iCub Facility
Istituto Italiano di Tecnologia, Genova



Pisa, May 2016

Schedule & content

- 10/5 10.30-12.30 (Aula Riunioni, largo Lazzarino)
- 13/5 15-17 (Aula Riunioni, piano terra via Caruso)
- 18/5 15-18 (Aula Riunioni, piano terra via Caruso)
- 20/5 15-18 (Aula Riunioni, piano terra via Caruso)

Topics:

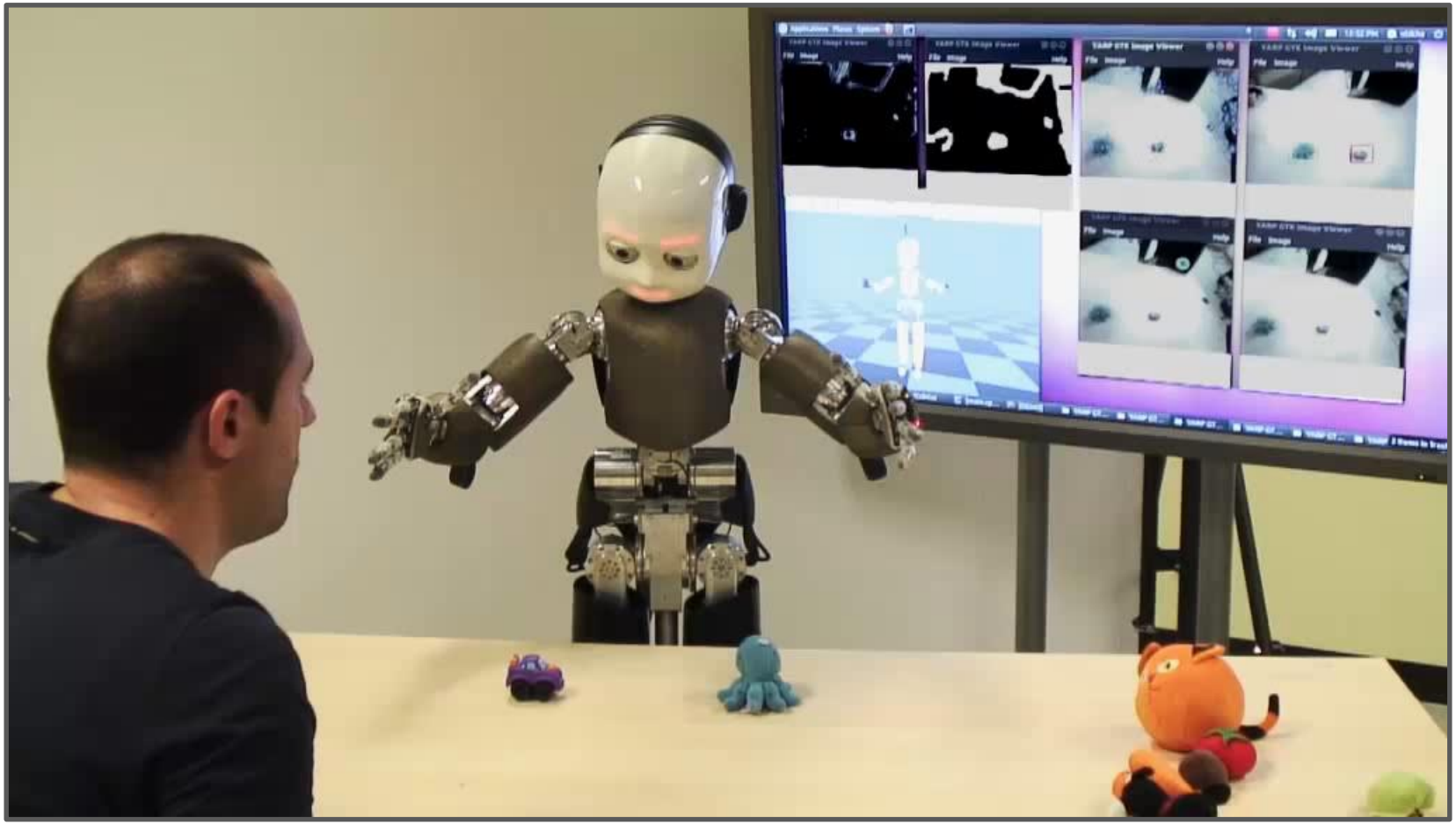
- Robotic middleware and component based programming, introduction to YARP
- Ports, Modules and threading
- Interface Definition Languages in YARP, interoperability with ROS
- Interfacing with OpenCV, image processing and motor control

References

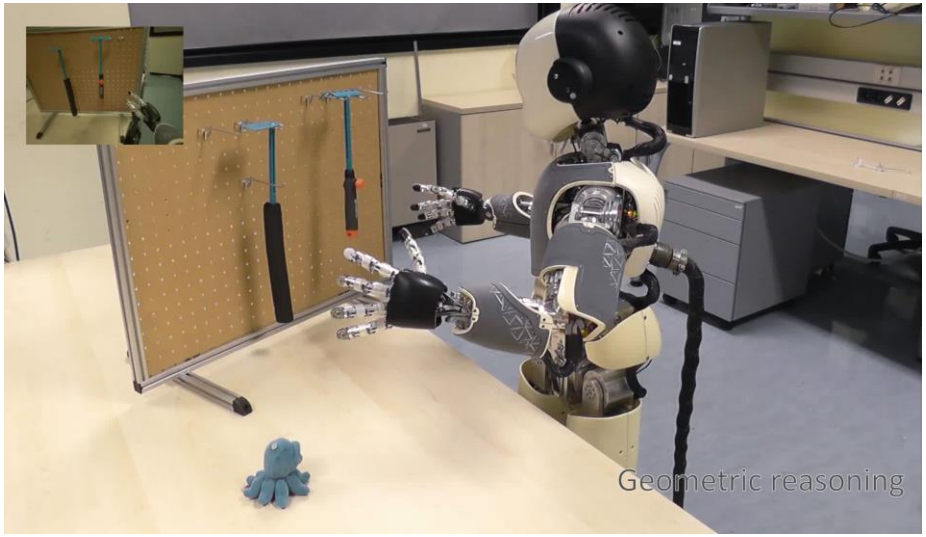
- YARP: www.yarp.it
  - Documentation
  - Papers (see also references in this presentation)
- iCub: http://wiki.icub.org/wiki/Manual
- CMake:
  - www.cmake.org → documentation & wiki
  - Mastering CMake, by Kitware Inc.
- C++:
  - Thinking in C++, Bruce Eckel
  - The C++ programming language, Bijarne Stroustrup
  - Anything else

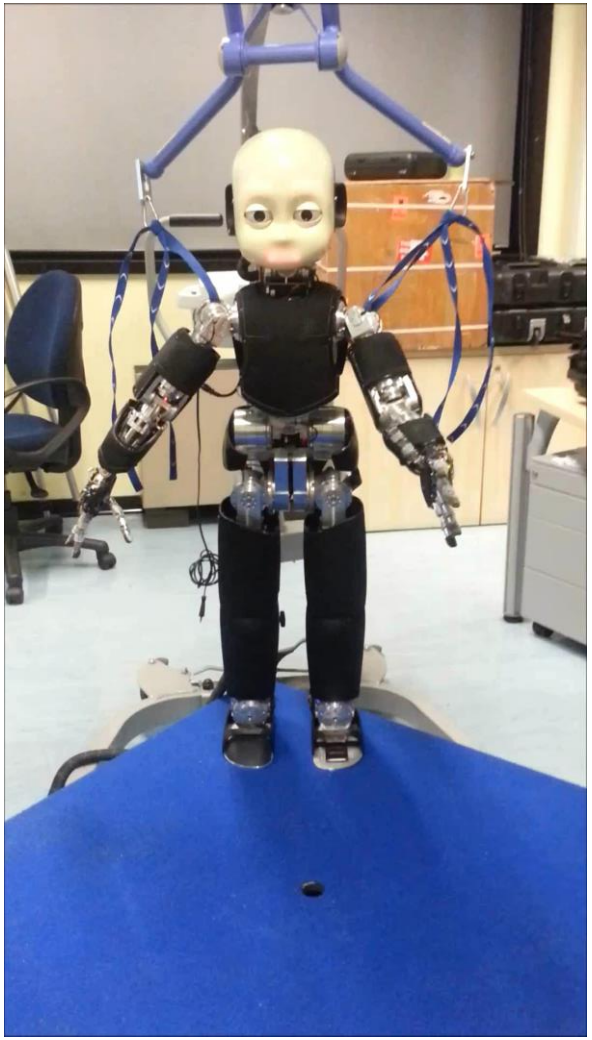
Motivations

Complex behaviors

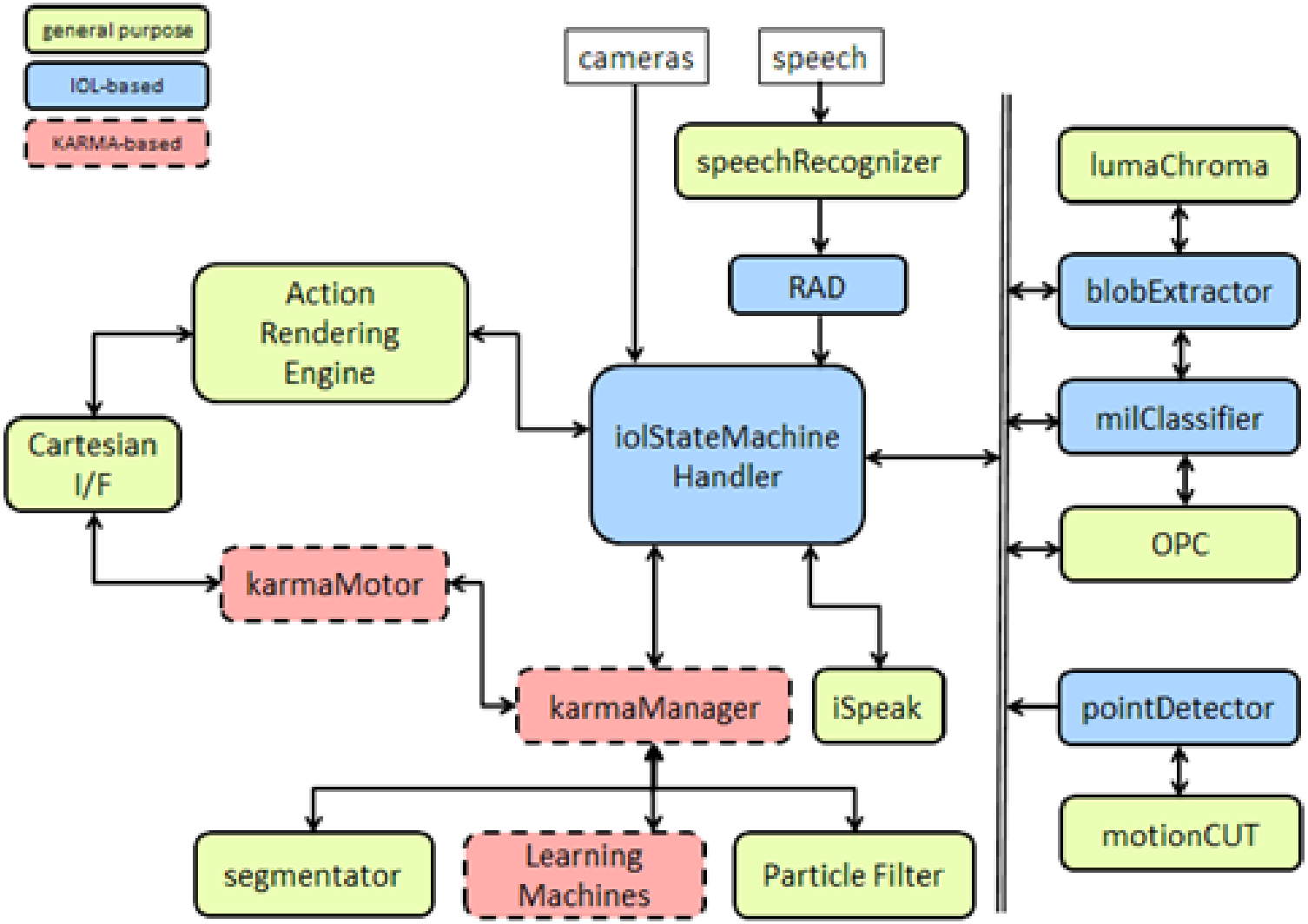


Integration of software components



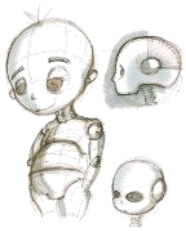


Complex behaviors



Key issues

- Inherent complexity:
  - Distributed processing
  - Heterogeneous systems
  - Real-world: must deal with uncertainty, noise
  - Real-time
- Asynchronous development
- Variability: various scenarios and platforms
- Fast prototyping
- Lack of standards
- Fluctuation in hardware and algorithms, lots of open questions








Software architecture

- Major cost in software development is debugging, recycling code is key
- Divide and conquer
- Modularity
- Factor out platform specificities
  - Hardware Abstraction Layer
  - Communication Abstraction
  - Operating system
  - Parameters
  - Computing infrastructure

Separation of concerns (5C)

Goal: separate software components

- Computation ← What we are interested in
- Communication ← Dependent on the hardware, network topology
- Configuration
- Coordination
- Composition

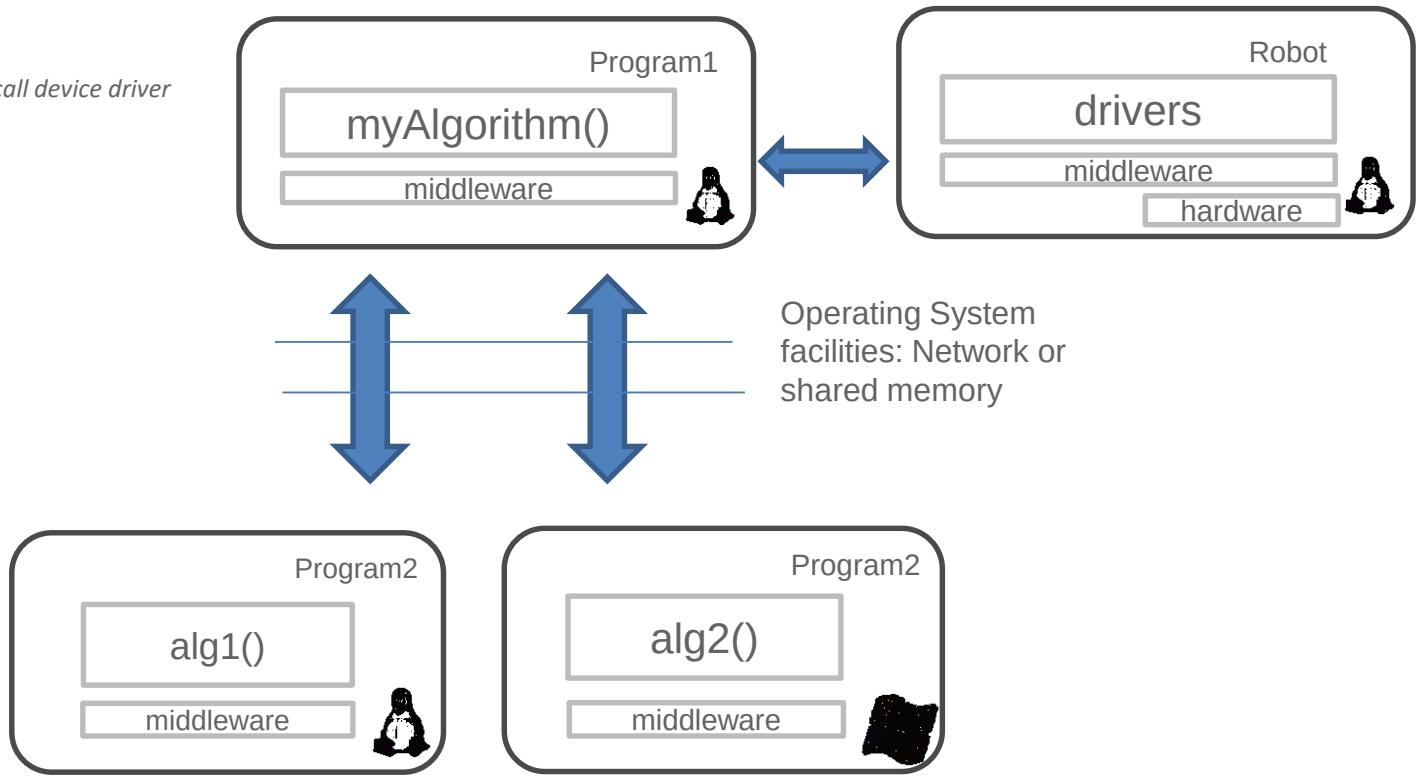
Application dependent

Component driven software development

```
output myAlgorithm(input)
{
   ...
   data=readSensor()  //call device driver
   out = call alg1(data)
   …
   out = call alg2(data)

   controlMotors(out)  //call device driver
   …
}

output alg1(input)
{code}

output alg2(input)
{code}
```

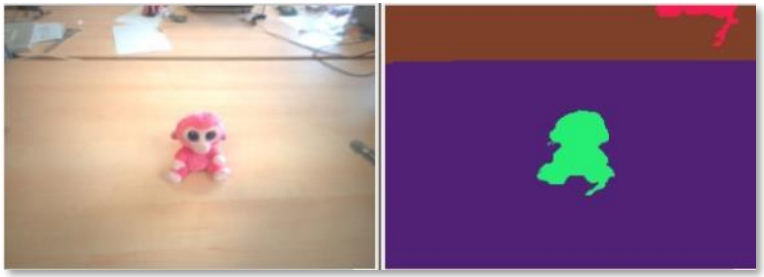
Component driven software development

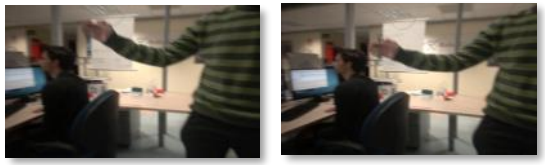
```
output myAlgorithm(input)
{
  data=readSensor()  //call device driver
  out = query(alg1, data)
  …
  out = query(alg2, data)

  controlMotors(out)  //call device driver
  …
}

output alg1(input)
{code}

output alg2(input)
{code}
```

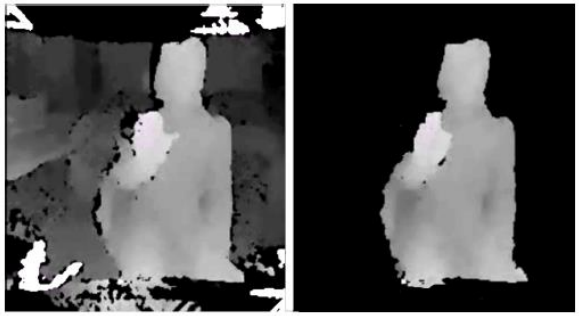


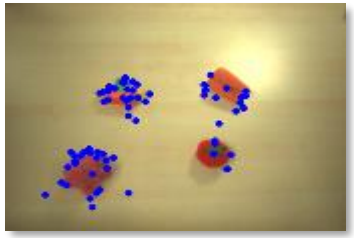
Components:
some examples from the iCub repository

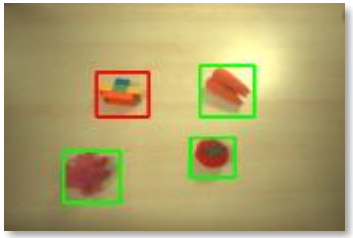
- Algorithms for motion computation and egomotion compensation
- Machine learning for vision
- Disparity map
- Action recognition
- Segmentation
- See https://github.com/robotology









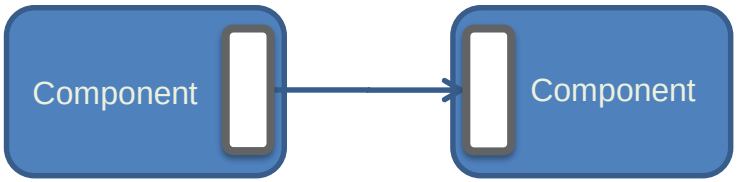


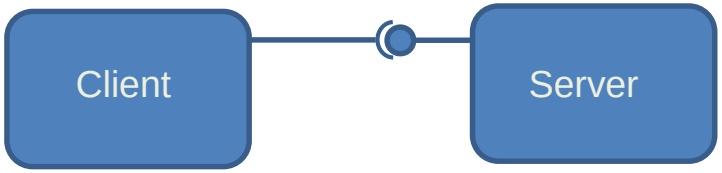
Component driven development

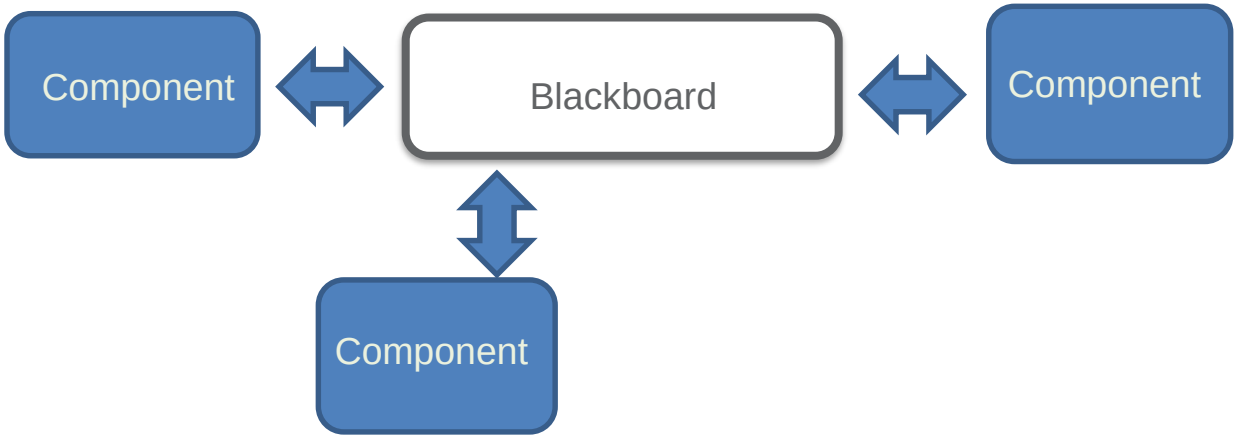
- Modular software: simple structure, data encapsulation, interface
- Reconfigurable components
- Reduced coupling: interaction between components happens through pre-defined standards (no direct inclusion of header files)
- Language independent

Information sharing model

- Data ports
- Services
- Data centric (blackboard)







Communication paradigm

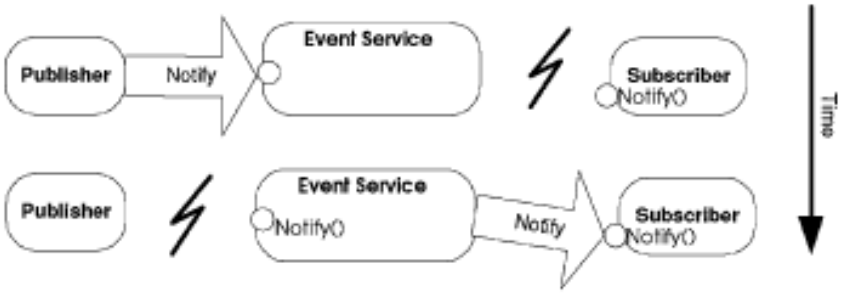
- Publish/Subscribe
  - Space, time, synchronization decoupling
- Remote Procedure Calls (RPC)
  - Remote invocation of an object
  - Synchronous nature (although variant exists)

Publish/Subscribe

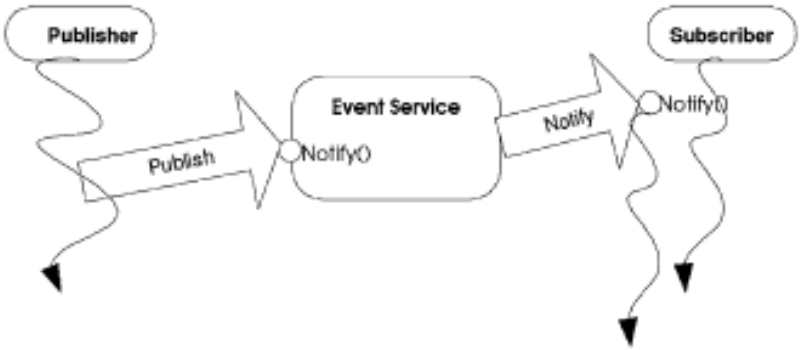
- **Space decoupling**: the interaction parties do not need to know each other; publisher publish events through an event service and the subscribers get these events indirectly through the event service
- **Time decoupling**: interaction parties do not need to be actively participating in the interaction at the same time, publisher might publish events while subscribers is disconnected and subscribers might get notified of an event while the original publisher is disconnected
- **Synchronization decoupling**: publishers are not blocked while producing events and subscribers can get asynchronously notified (through callback) of the occurrence of the event while performing concurrent activity

Eugster, et al., The Many Faces of Publish/Subscribe, 2003.

Space decoupling



Time decoupling



Synchronization decoupling

Eugster, et al., The Many Faces of Publish/Subscribe, 2003.

Which Middleware

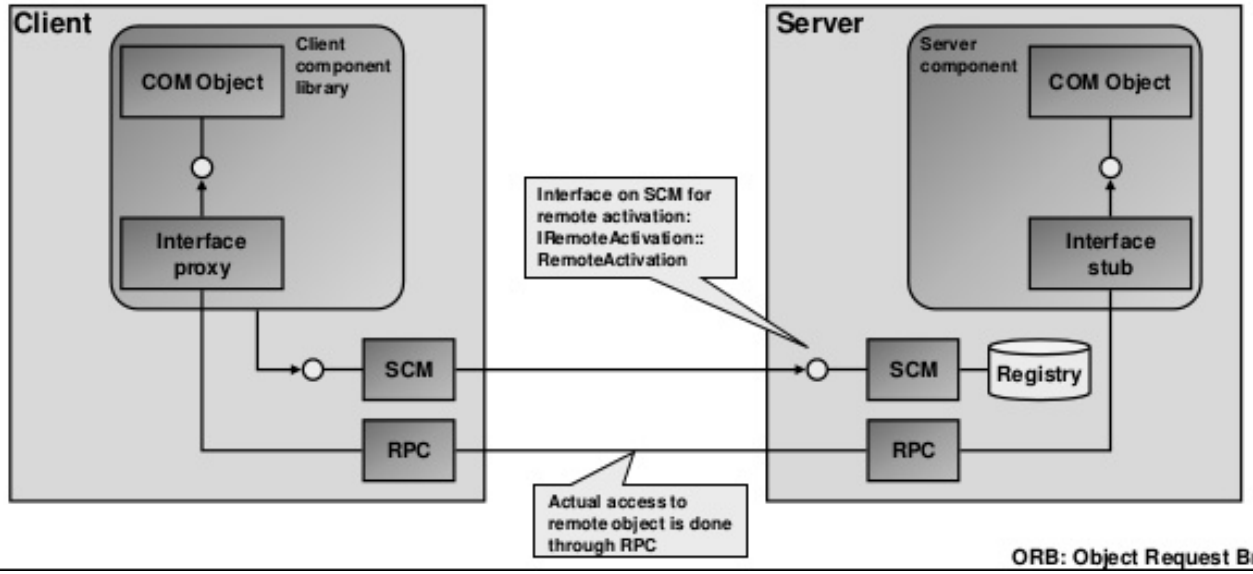
- Robot Operating System
- OROCOS
- YARP
- SmartSoft
- CORBA
- ICE
- OMG DDS
- Many others: OpenRDK, Mira…

ICE/CORBA/COM-DCOM

- Remote object invocation
- An object-oriented specification language
- Synchronous/asynchronous
- May have deployment tool, versioning etc..

8. DCOM Architecture

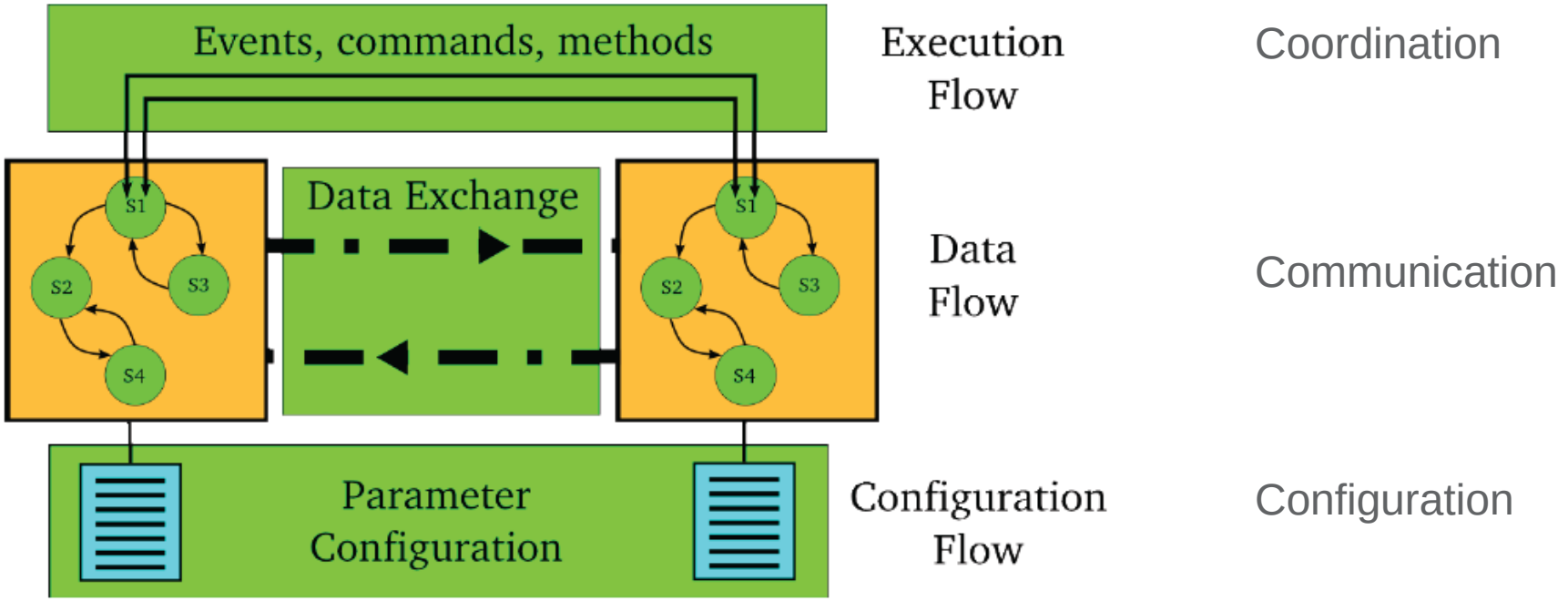
Client proxy:	Proxy object on client side for accessing the server object.
Stub:	Server interface stub that complements the client interface proxy.
Registry:	Contains a list of mappings of class / object GUID to implementation library.
SCM:	Service Control Manager (RPCSS.exe) which consults registry and creates / instantiates a new server object based on the GUID (comparable to ORB in CORBA). The SCM hides the registry from (D)COM.

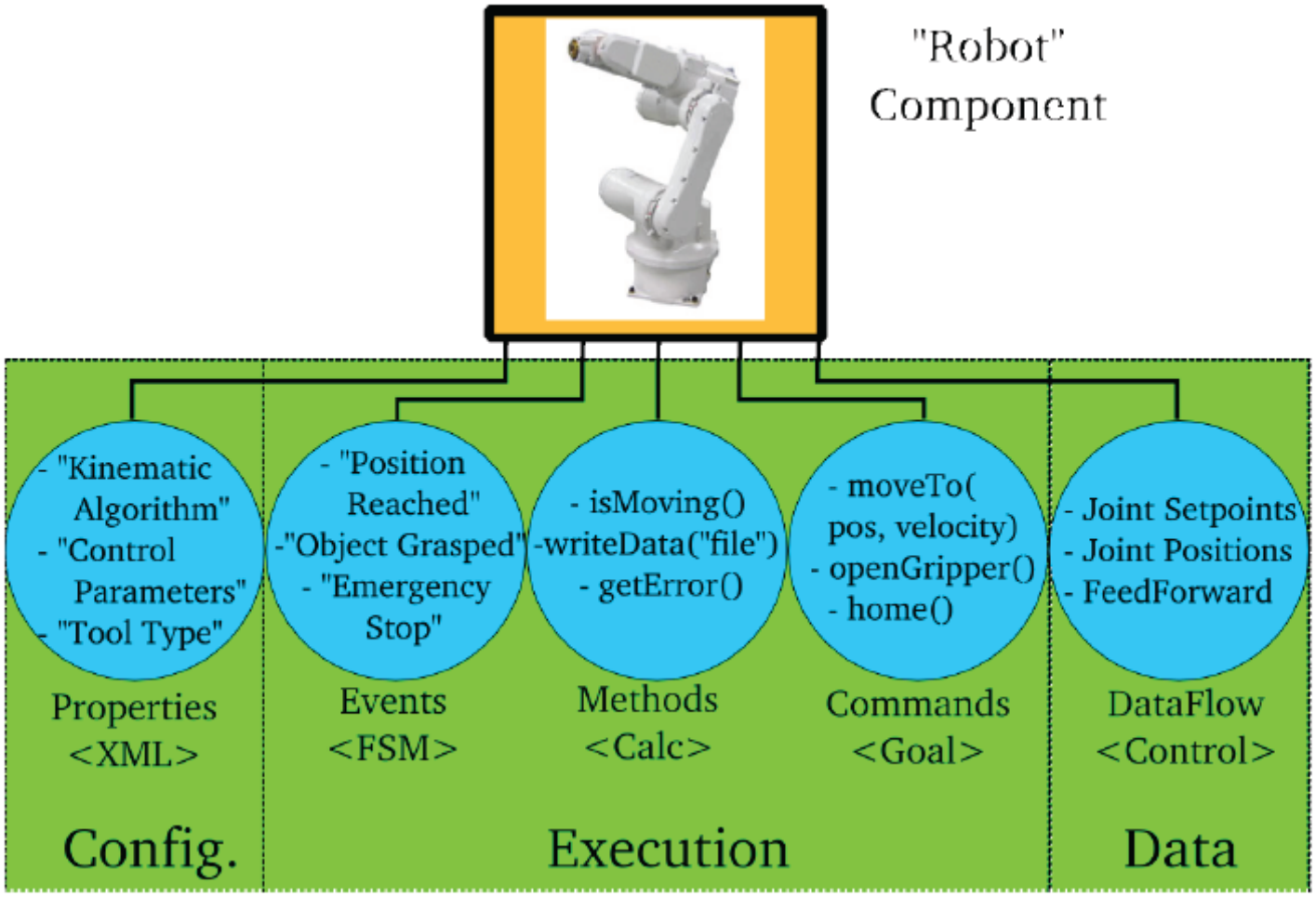


ORB: Object Request Broker

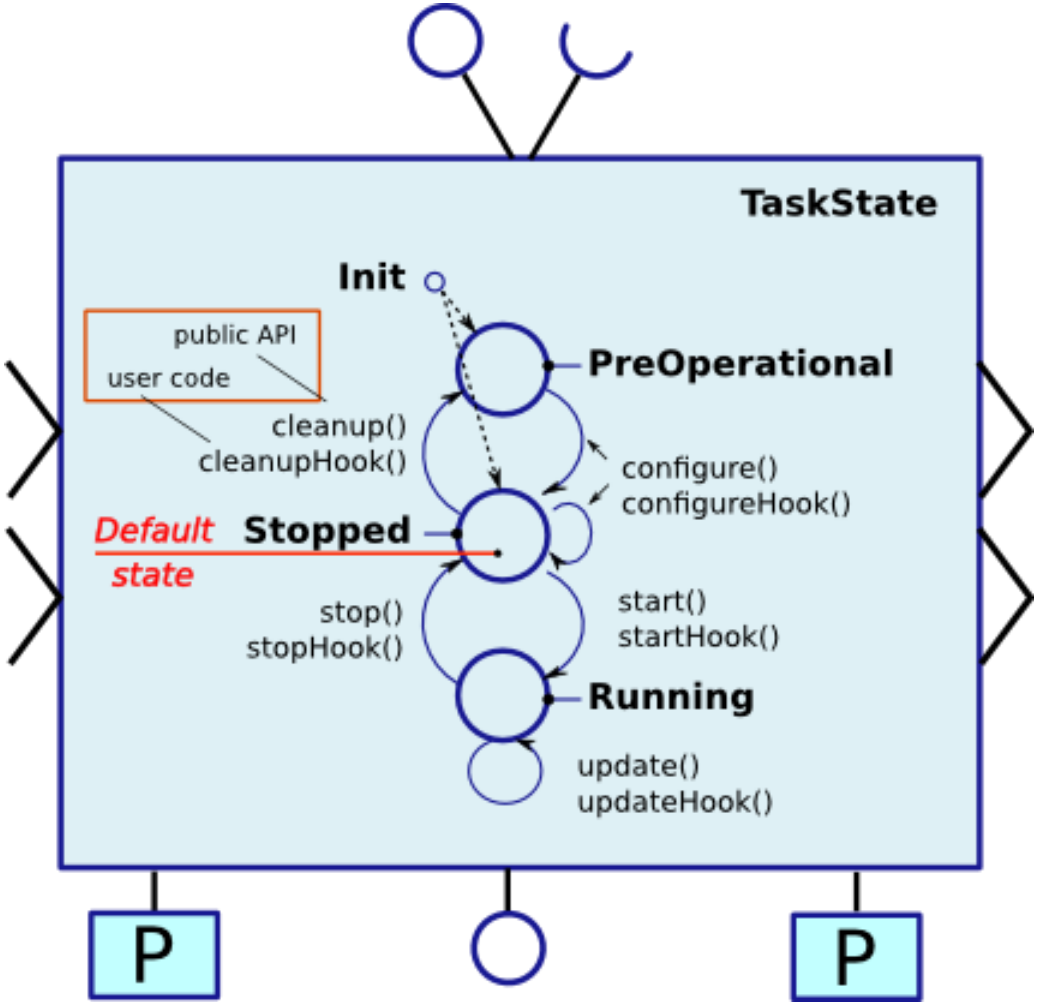
OROCOS:
Open Robot Control Software

- Kinematics and Dynamics Library and Bayesian Filtering Library
- Real-Time Toolkit: it provides the infrastructure and the functionalities to build *real-time* robotics applications in C++
- OROCOS Component Library: ready to use components




"Robot"
Component
- "Kinematic Algorithm"
- "Control Parameters"
- "Tool Type"
Properties
<XML>
- "Position Reached"
-"Object Grasped"
- "Emergency Stop"
Events
<FSM>
- isMoving()
-writeData("file")
- getError()
Methods
<Calc>
- moveTo(pos, velocity)
- openGripper()
- home()
Commands
<Goal>
- Joint Setpoints
- Joint Positions
- FeedForward
DataFlow
<Control>
Config.
Execution
Data

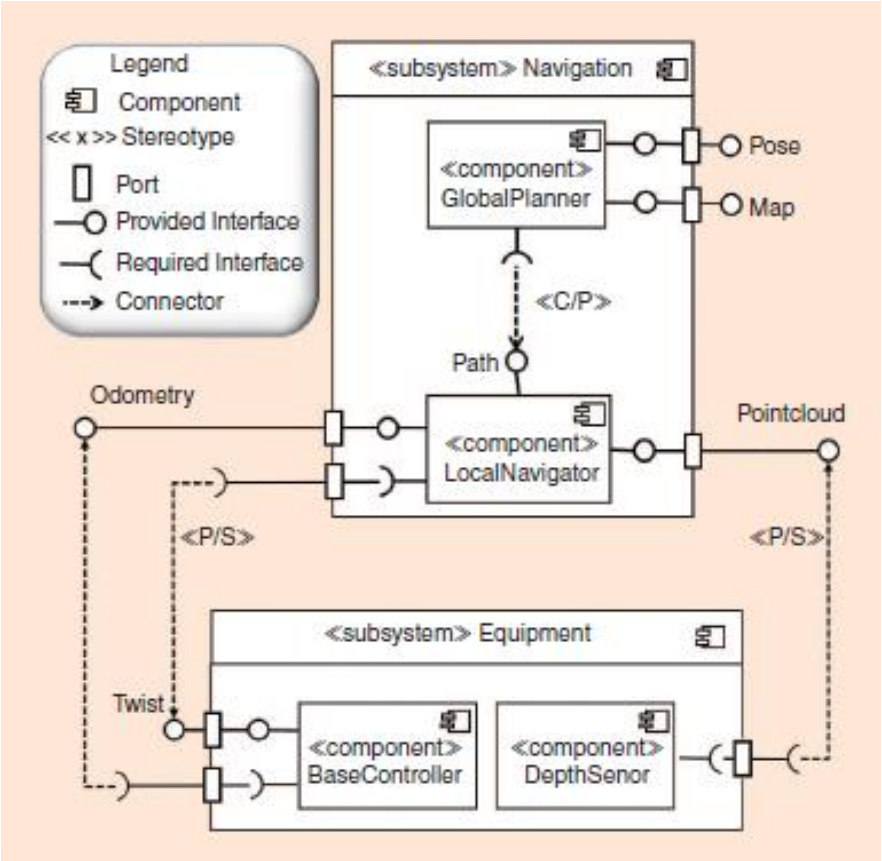
Component lifecycle



Model Driven Software Engineering

- Models can make a particular system or phenomenon easier to understand, quantify, visualize simulate, or predict
- Models are used in many fields of engineering, either graphical (architecture) or textual (equations of physics)
- They are created using modeling languages (Domain specific languages, or DSLs)
- Software engineering can greatly benefit from the use of models because software and models have the same nature, and it is easy to automate the process of software generation and analysis
- Example: SmartSoft MDE toolchain

From: D. Brugali, Model-Driven Software Engineering in Robotics, IEEE Robotics and automation magazine, Sept. 2015

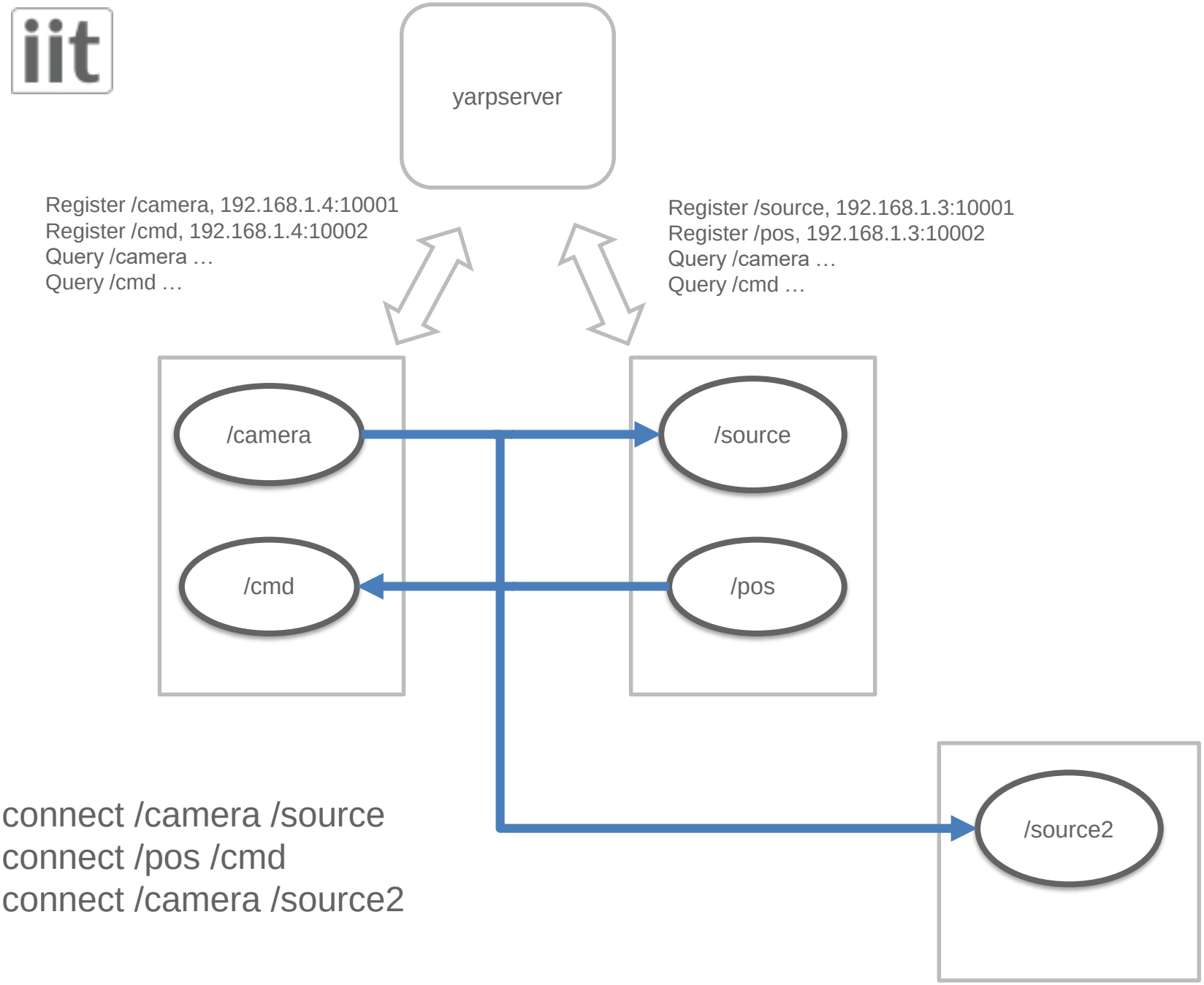


- <<component>>: elemental components
- <<subsystem>>: composite components
- Each component is characterized by a set of ports, interaction point with other components
- Provide and requested interfaces
- Connectors: Caller/Provider, Publisher/Subscriber

- From an architectural model it is possible to generate platform specific code templates (that implement the specified architecture and incude calls to the target framework, i.e. the middleware)
- Code templates is complemented with code provided by the developer (specific robot functionalities) to produce executable code
- Challenges: hardware requirements, real-time constraints, embedded and distributed systems, many application scenarios, dynamic interaction with the environment
- DSL for robotics should include: timing constraints, communication infrastructure, allocation of threads and processes to difference processors

YARP/ROS

- Components are executables which communicate through well-defined named objects called Ports (YARP), Topics (ROS)
- A central server (yarpserver/roscore) keeps tracks of names and allows Ports/Topics to be reachable
- Communication is peer-to-peer between Ports/Topics


yarpserver
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 …
/camera
/cmd
/source
/pos
/source2
connect /camera /source
connect /pos /cmd
connect /camera /source2

Interoperability

- No best platform:
  - Application (service robotics, industrial robotics, multi-robot..)
  - Focus (low-level real-time, AI, ML…)
  - Community
  - Hardware
- The ability of heterogeneous systems to suitably exchange information, using common protocols and abstractions

Data Distribution Service

- Omg standard
- Data (as opposed to message) centric
- Share data even between time-decoupled publishers and consumers
- Interest-based filter on content, age and/or lifecycle
- DDS dynamically discovers publishers and subscribers (no central server naming server), determines data they share and Quality of Service (QoS)
- For example, if a subscriber requires an update every 10ms and its matched publisher does not deliver, the system declares an error, enabling remedial action
- QoS: urgency, importance, reliability, persistence, and liveliness
- It is built on two standards: real-time wire protocol (RTPS) and C++ API
- Several implementations exists: proprietary as well as open source (opensplice, opendds, RTI connext)
- ROS 2.0 is going to be based on DDS

Libraries & Tools

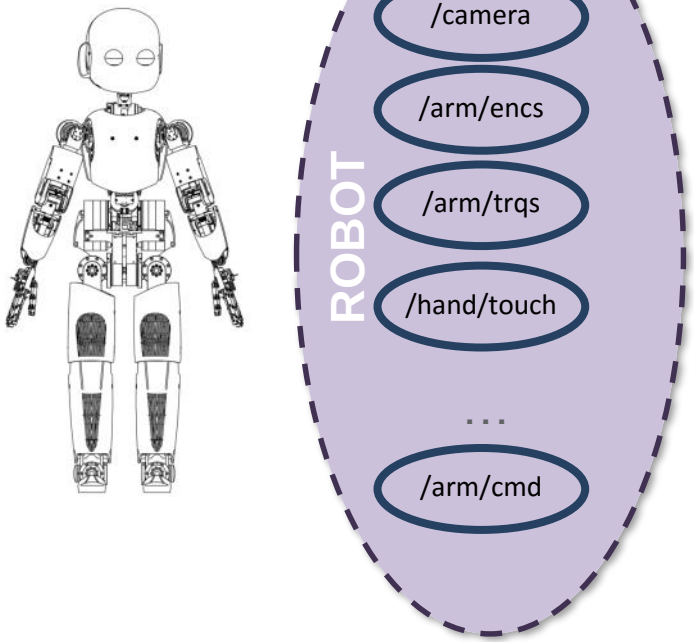
- Computer Vision:
  - OpenCV
  - Point Cloud Library
- Machine learning
  - GURLS
  - libSVM
- KDL
- Eigen, GSL
- Simulators:
  - Gazebo
  - MORSE
- ….

YARP approach

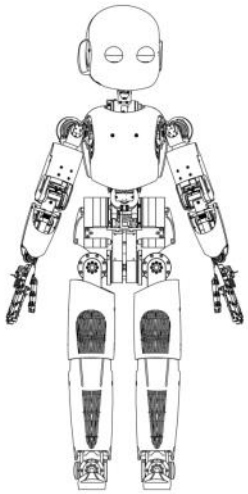
- Limited form of publish-subscribe
- *Observer patter*: subscribers register their interest directly with publishers, which mange subscriptions and sends events
- Publishers notify subscribers synchronously or asynchronously
- Space decoupling (from user perspective)
- Time coupling
- Dynamic (re)configuration

See also:
YARP: Yet Another Robot Platform, G. Metta, P. Fitzpatrick, L. Natale, 2006
Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects, Stewart et al., 1997

iCub software architecture



iCub software architecture



ROBOT

/camera

/arm/encs

/arm/trqs

/hand/touch

...

/arm/cmd

Object det.

...

...

Reaching

...

...

...

...

...

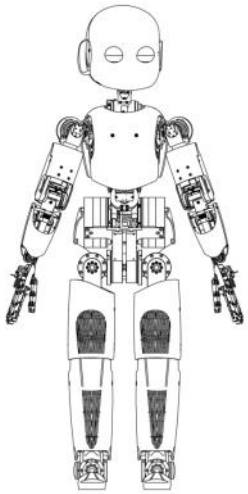
Grasping

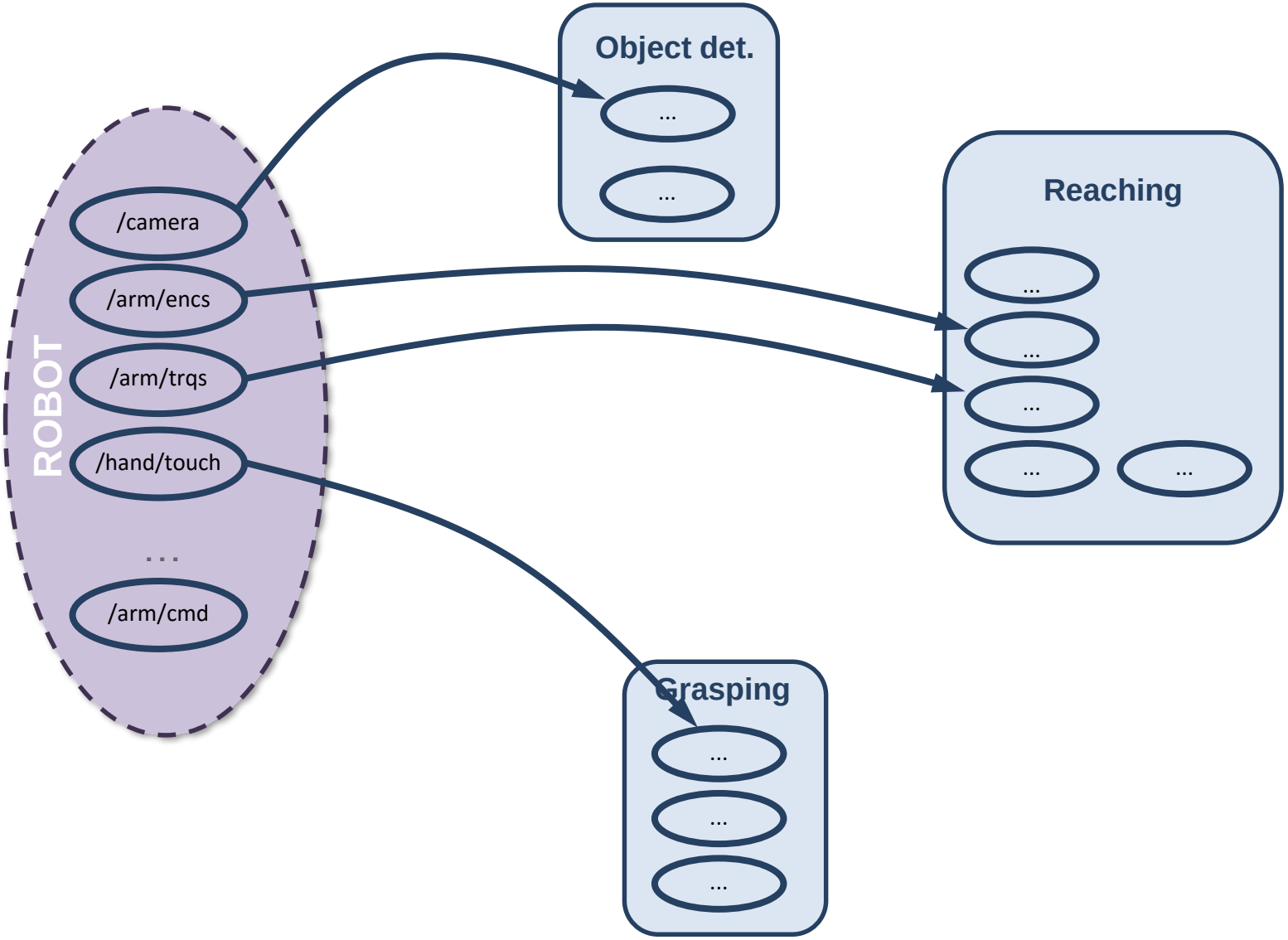
...

...

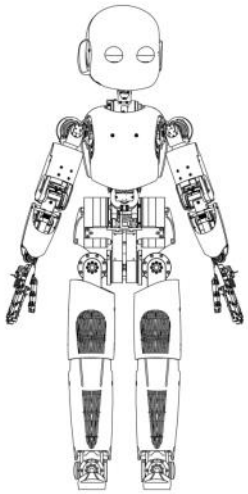
...

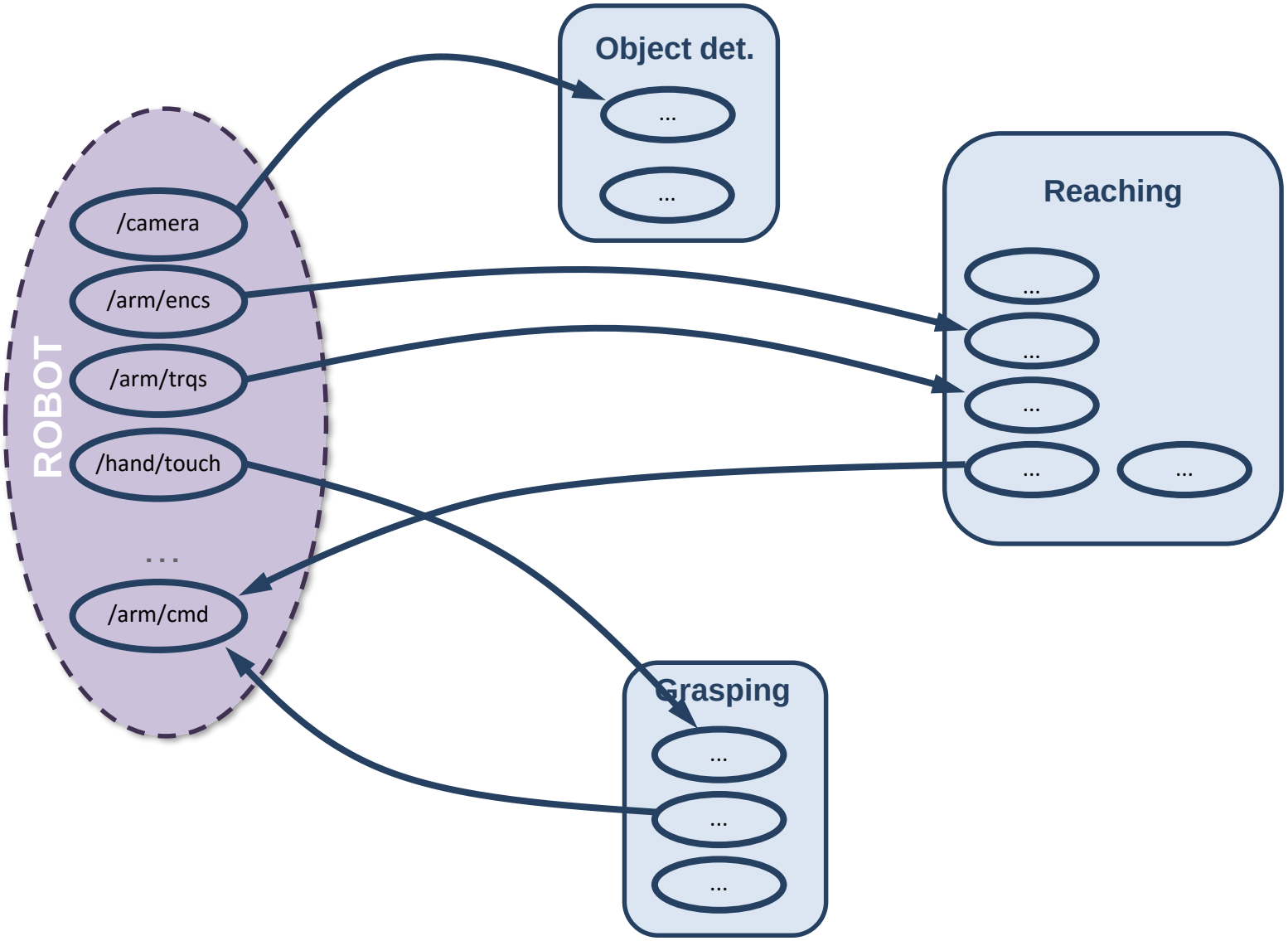
iCub software architecture



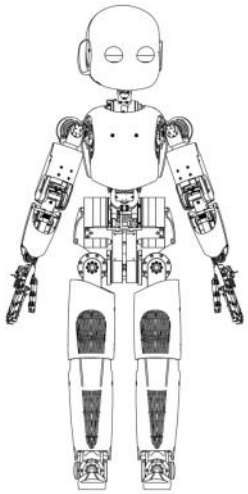


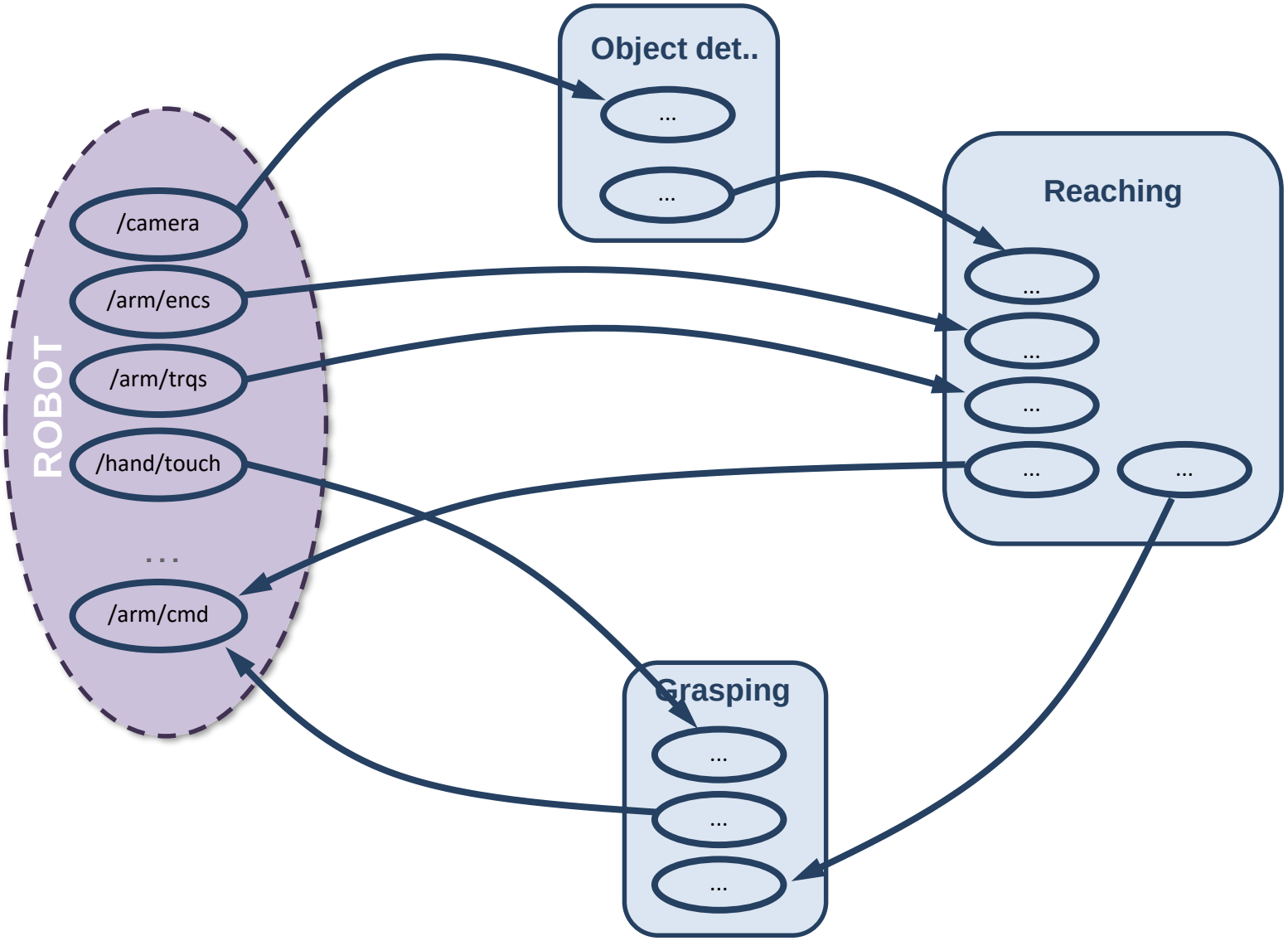
iCub software architecture



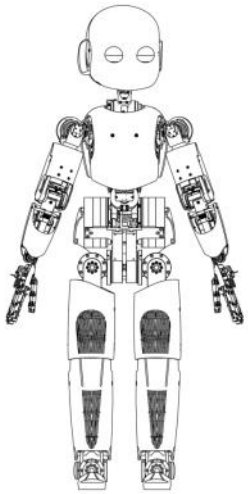


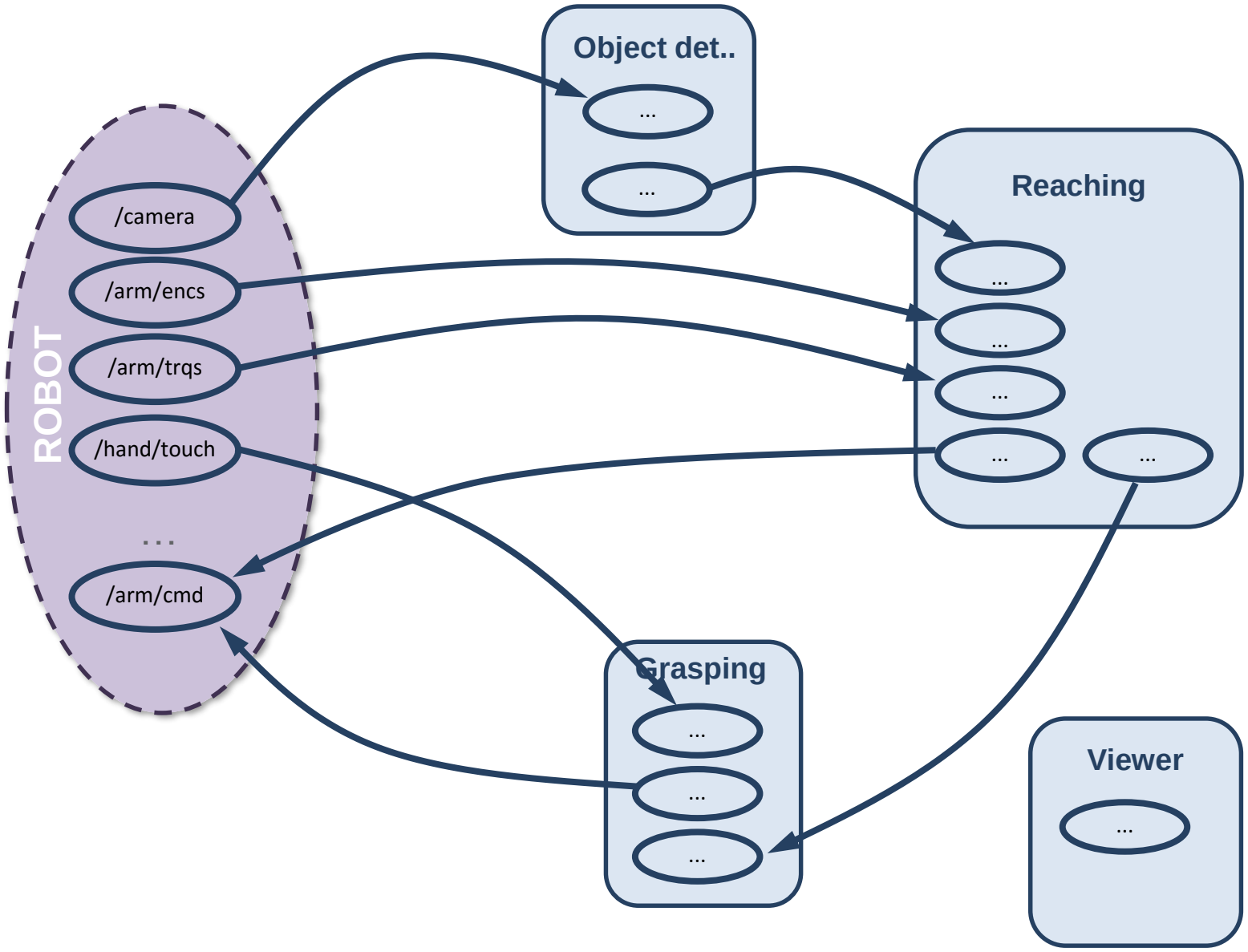
iCub software architecture



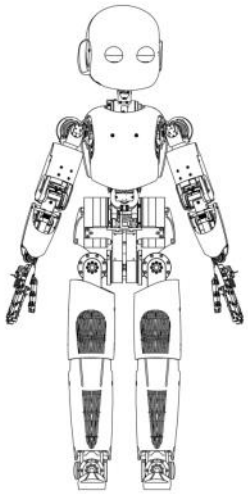


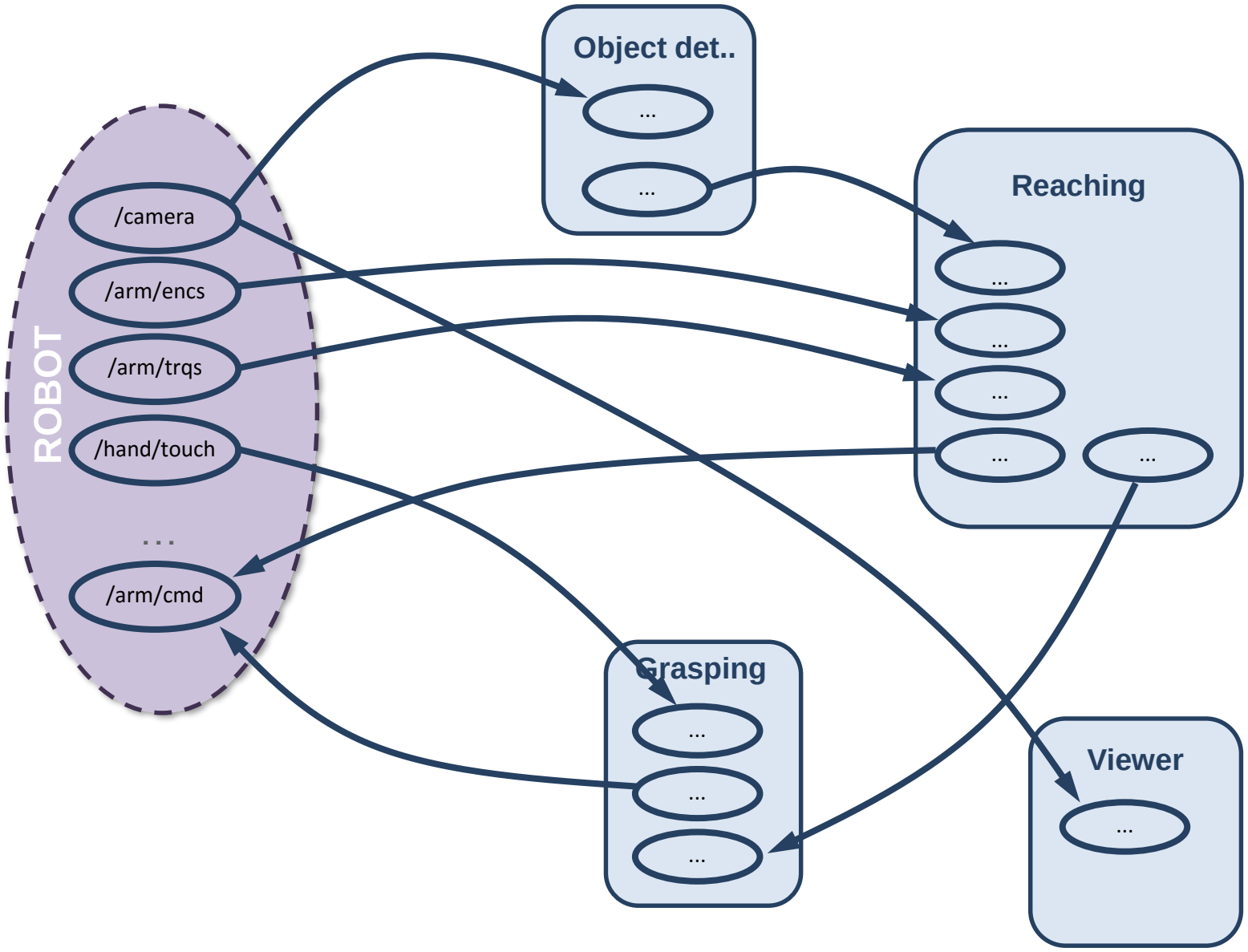
iCub software architecture



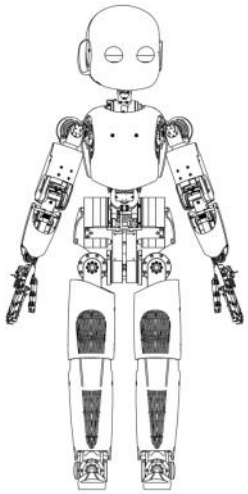


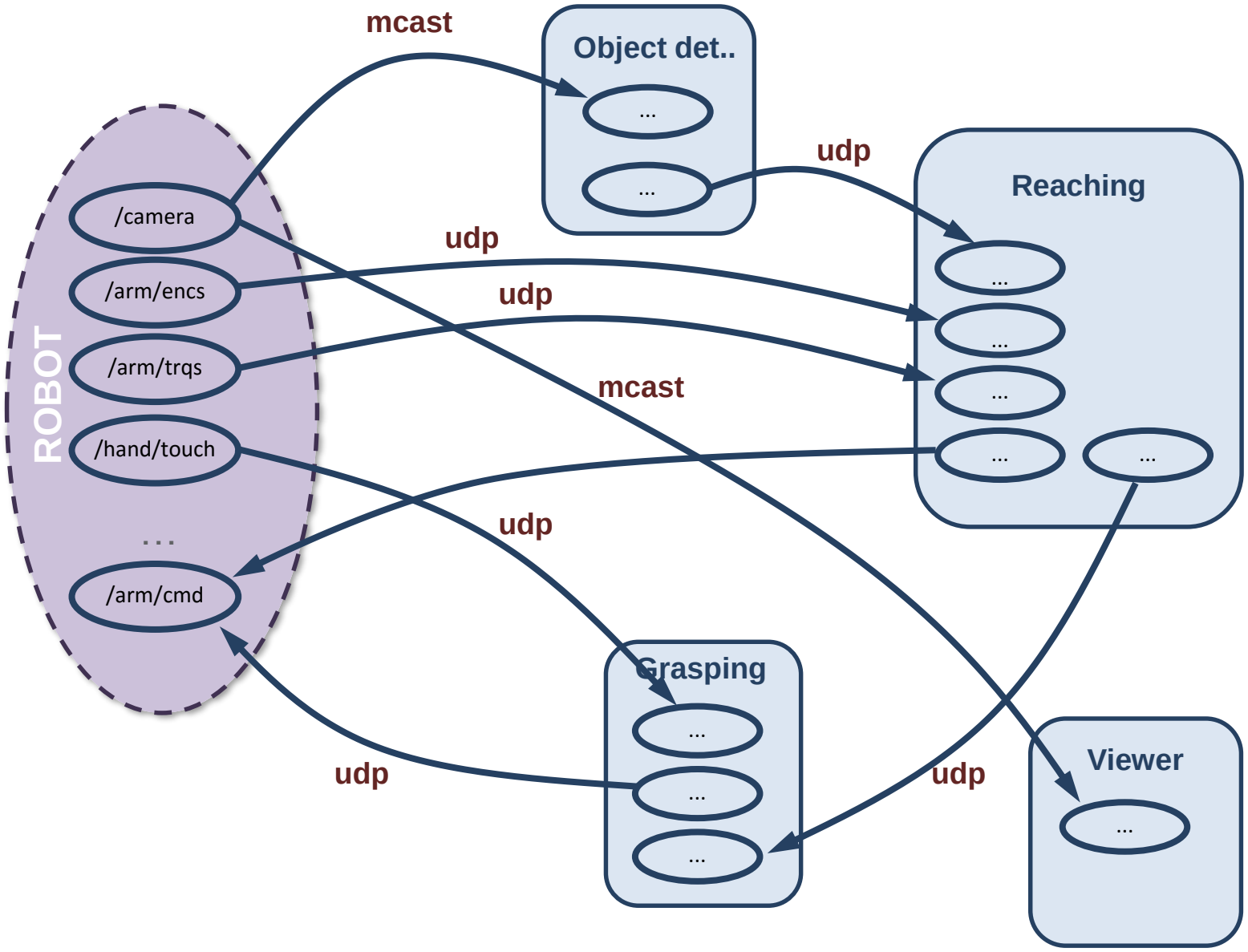
iCub software architecture



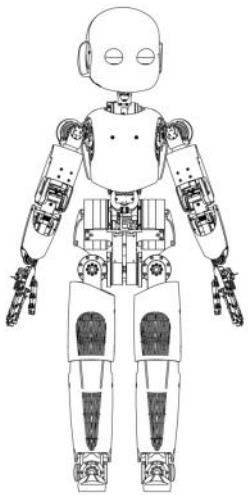


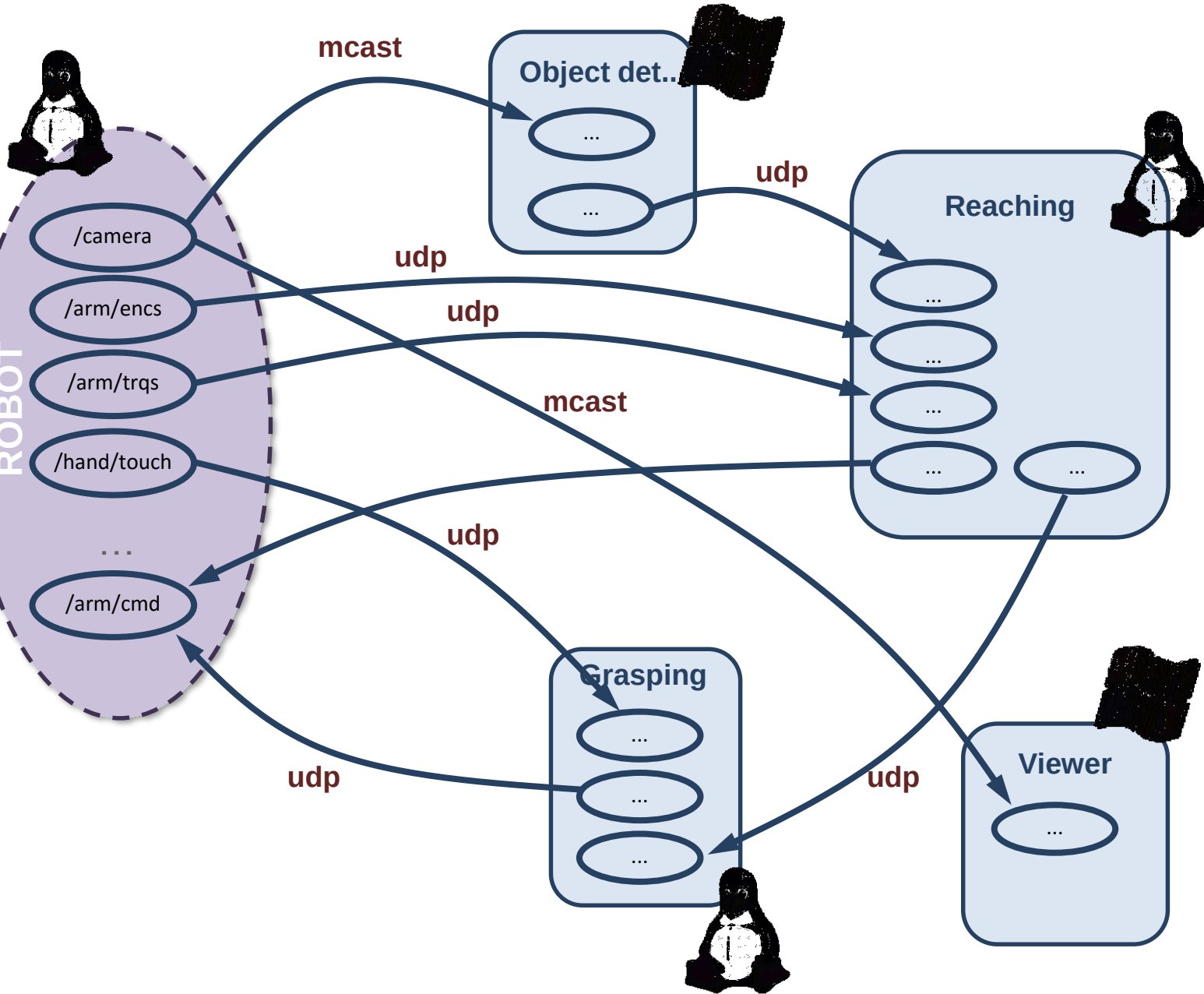
iCub software architecture





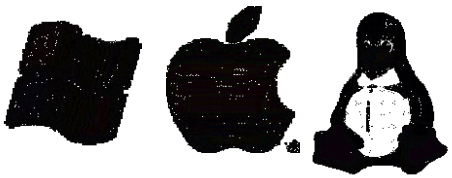
iCub software architecture





YARP: main features

- Peer-to-peer, loosely coupled, communication
- Very stable code base >10 years old
- Flexibility and minimal dependencies, fits well with other systems
- Easy install with binaries on many OSes/distributions (Ubuntu, Debian, Windows, MacOs
- Recently added: channel prioritization with QoS and thread priorities
- Many protocols:
  - Built-in: tcp/udp/mcast
  - Plug-ins: ROS tcp, xml rpc, mjpg etc..



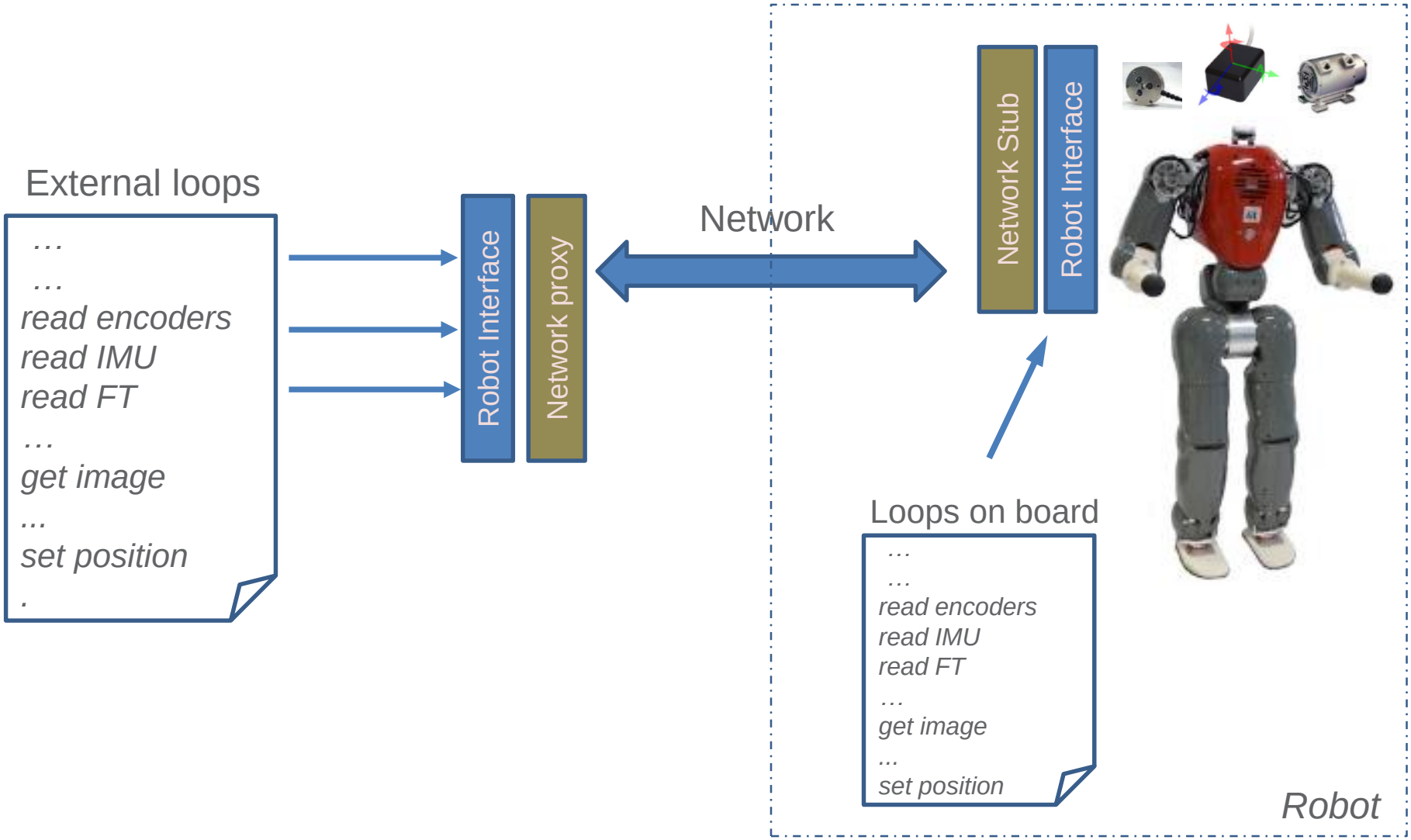
CMake
Cross-platform Make

SWIG

Interfaces

- Define interfaces to motors and sensors so to minimize the impact of changes in the hardware
- Also: network stubs allow remotization

Interfaces

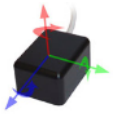

External loops
…
…
read encoders
read IMU
read FT
…
get image
...
set position
.
Robot Interface
Network proxy
Network
Network Stub
Robot Interface
Loops on board
…
…
read encoders
read IMU
read FT
…
get image
...
set position
Robot

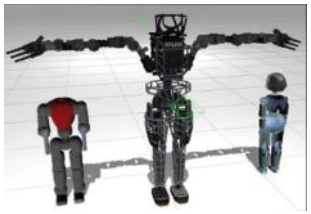
Interfaces

Control loops

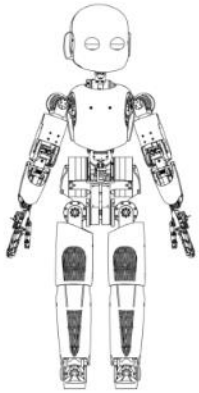
```
…
…
read encoders
read IMU
read FT
…
get image
...
set position
```

Robot Interface





Gazebo



iCub

COMAN

Walkman

Recycle code across different robots and simulators (testing/fast prototyping)

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, etc…
  - bayer carrier, priority based communication

Examples

yarp connect /camera /receiver rec.bayer

Examples

yarp connect /camera /receiver rec.bayer

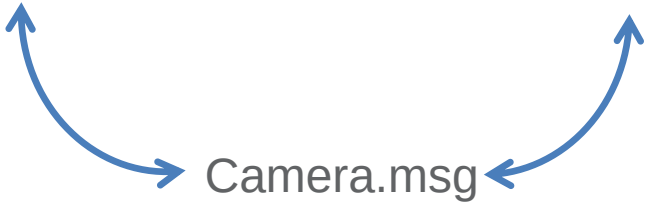
yarp connect /65.52.88.202:5159 /receiver mjpeg

Examples

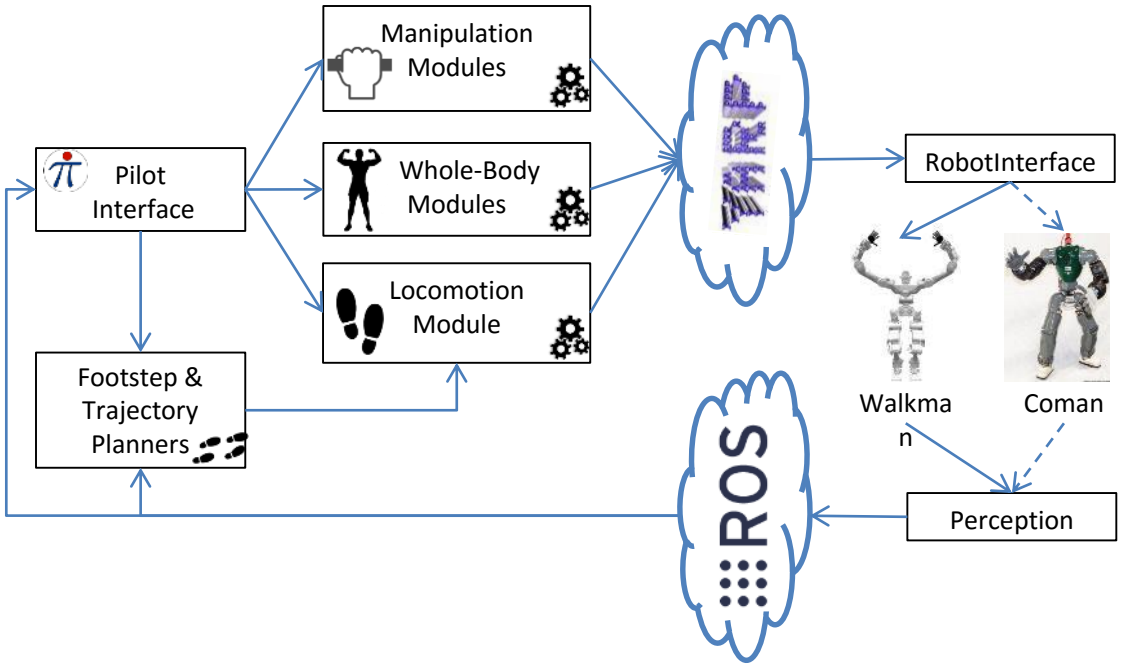
yarp connect /camera /receiver rec.bayer

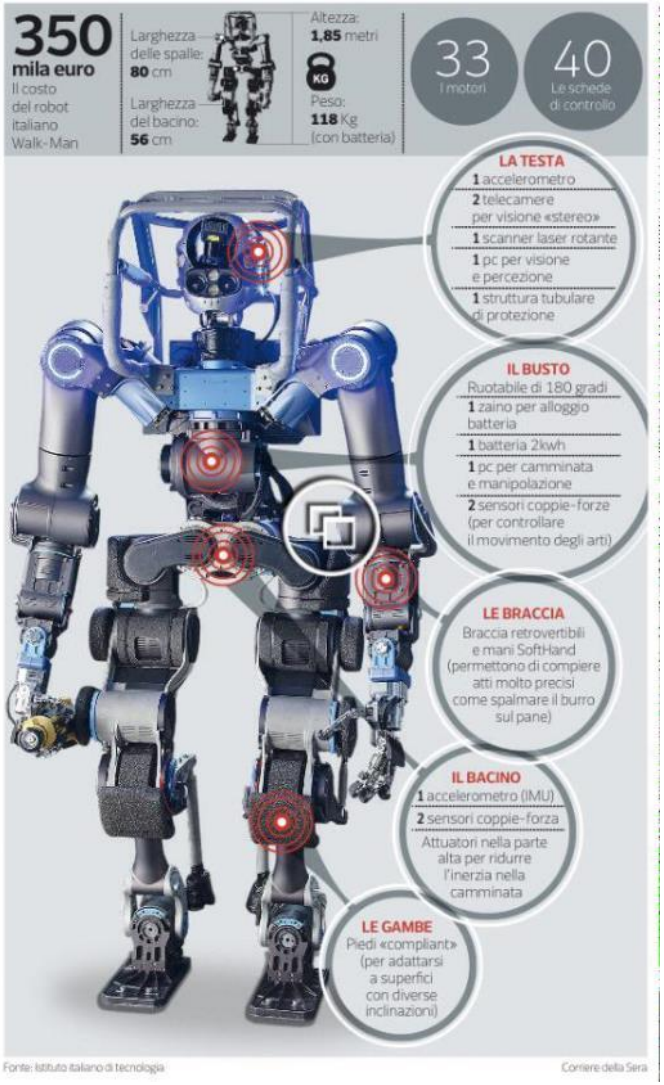
yarp connect /65.52.88.202:5159 /receiver mjpeg

yarp connect /image@/camera /receiver



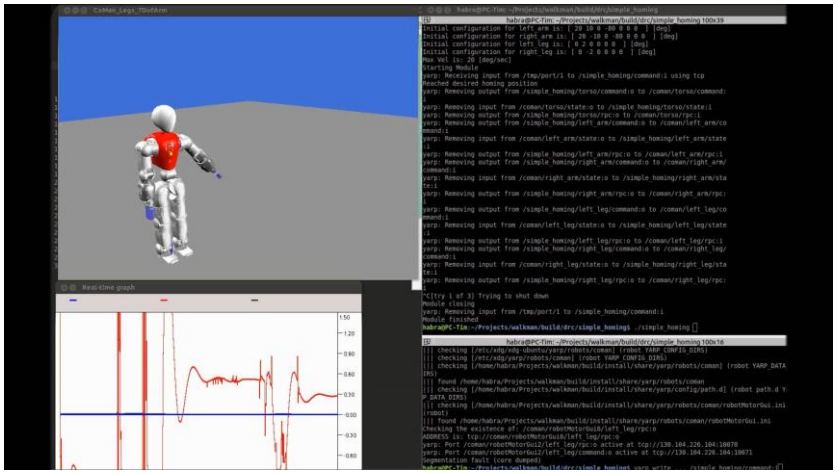
Success story



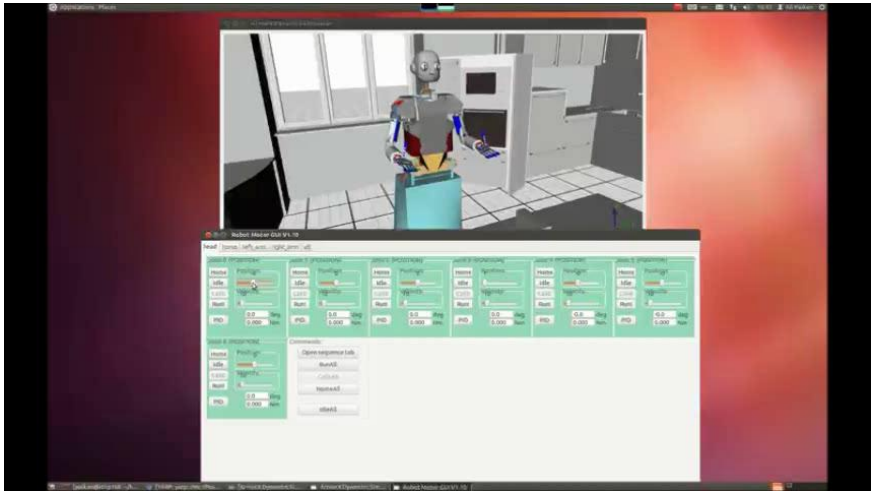


Simulators & robots

- iCub, COMAN, Walk-man, ARMAR III
- iCub_SIM (in the iCub main repository)
- Gazebo (https://github.com/robotology/gazebo-yarp-plugins)
- Robotran



Robotran-Yarp interface: a framework for real-time controller development based on multibody dynamics simulation, T. Habra, et al., 2015



Paikan et al., Transferring Object Grasping Skills and Knowledge Across Different Robotic Platforms, ICAR 2015

Managing complexity

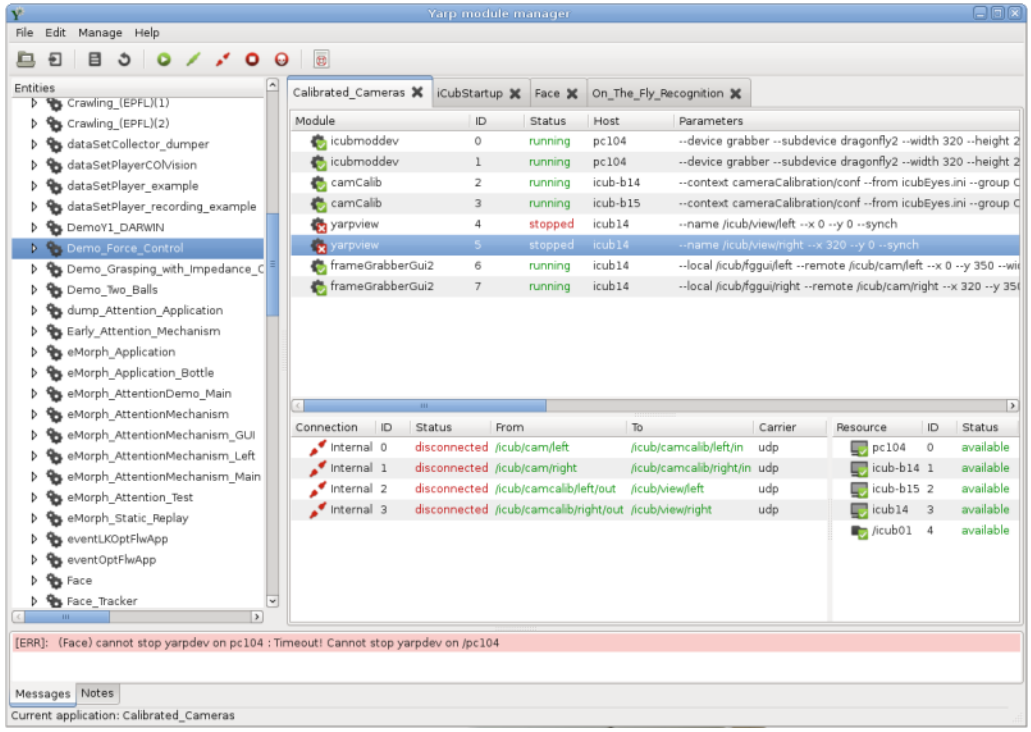
In a modular system integration becomes an issue:

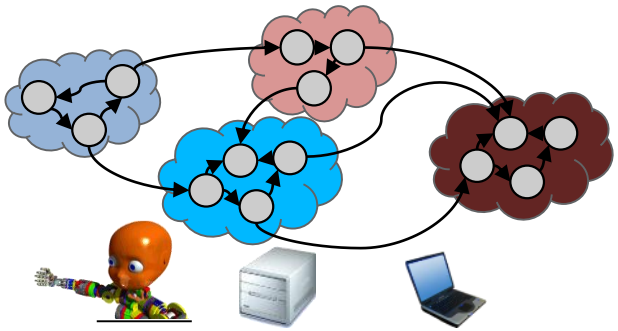
- Execution and monitoring
- Development
- Coordination

Execution and monitoring: YARP manager

Required modules connections nodes resources

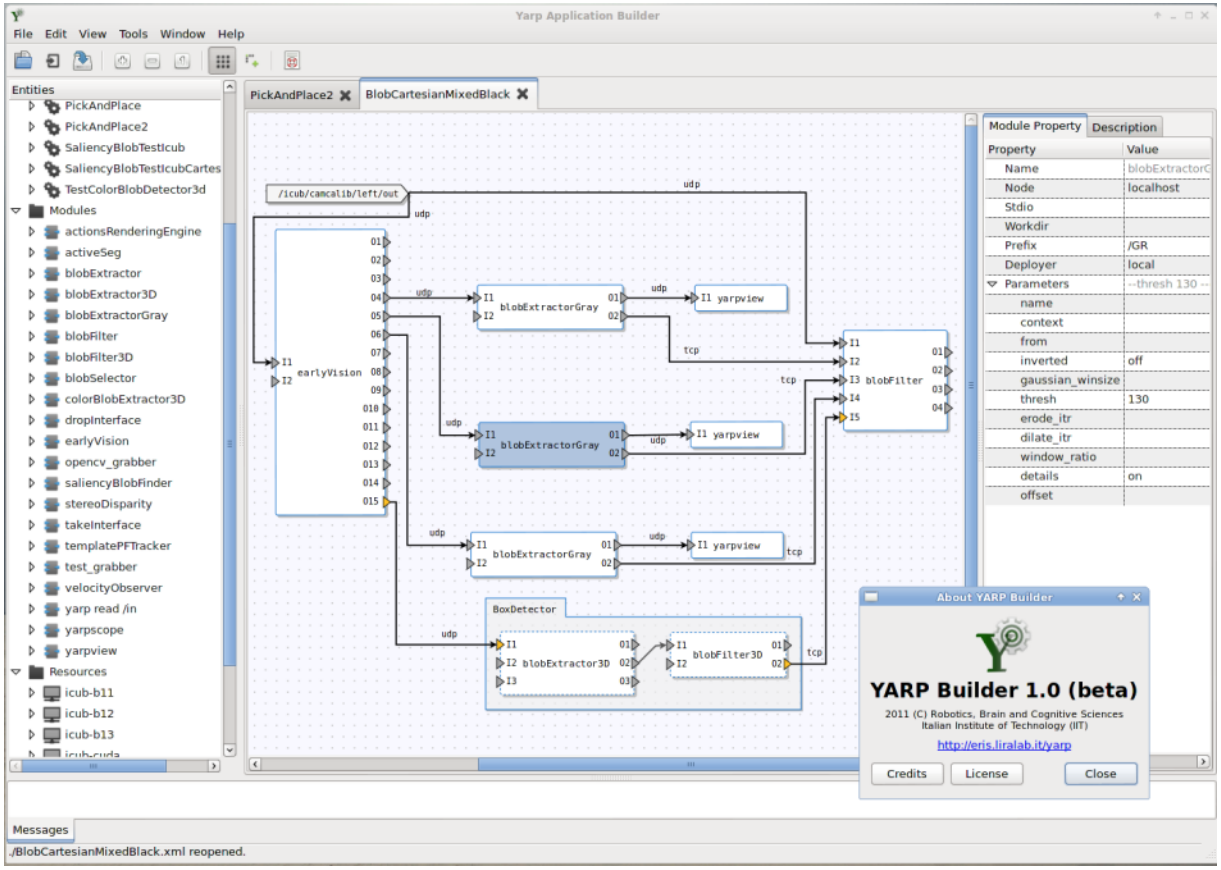
Available resources



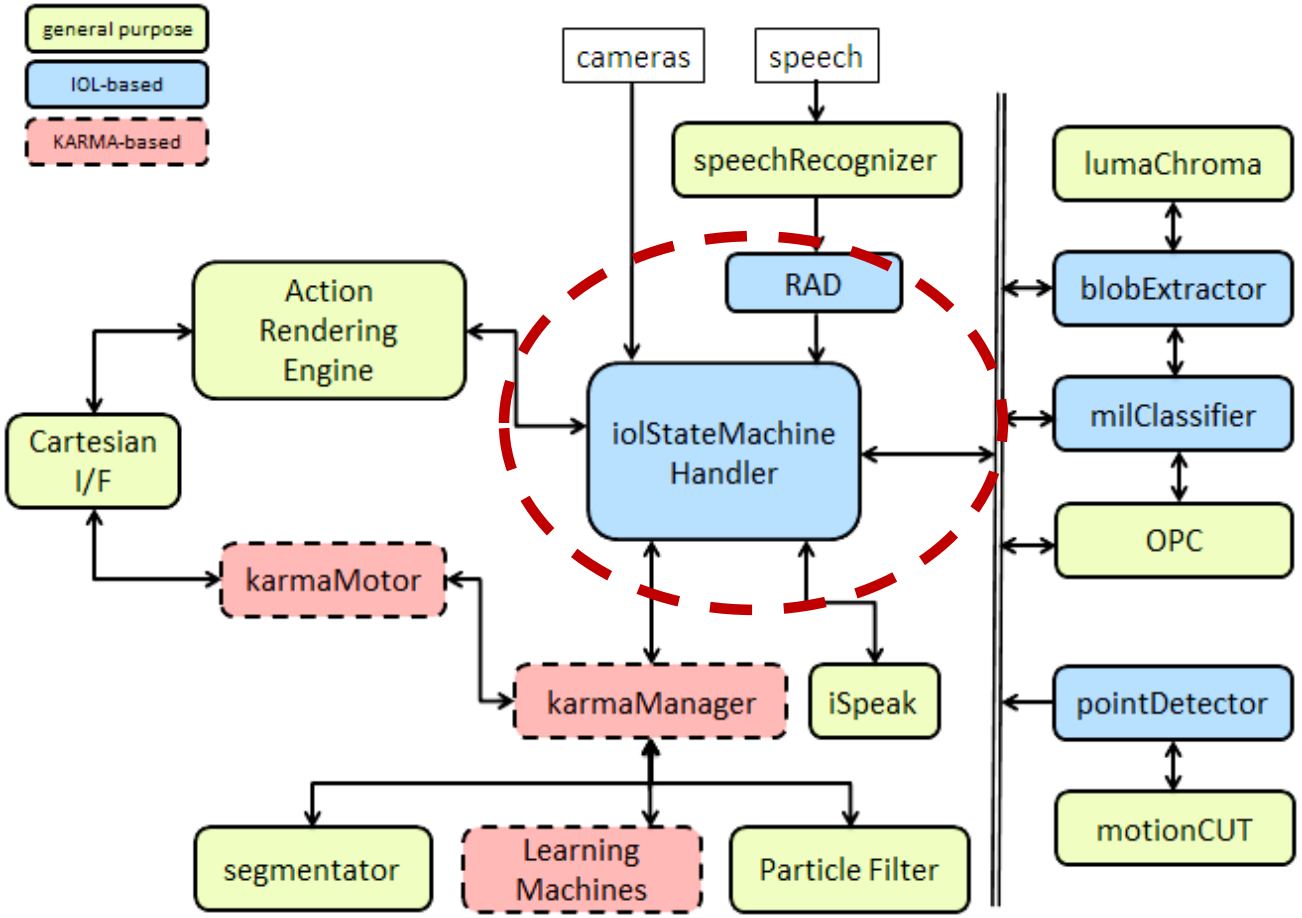


Tools for rapid development

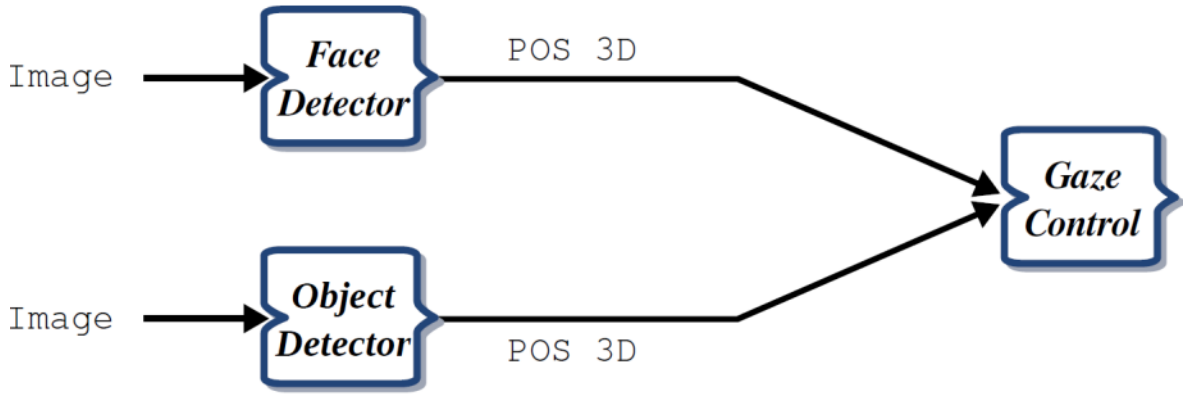
- YARP builder: graphical tool to design application
- Interface Definition Language (IDL):
  - formalization of types and interfaces between modules
  - automatic generation of message handlers



Coordinating modules

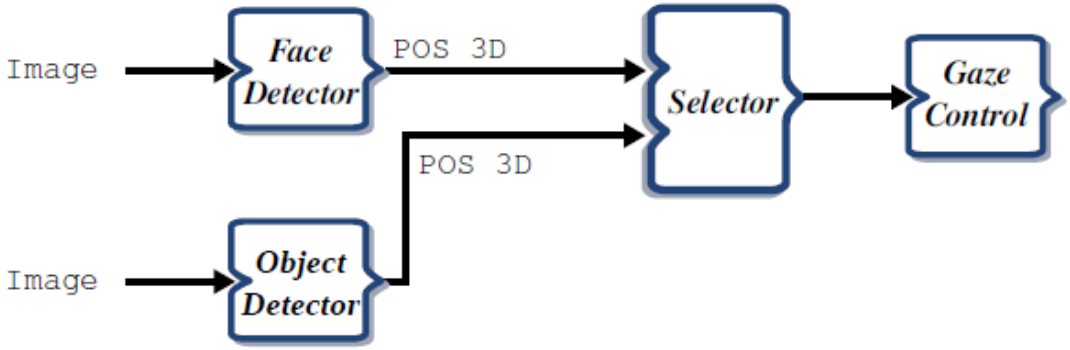


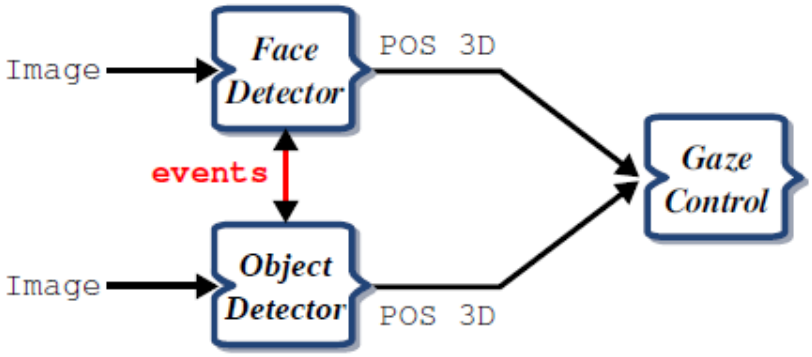
Arbitration and coordination

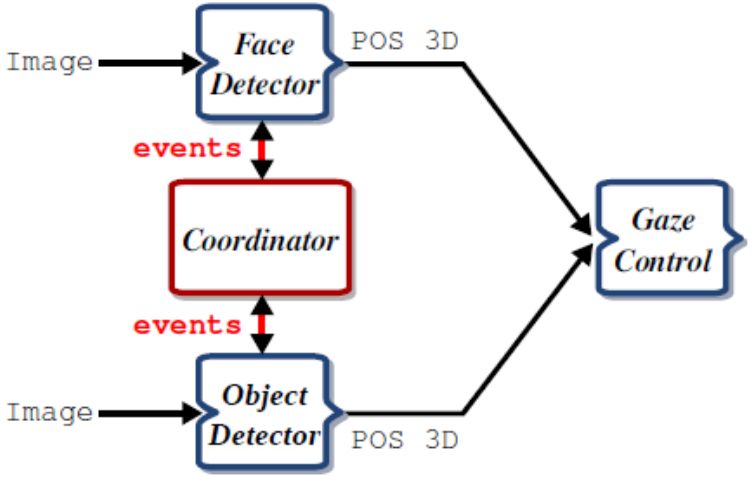


Paikan, A., et. al., *Data Flow Port's Monitoring and Arbitration*, Joser 2014.

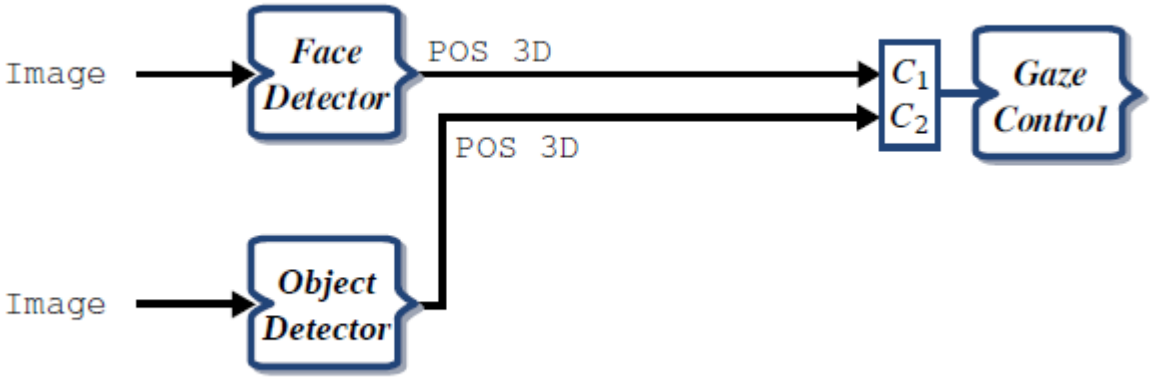
Arbitration and coordination



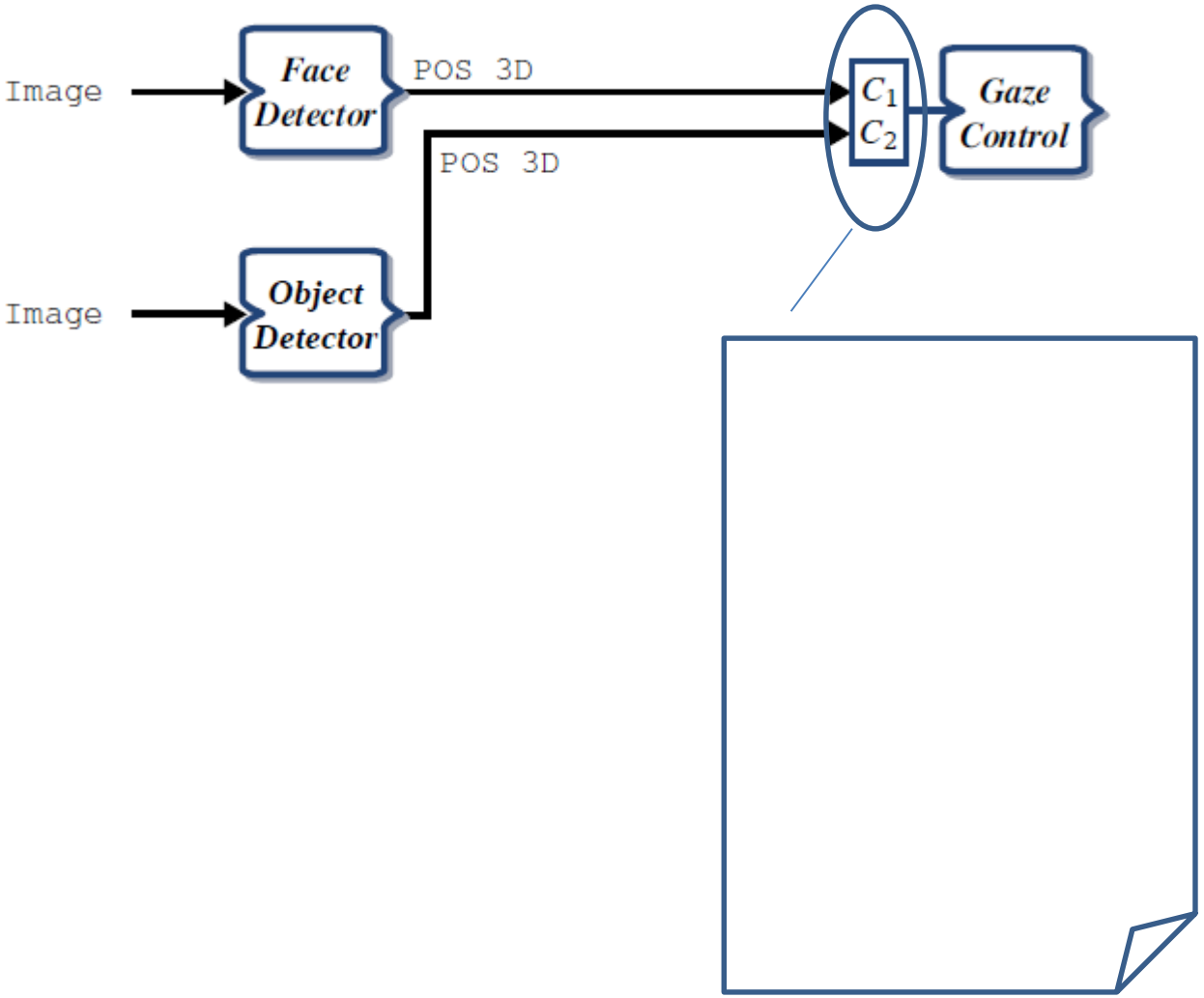




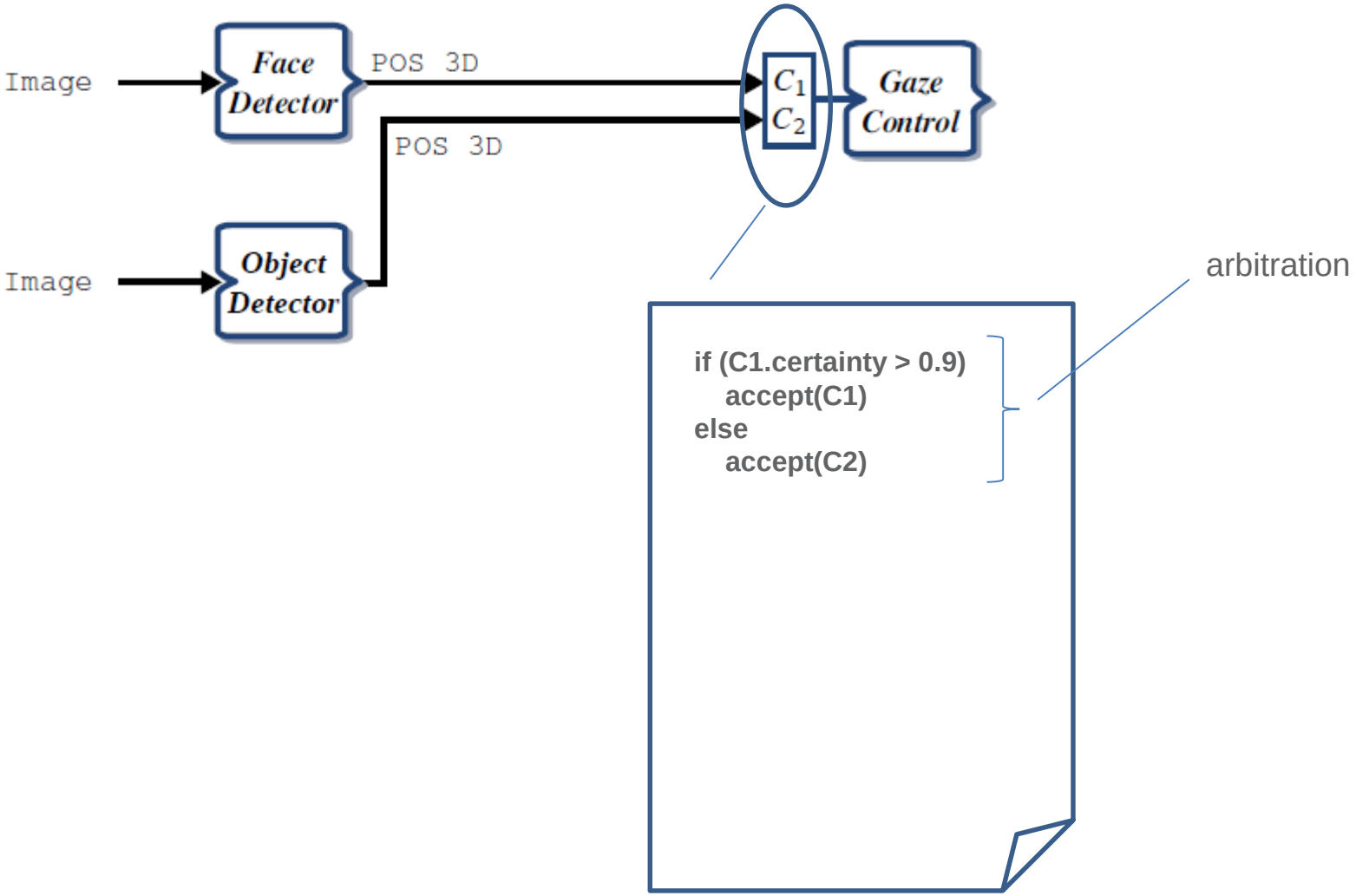
Our solution: Port Monitor



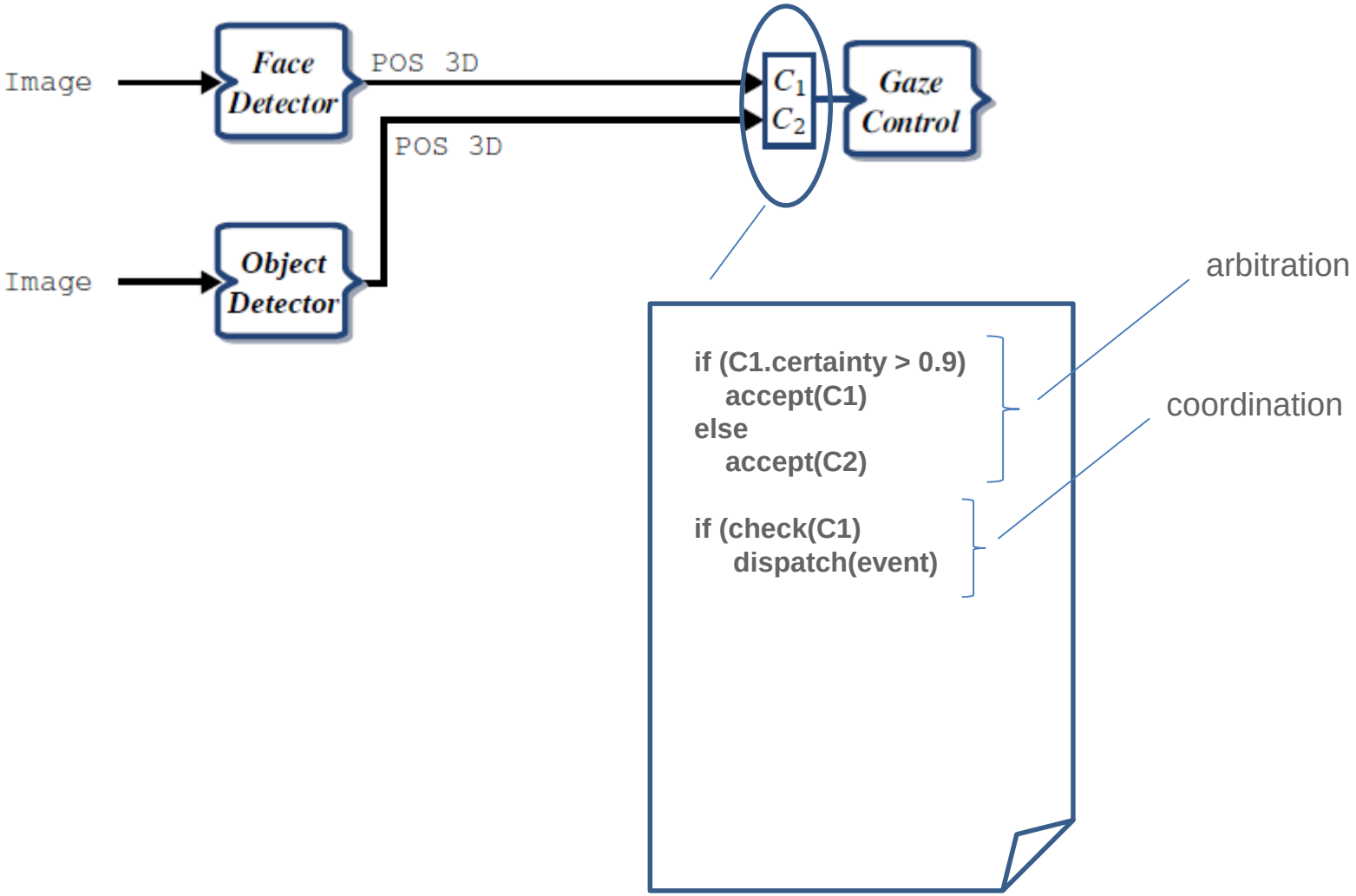
Our solution: Port Monitor



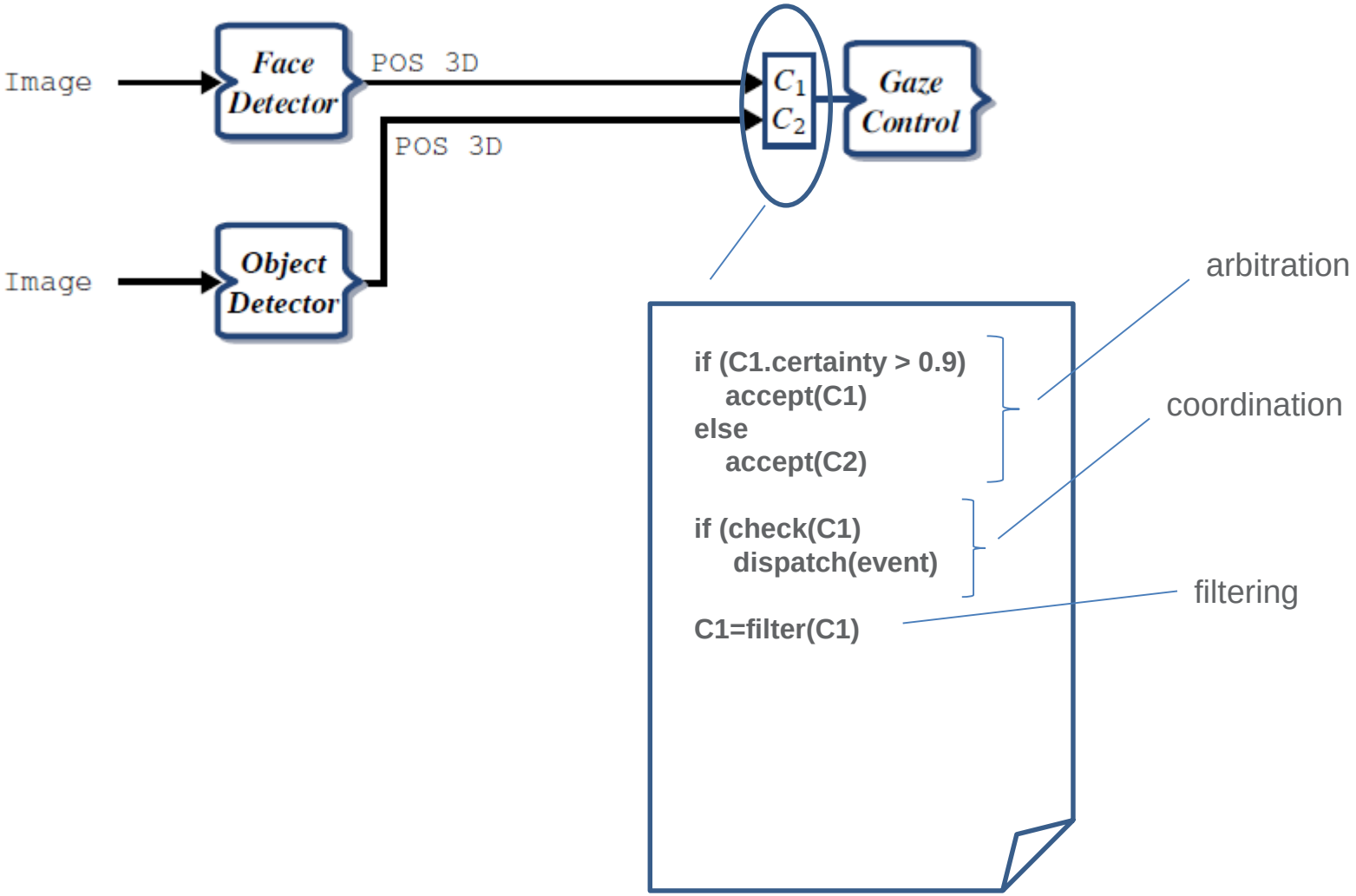
Our solution: Port Monitor



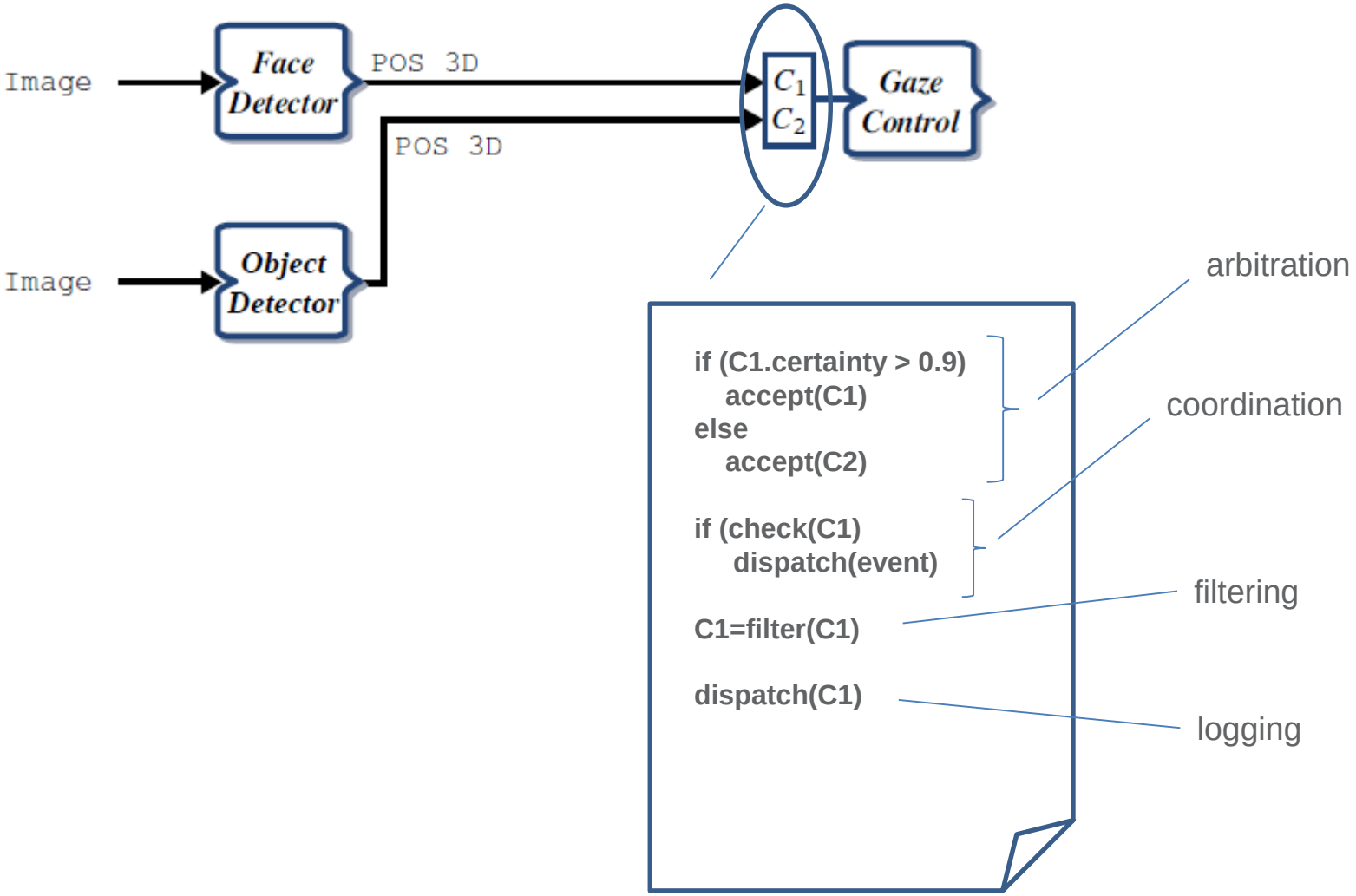
Our solution: Port Monitor



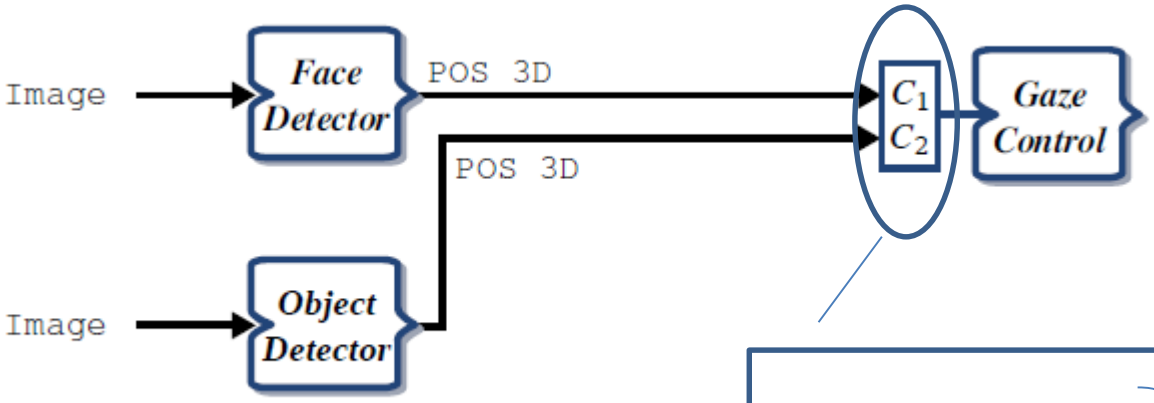
Our solution: Port Monitor



Our solution: Port Monitor



Our solution: Port Monitor




```
if (C1.certainty > 0.9)
    accept(C1)
else
    accept(C2)

if (check(C1)
     dispatch(event)

C1=filter(C1)

dispatch(C1)

If (C1)
    T1=getTime()
```

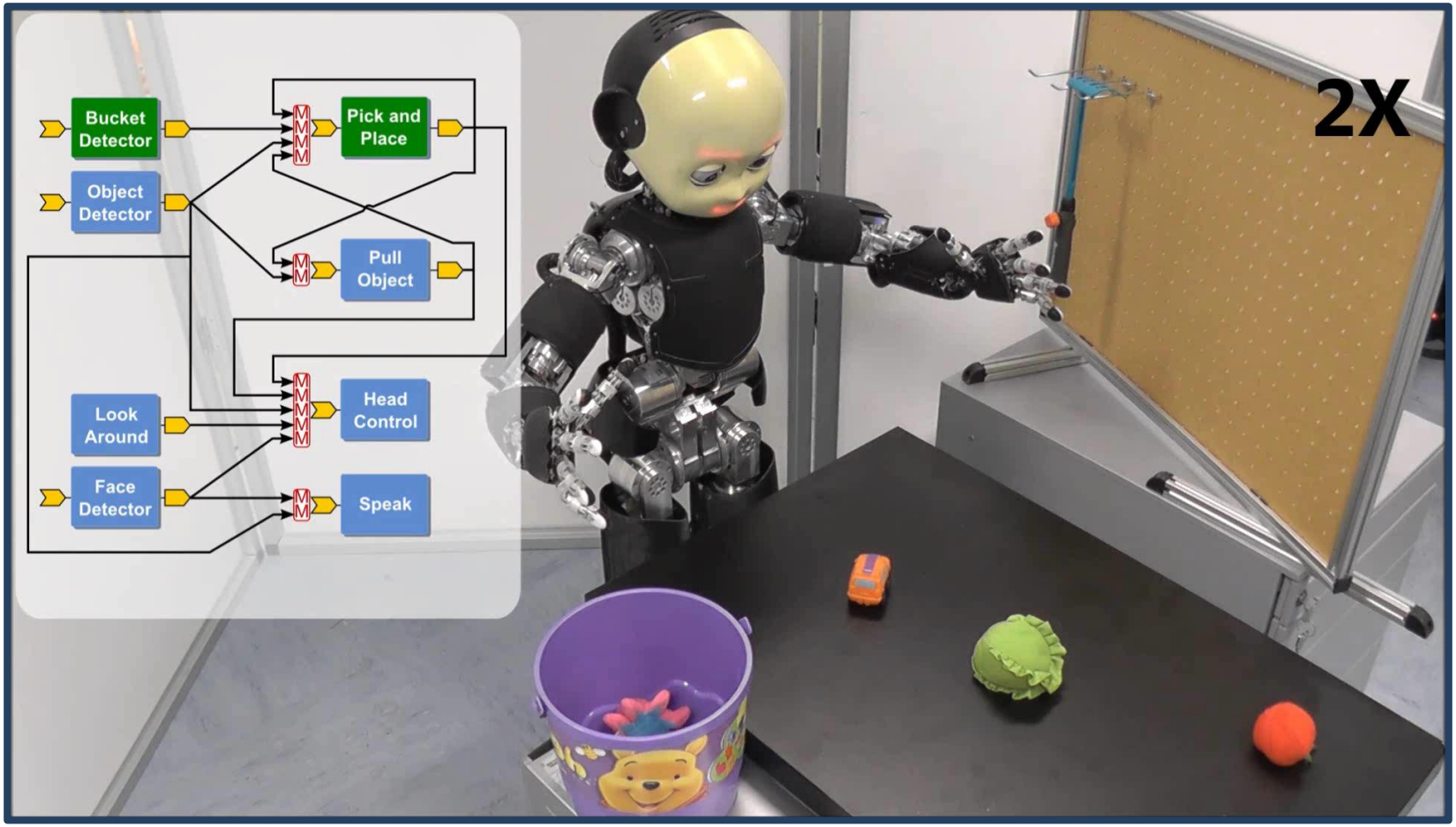
arbitration

coordination

filtering

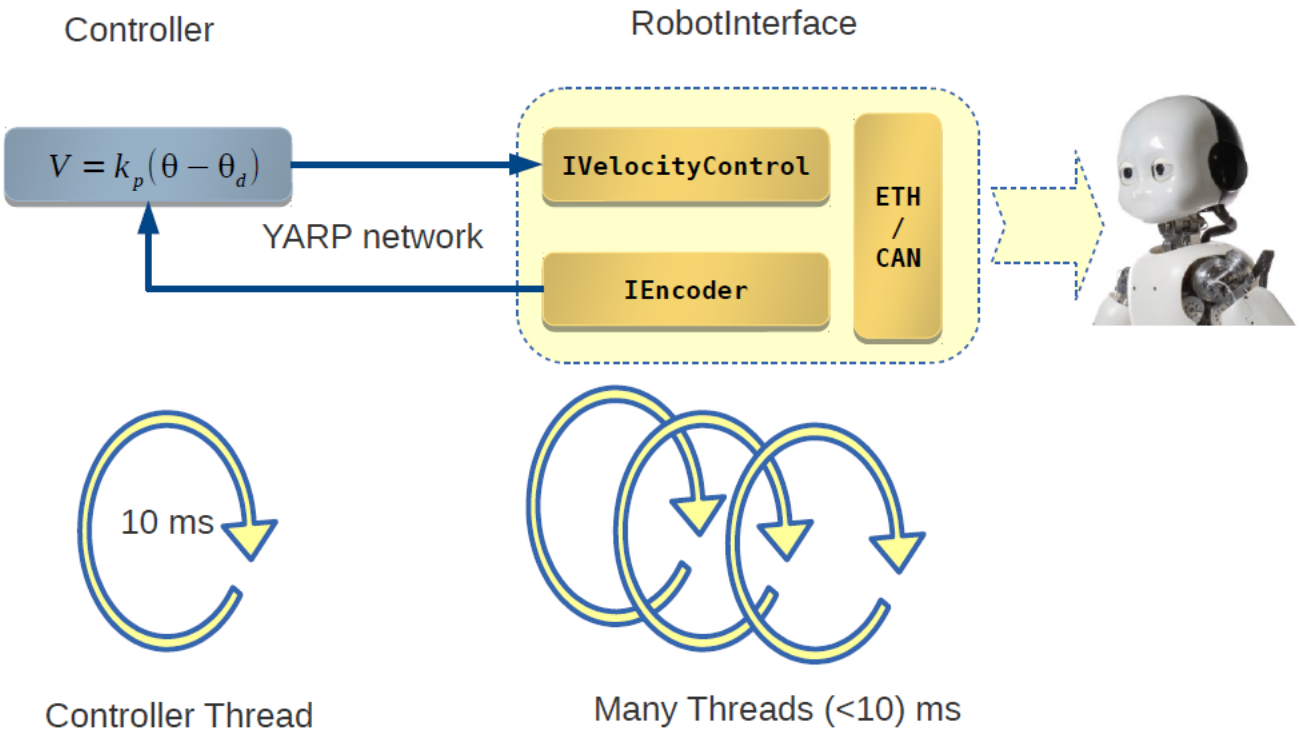
logging

Monitoring delays, QoS



Paikan, A., et al., *Enhancing software module reusability using port plug-ins: an experiment with the iCub robot*, IROS 2014.

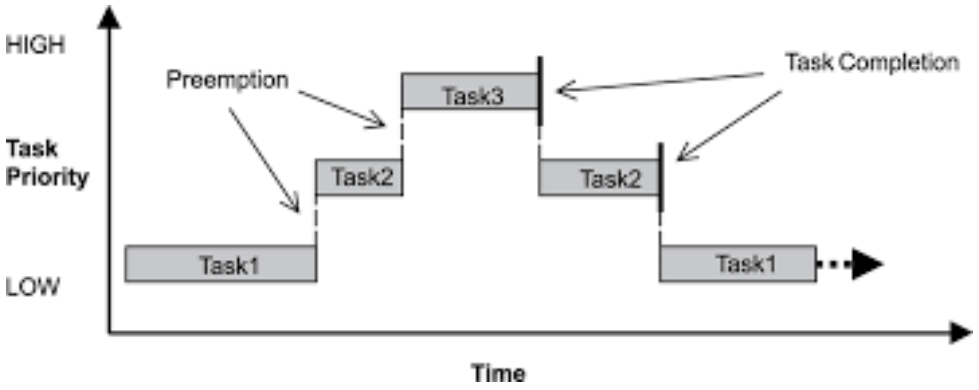
Improving determinism

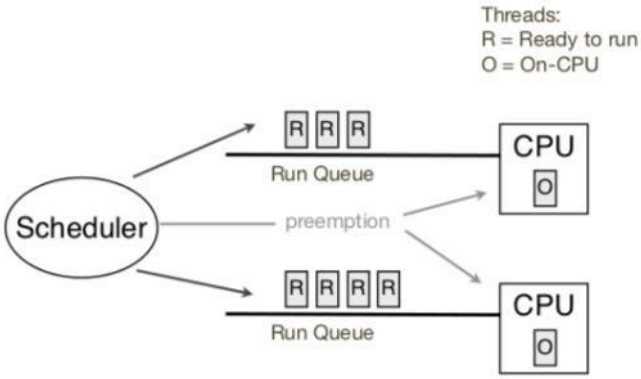


QoS

Quality of Service

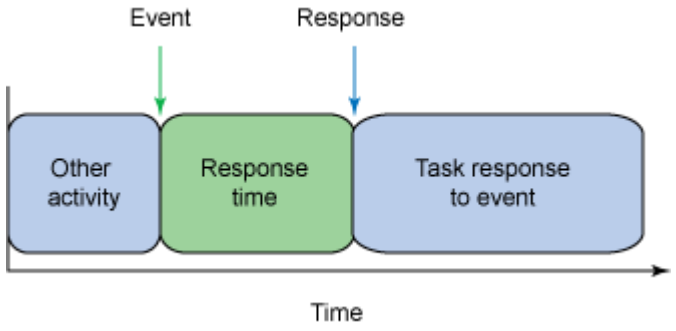
Threads and scheduling, *quick* overview





Issues:

- Scheduling and interrupt latency
- Certain operations may be *blocking* i.e. not preemptable (usually system calls)
- System calls: file management, process management, I/O, memory requests, communication (shared memory), error handling

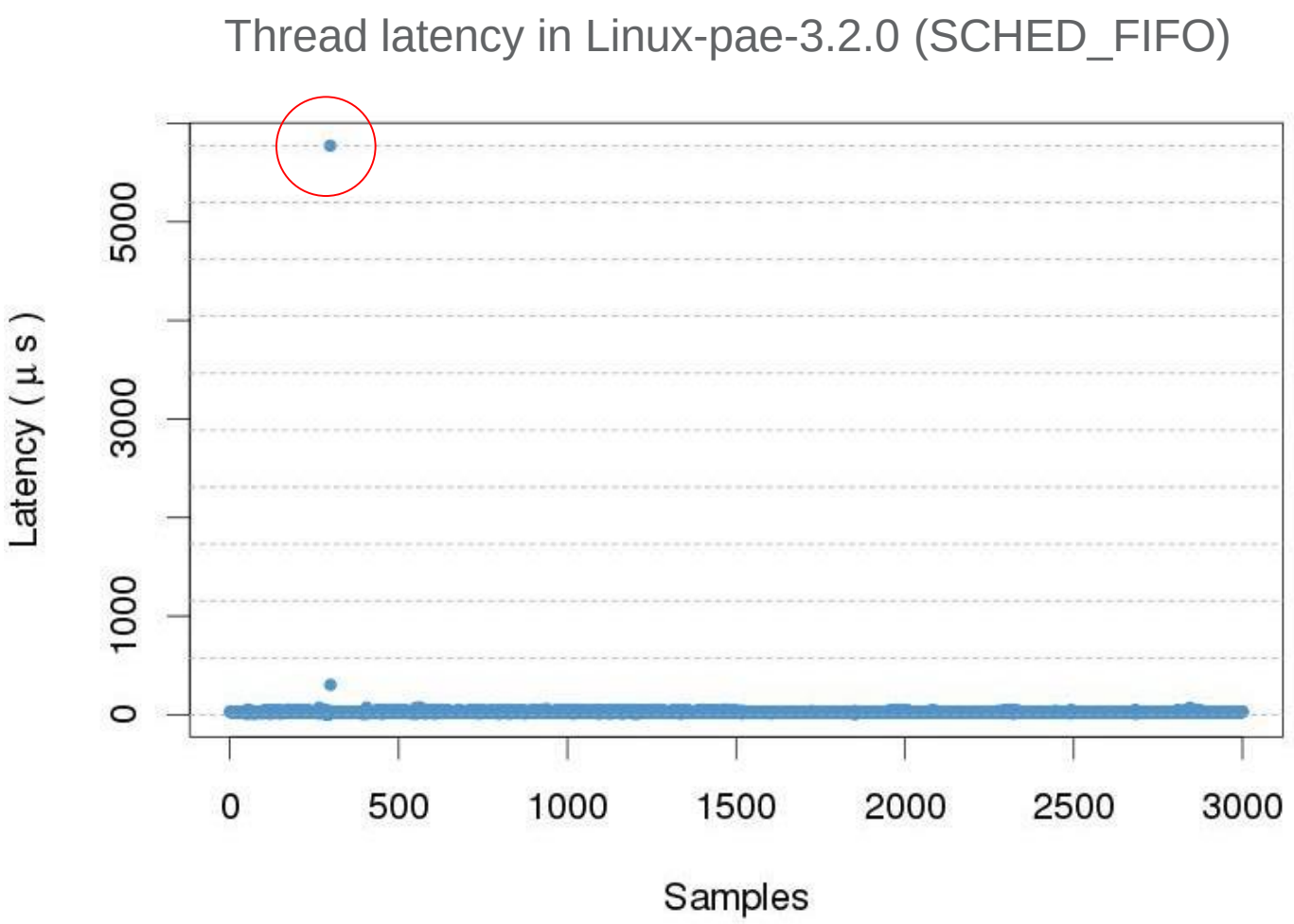


Which RT solution

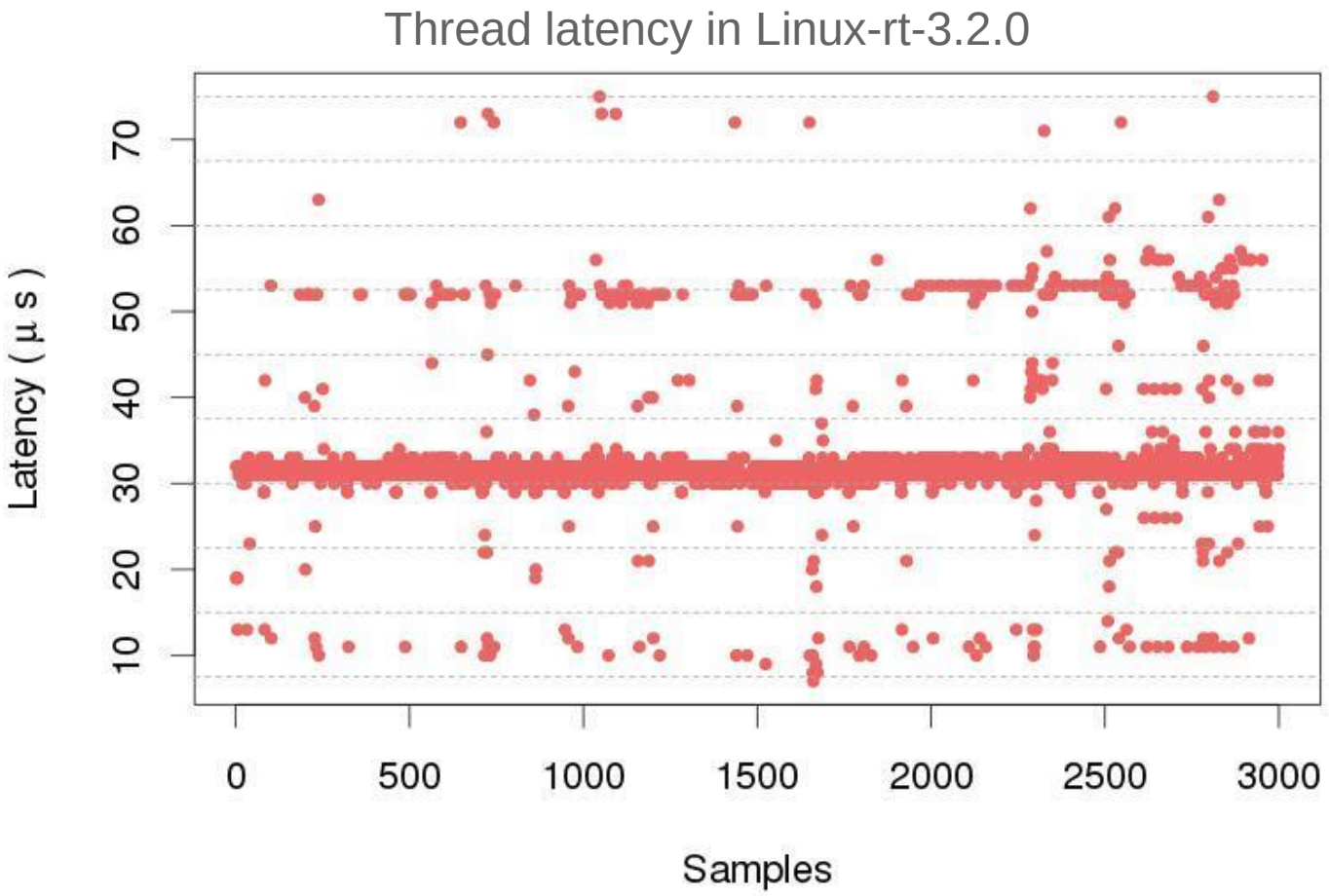
- Standard Linux
- Linux with RT patch
- Xenomai/RTAI

The latency of a Linux thread with the highest priority (period 1ms)

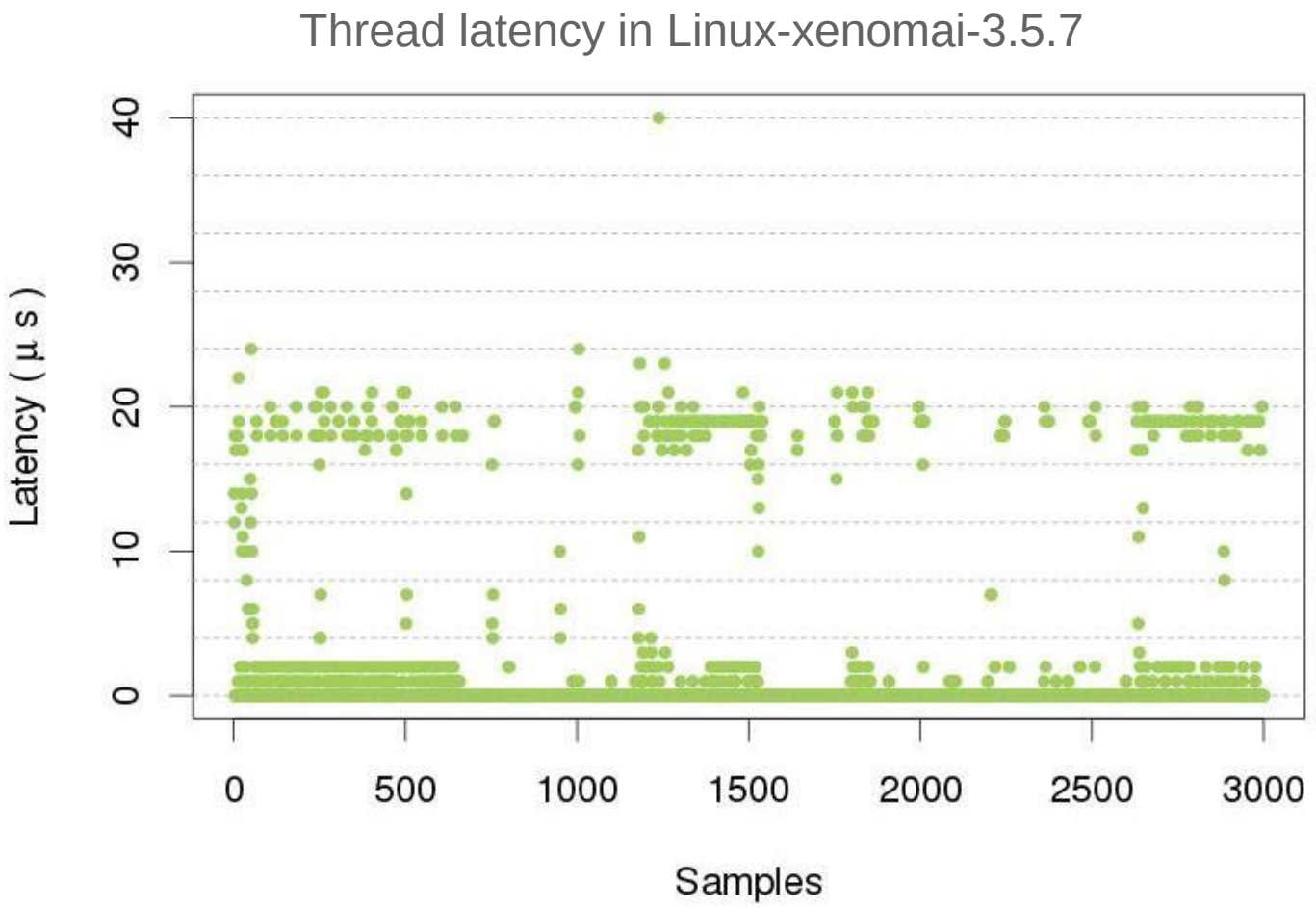
(period: 1ms, PC104 / 1Ghz)



The latency of a Linux thread with preemption patch (period: 1ms, PC104 / 1Ghz)

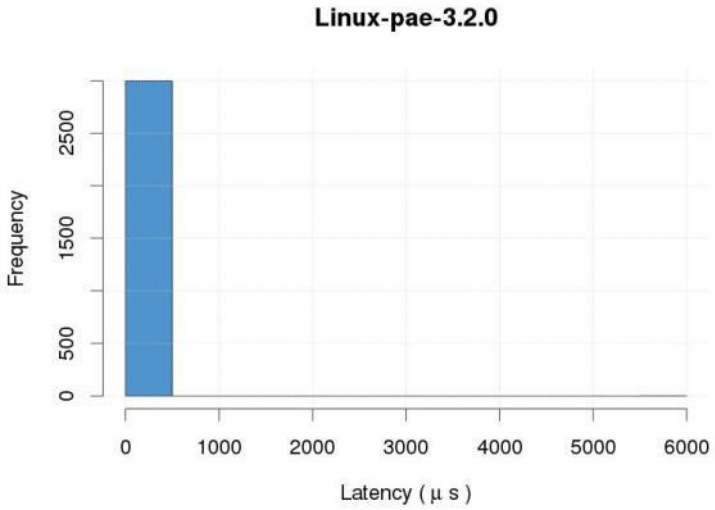


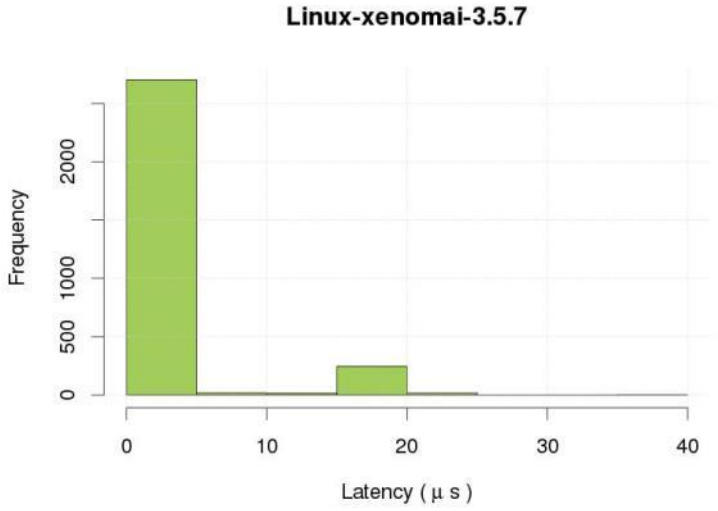
The latency of a Linux thread with Xenomai patch (period: 1ms, PC104 / 1Ghz)

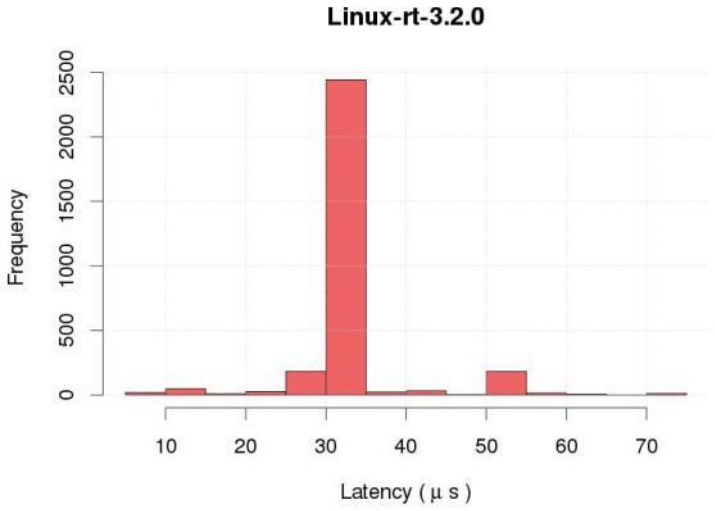


Which RT solution?

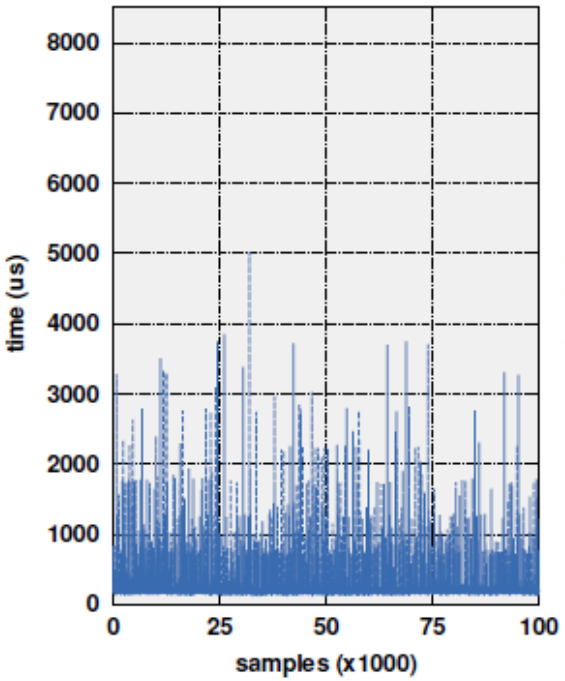
Standard Linux
Linux with RT patch (our choice)
Xenomai/RTAI



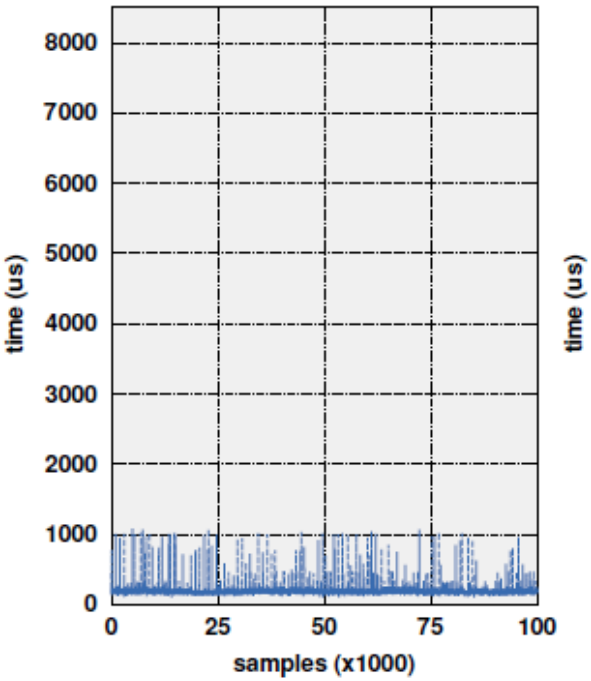




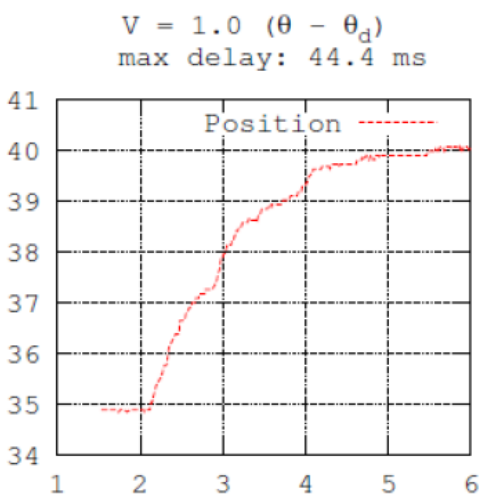
Standard YARP

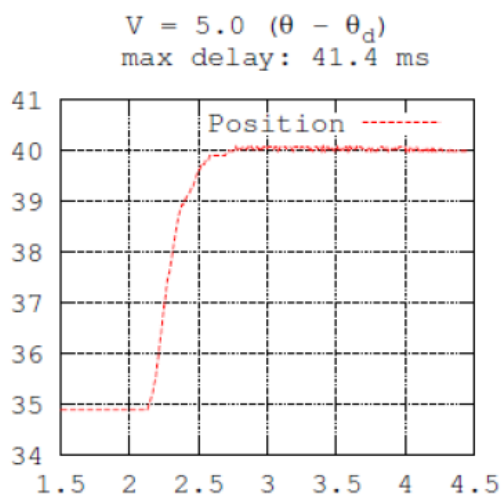


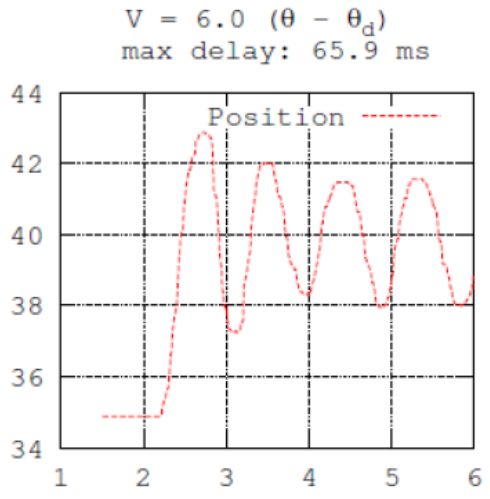
With priority

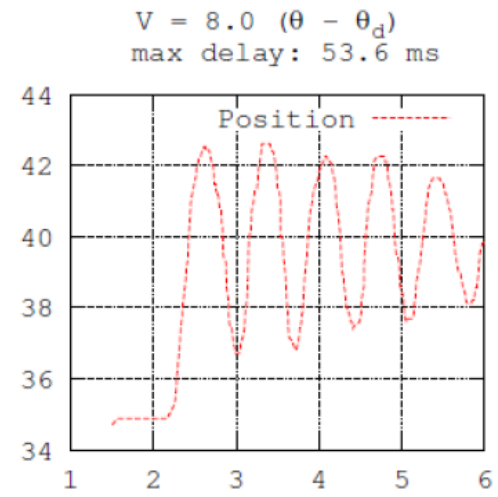


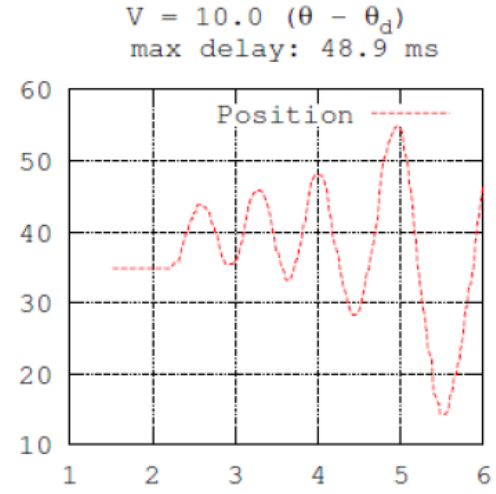
System under heavy load

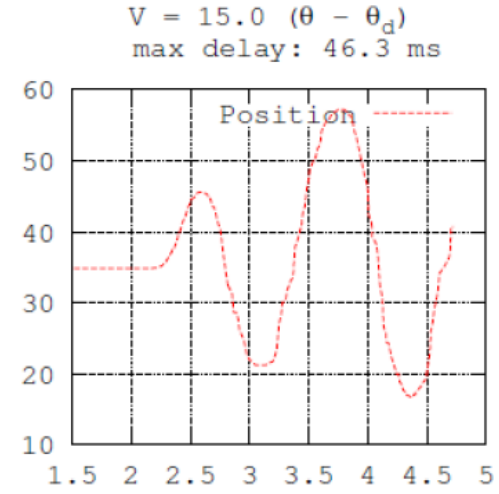




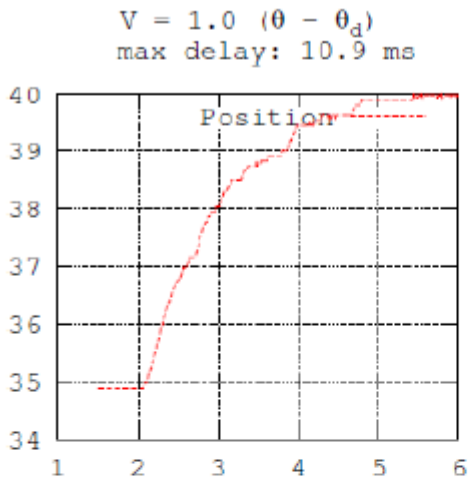


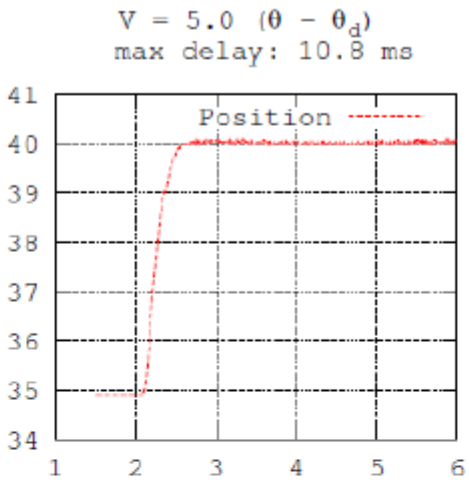


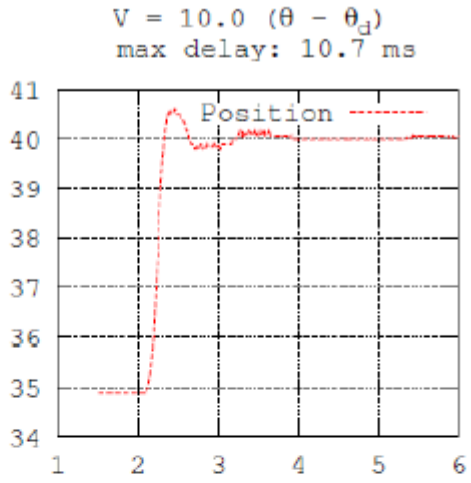


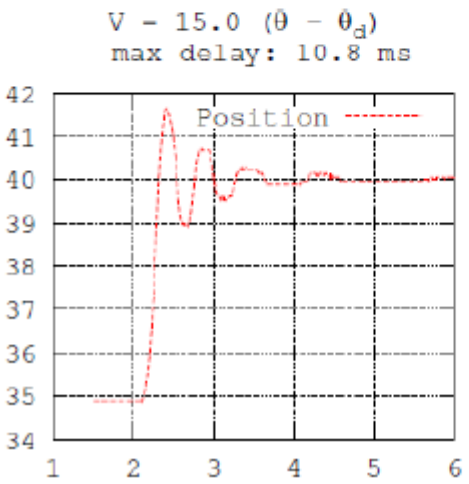


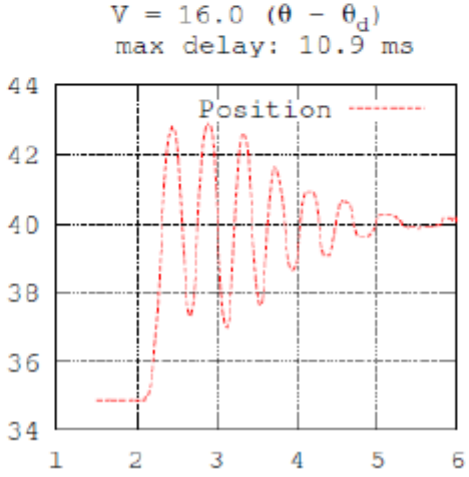
After adjusting thread priorities

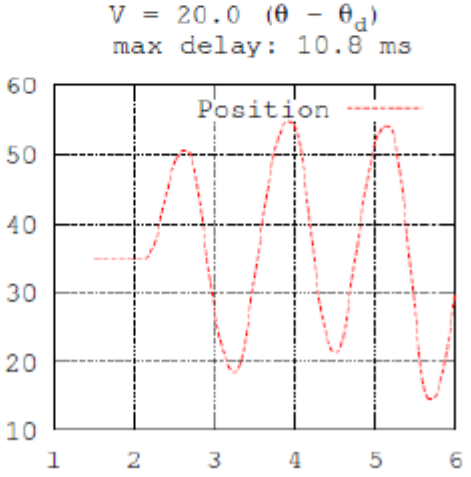






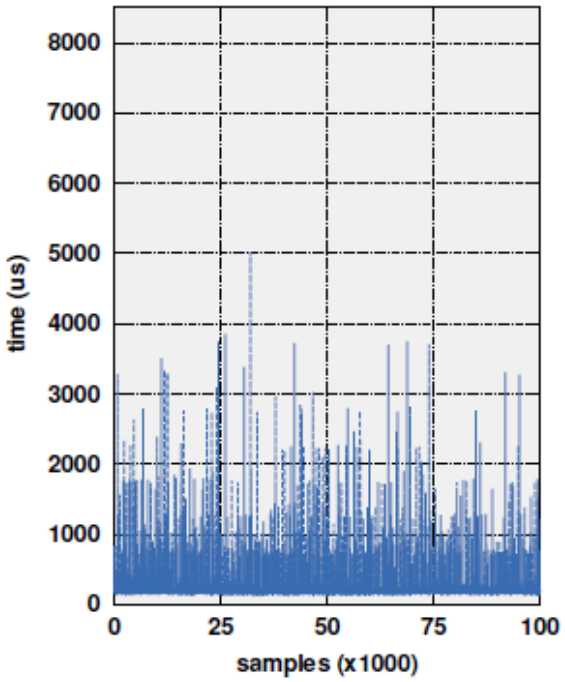




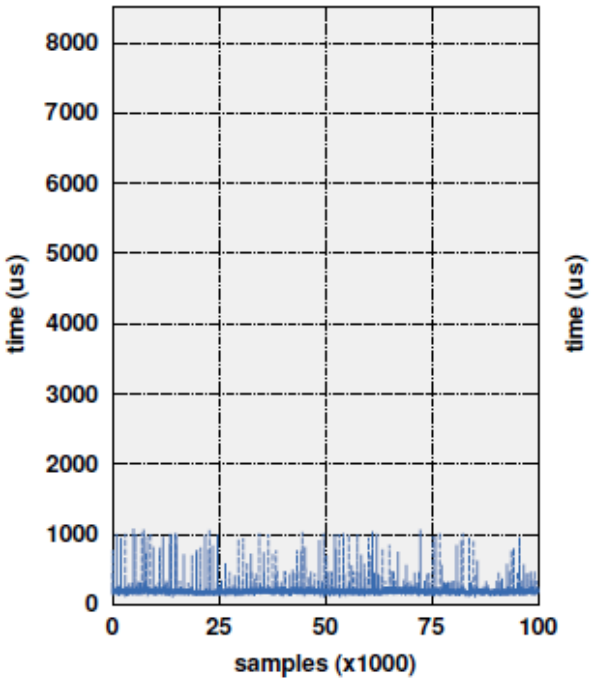


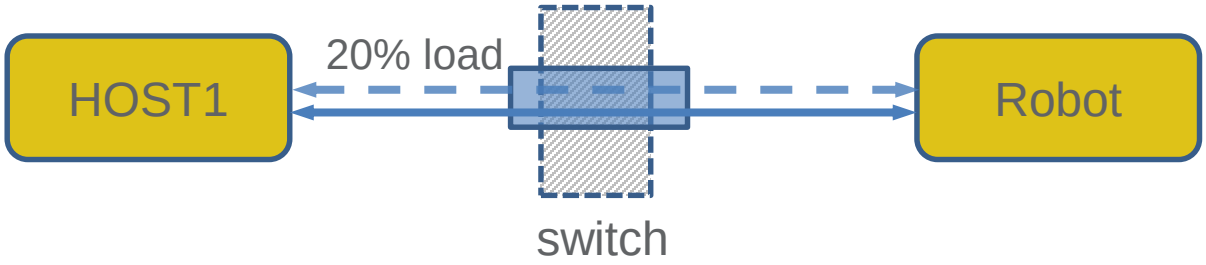
switch

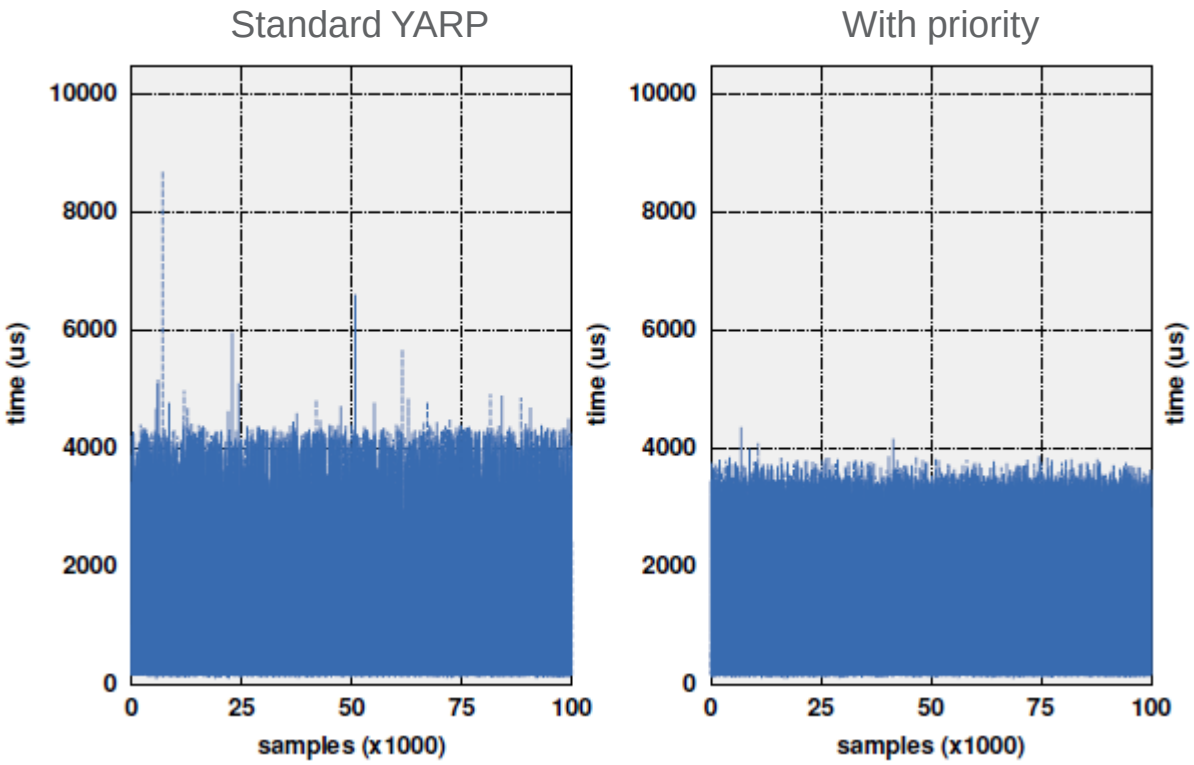
Standard YARP

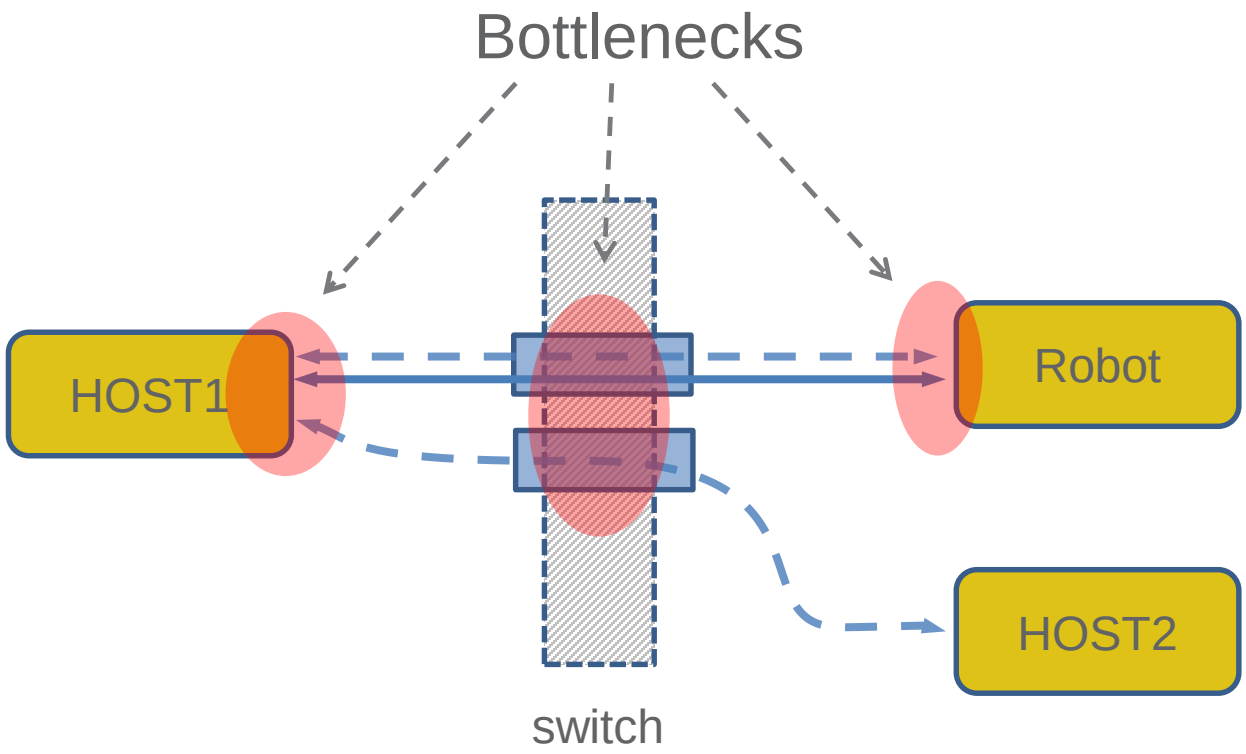


With priority

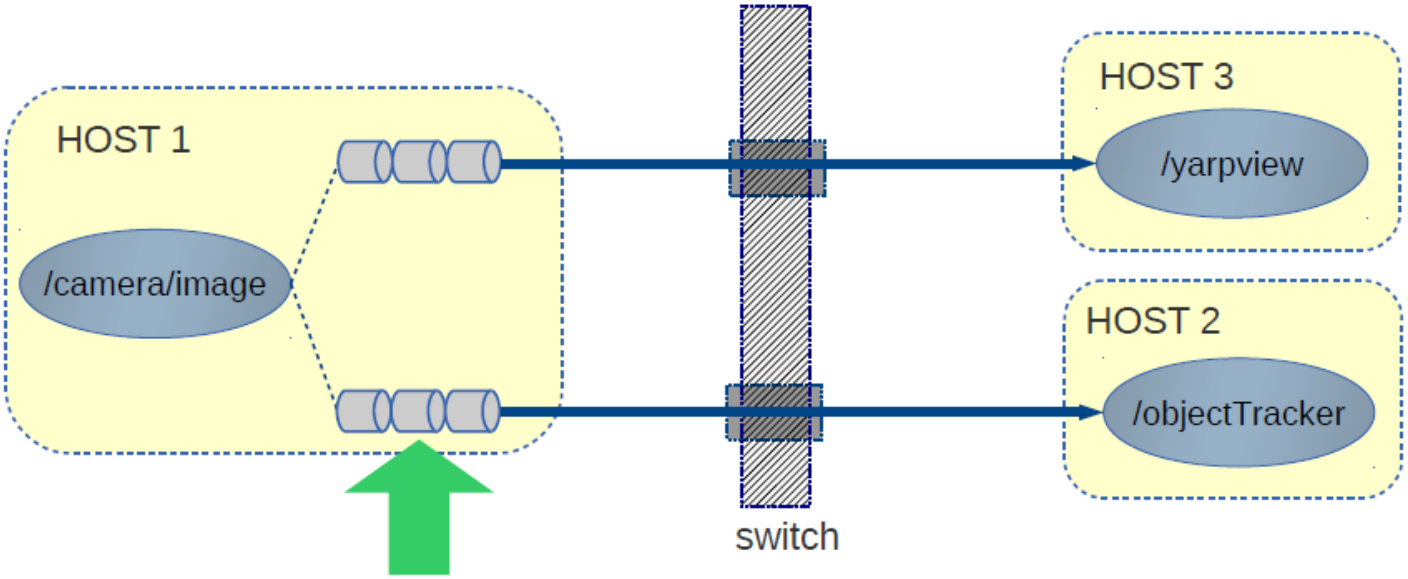



HOST1
20% load
Robot
switch


Standard YARP
With priority
time (us)
samples (x1000)
0
2000
4000
6000
8000
10000
0
25
50
75
100

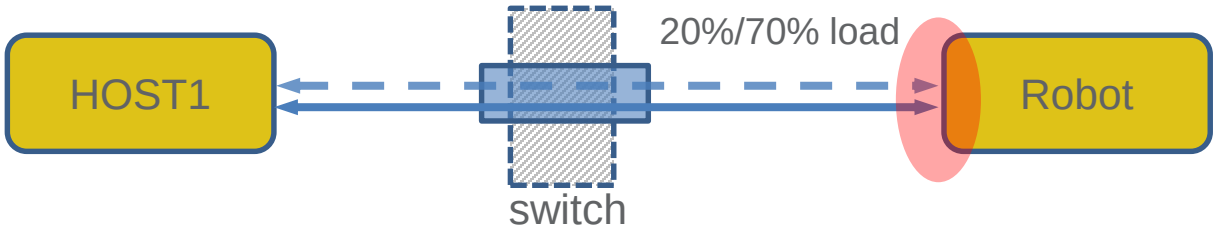

Bottlenecks
HOST1
Robot
HOST2
switch

- Approach: improve determinism by increasing thread priorities and reducing network bottlenecks using QoS

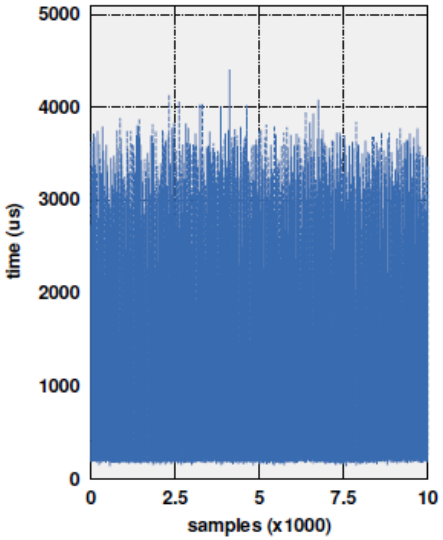



```
> prop sched policy 1 priority 30
> prop set qos priority HIGH
```

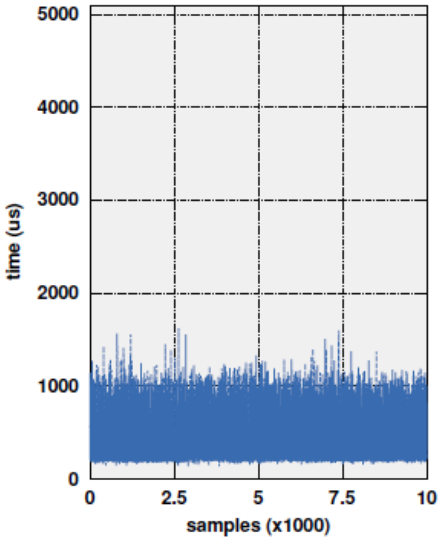
Paikan et al. IROS 2015



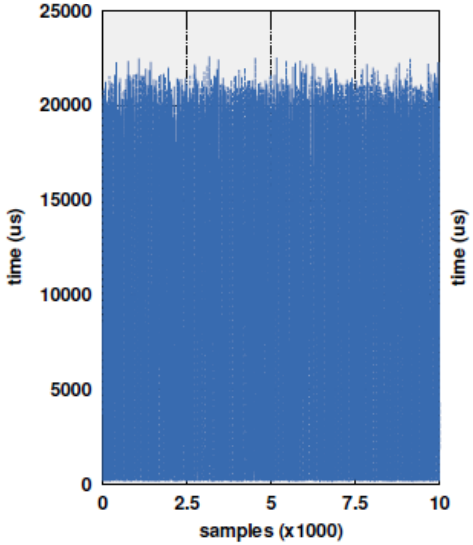
Standard YARP (20%)



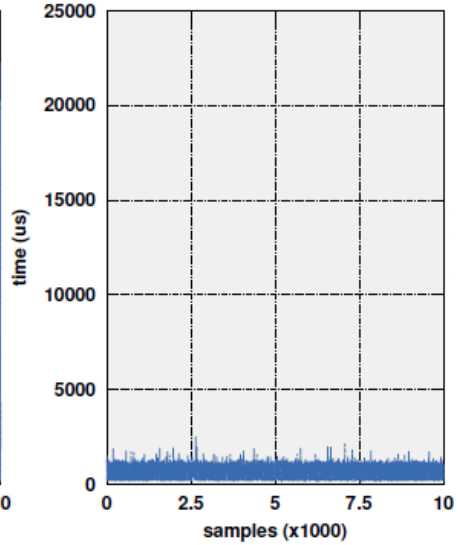
With priority (20%)

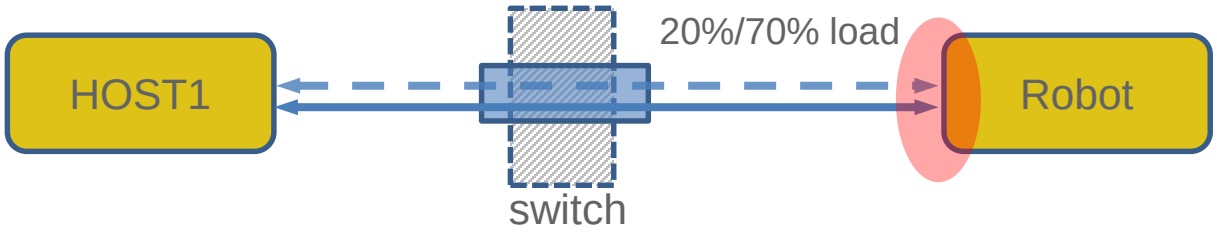


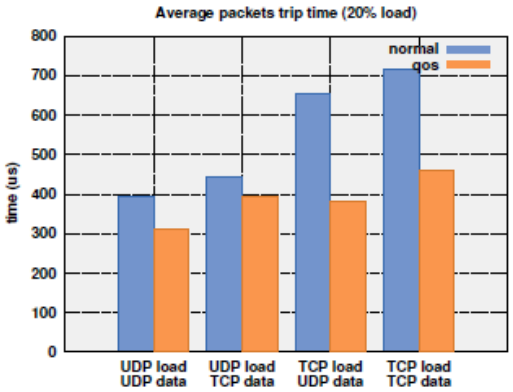
Standard YARP (70%)

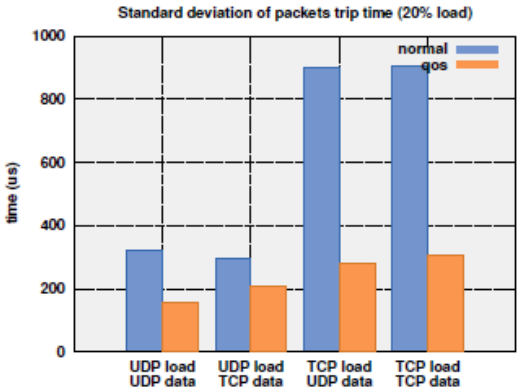


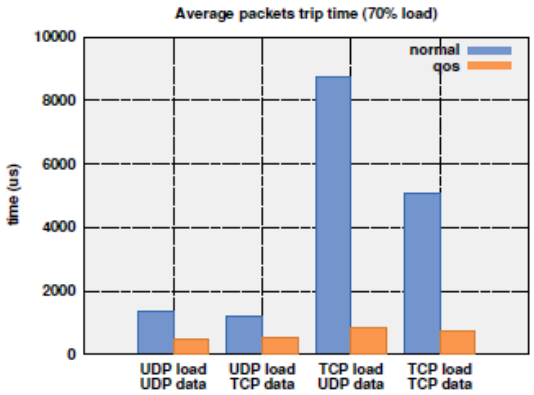
With priority (70%)

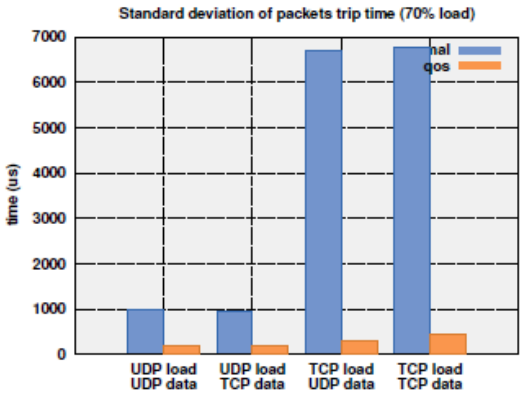


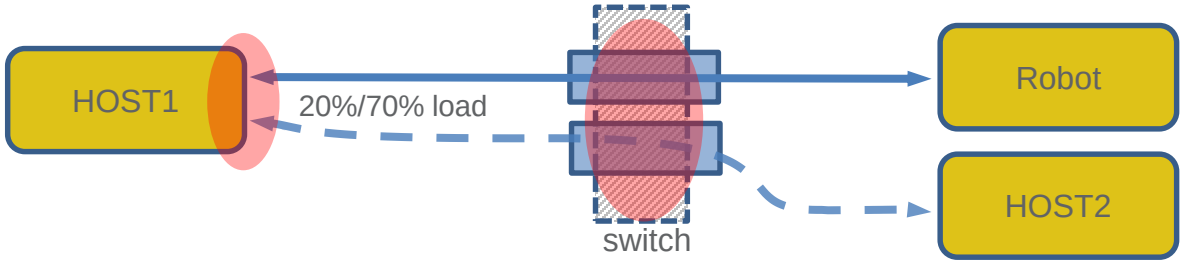

HOST1
20%/70% load
Robot
switch

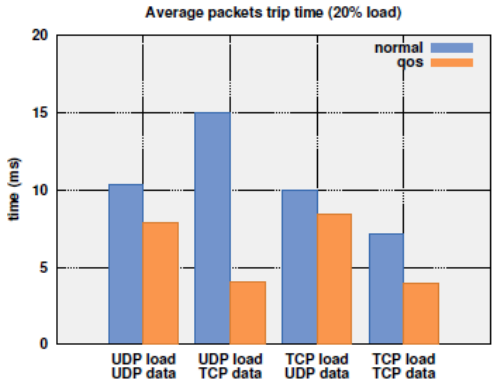

Average packets trip time (20% load)
time (us)
normal
qos
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data

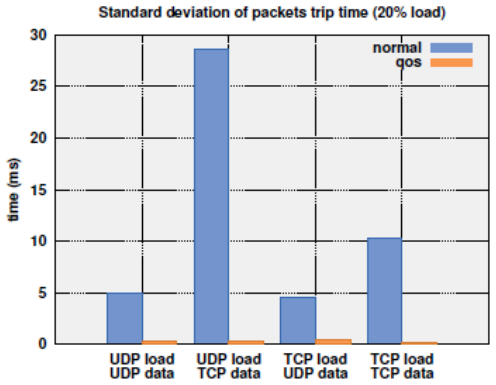

Standard deviation of packets trip time (20% load)
time (us)
normal
qos
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data

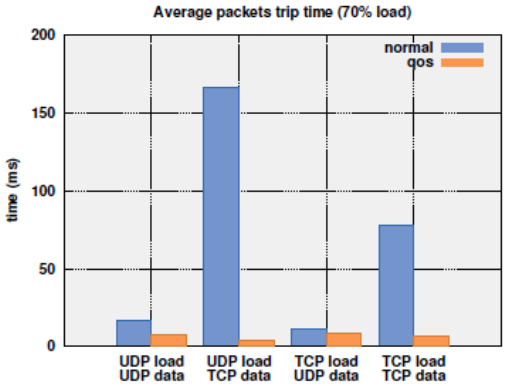

Average packets trip time (70% load)
time (us)
normal
qos
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data

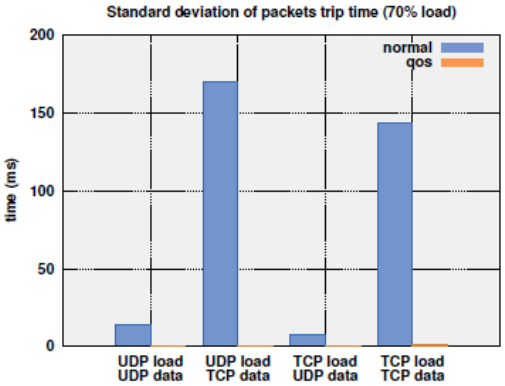

Standard deviation of packets trip time (70% load)
time (us)
qos
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data


HOST1
20%/70% load
Robot
HOST2
switch


Average packets trip time (20% load)
normal
qos
time (ms)
0
5
10
15
20
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data


Standard deviation of packets trip time (20% load)
normal
qos
time (ms)
0
5
10
15
20
25
30
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data


Average packets trip time (70% load)
normal
qos
time (ms)
0
50
100
150
200
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data


Standard deviation of packets trip time (70% load)
normal
qos
time (ms)
0
50
100
150
200
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data

YARP/ROS comparison

YARP

- Run-time reconfiguration of connections
- Pluggable protocols and devices
- Multicast for efficient one-to-many communication
- Multi-platform
- QoS, channel prioritization
- LGPL/GPL
- Smaller community
- No packet management

ROS

- Strongly typed
- Rich set of libraries and tools
- Eco-system, active community
- Packet management
- BSD license
- Ubuntu based
- Restricted set of protocols
- All connections from a topic use the same protocol

References

- Ali Paikan, Silvio Traversaro, Francesco Nori and Lorenzo Natale, "A Generic Testing Framework for Test Driven Development of Robotic Systems", In Lecture Notes in Computer Science (MESAS15), Springer, pp. , 2015
- Paikan, A., Domenichelli, D., and Natale, L., *Communication channel prioritization in a publish-subscribe architecture*, in Proc. Software Engineering and Architectures for Realtime Interactive Systems Workshop, Arles, France, 2015.
- Paikan, A., Tikhanoff, V., Metta, G., and Natale, L., *Enhancing software module reusability using port plug-ins: an experiment with the iCub robot*, in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, Illinois, 2014.
- Fitzpatrick, P., Ceseracciu, E., Domenichelli, D., Paikan, A., Metta, G., and Natale, L., *A middle way for robotics middleware*, Journal of Software Engineering for Robotics, vol. 5, no. 2, pp. 42-49, 2014.
- Paikan, A., Fitzpatrick, P., Metta, G., and Natale, L., *Data Flow Port's Monitoring and Arbitration*, Journal of Software Engineering for Robotics, vol. 5, no. 1, pp. 80-88, 2014
- Metta, G., Fitzpatrick, P., Natale, L., *YARP: Yet Another Robot Platform*, International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics, Volume 3, Issue 1, pp. 43-48, March 2006
- Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 2 (June 2003), 114-131.
- David, B. Stewart, Richard, A. Volpe, Pradeep, K. Khosla, Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects, IEEE Trans. On Software Engineering, 23(12), 1997
- Dušan Bálek, František Plášil, Software Connectors and Their Role in Component Deployment, New Developments in Distributed Applications and Interoperable Systems, 70, 2002, pp. 69-84.
- Farhad Arbab: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3):329--366, 2004.
- D. Brugali, Model-Driven Software Engineering in Robotics, IEEE Robotics and automation magazine, Sept. 2015