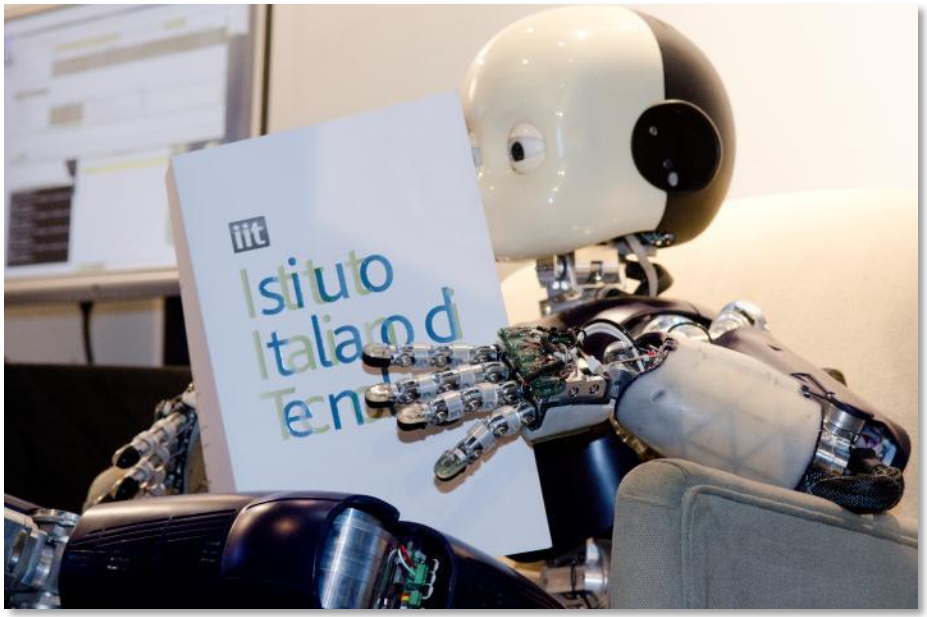
ISTITUTO ITALIANO
DI TECNOLOGIA

The YARP middleware

Lorenzo Natale

iCub Facility
Istituto Italiano di Tecnologia, Genova



Pisa, June 2015

Program

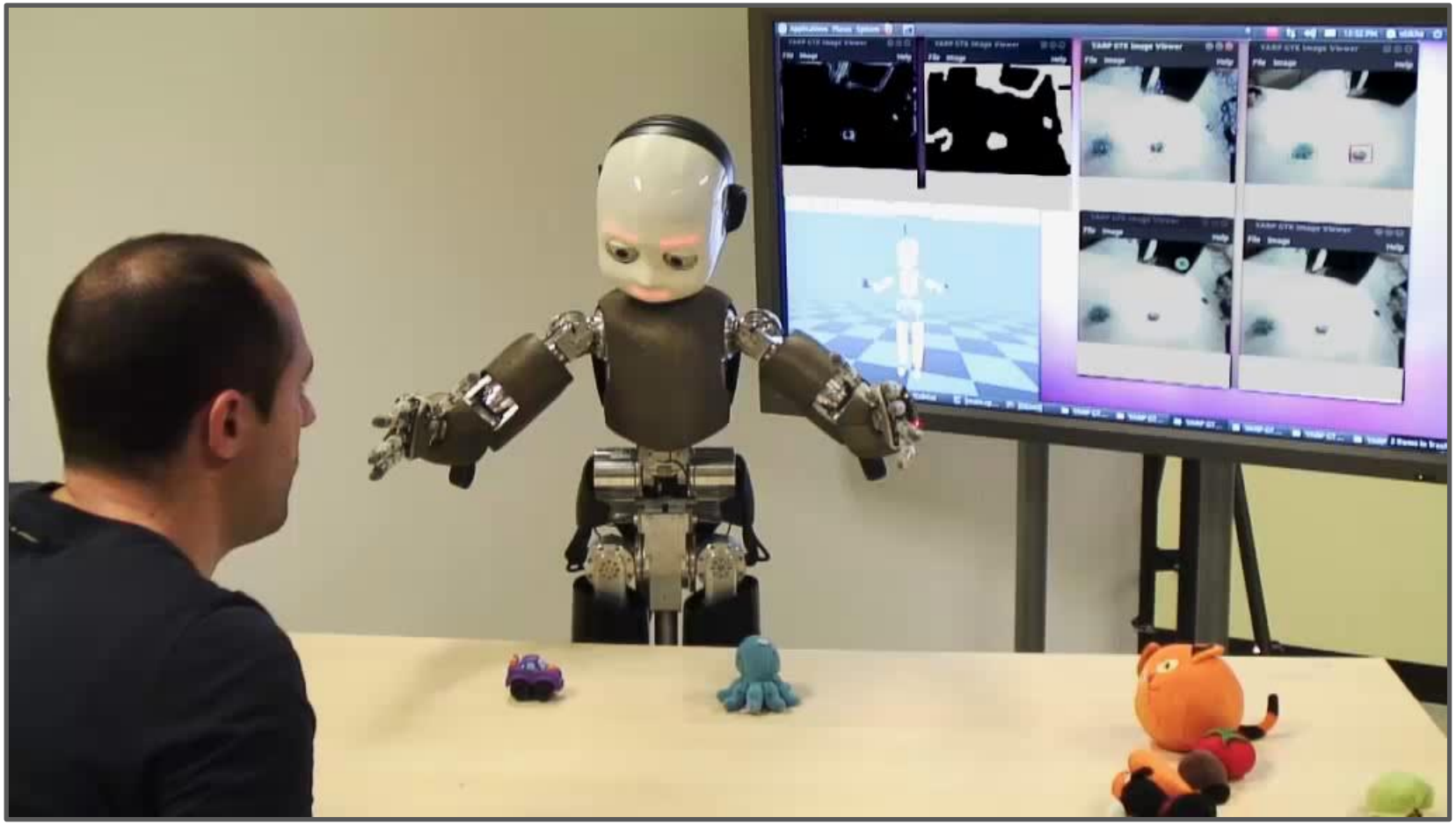
- **June 9 – h 15:00-17:00** Overview of YARP and introduction to the libraries. Command line tools, Hello world.
- **June 11 – h 15:00-18:00** Ports and Modules in YARP, threading. Example of image processing.
- **June 18 – h 15:00-18:00** Motor control. Example of control of the iCub simulator, visual tracking.

References

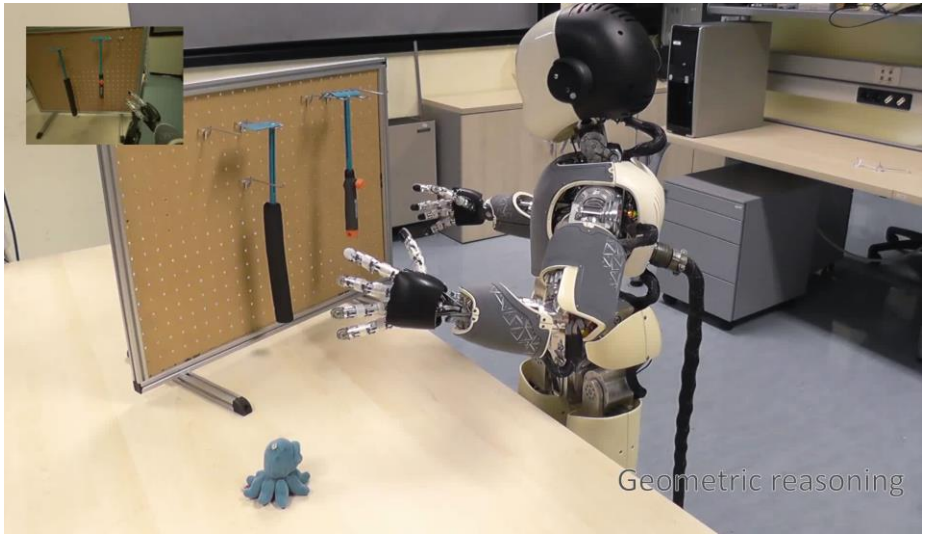
- YARP: www.yarp.it
  - Documentation
  - Papers (see also references in this presentation)
- iCub: http://wiki.icub.org/wiki/Manual
- CMake:
  - www.cmake.org → documentation & wiki
  - Mastering CMake, by Kitware Inc.
- C++:
  - Thinking in C++, Bruce Eckel
  - The C++ programming language, Bijarne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

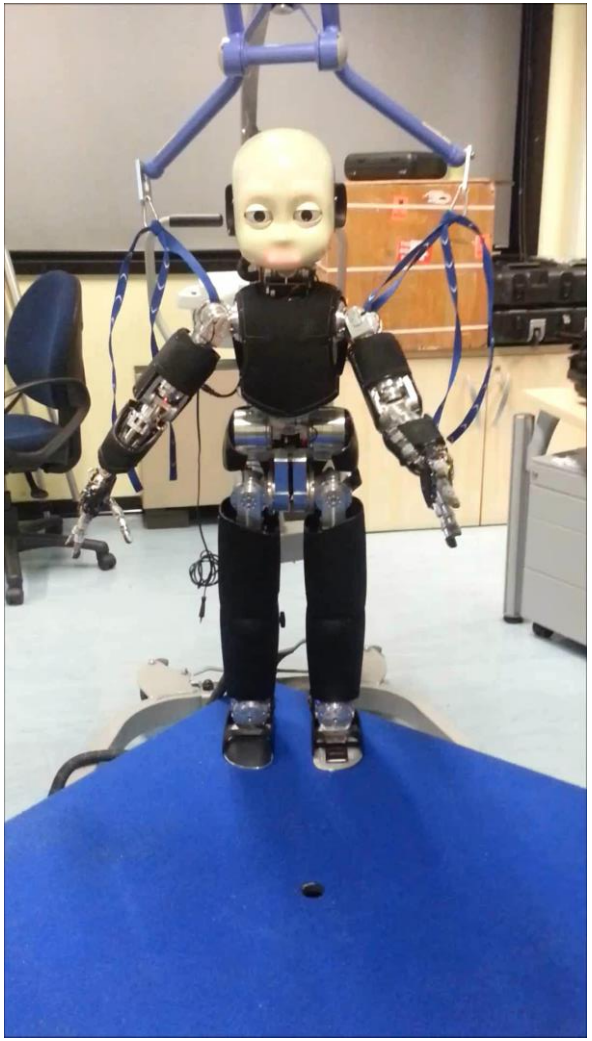
Motivations

Complex behaviors

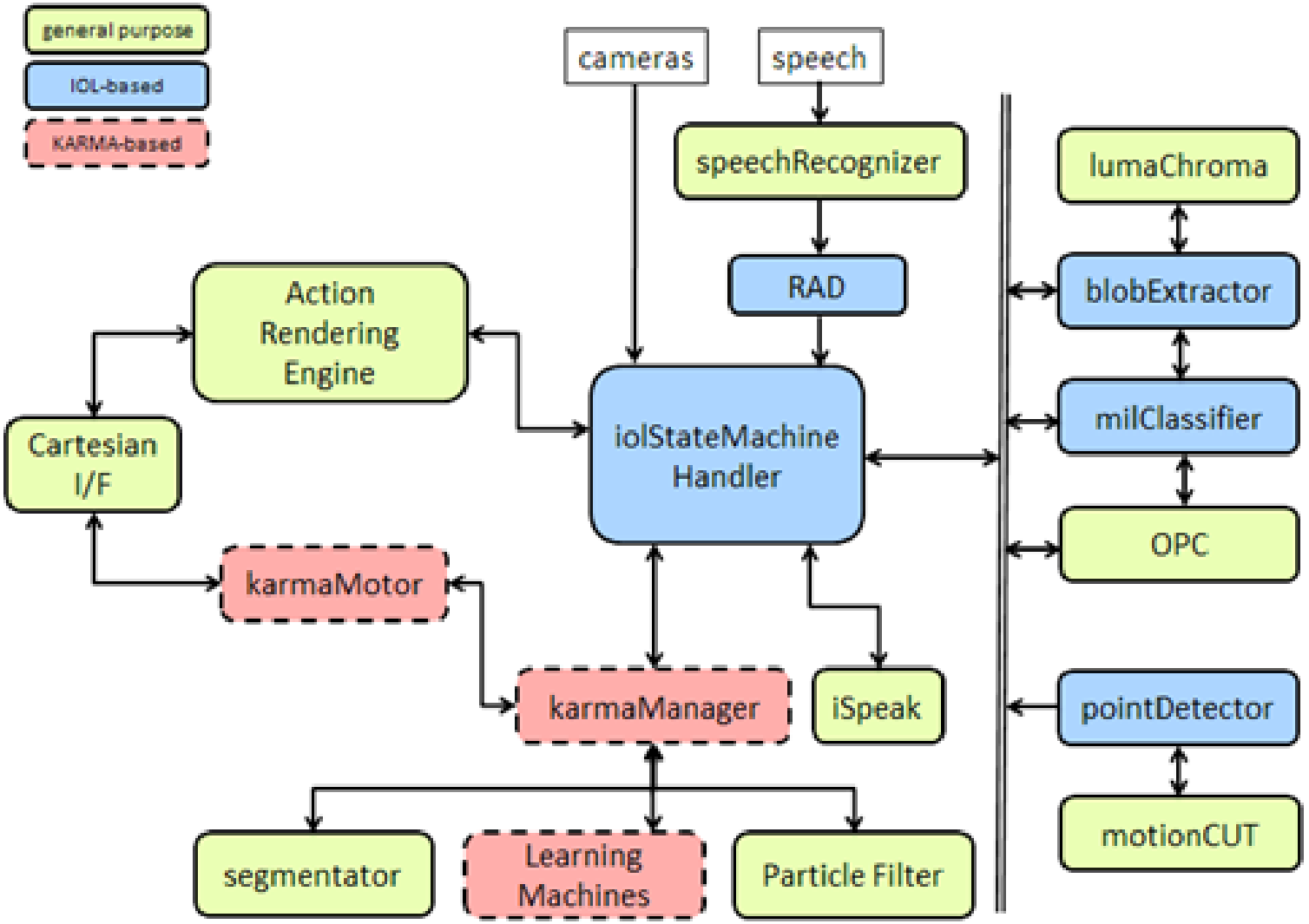


Integration of software components



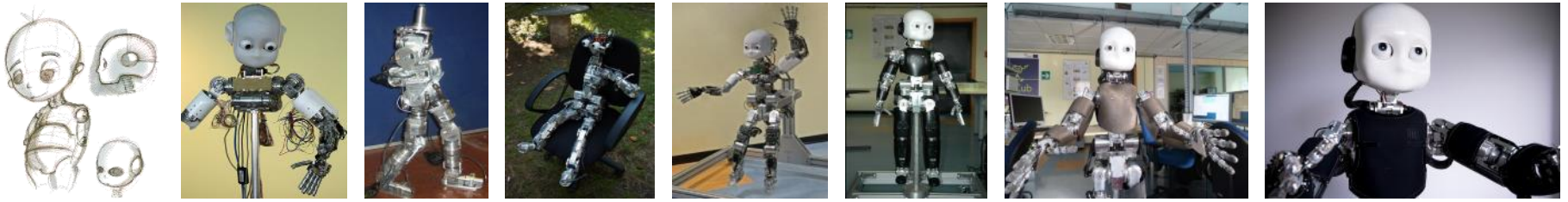


Complex behaviors



Key issues

- Asynchronous development
- Variability: various scenarios and platforms
- Lack of standards
- Inherent complexity, distributed processing, lots of sensors, real-time
- 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

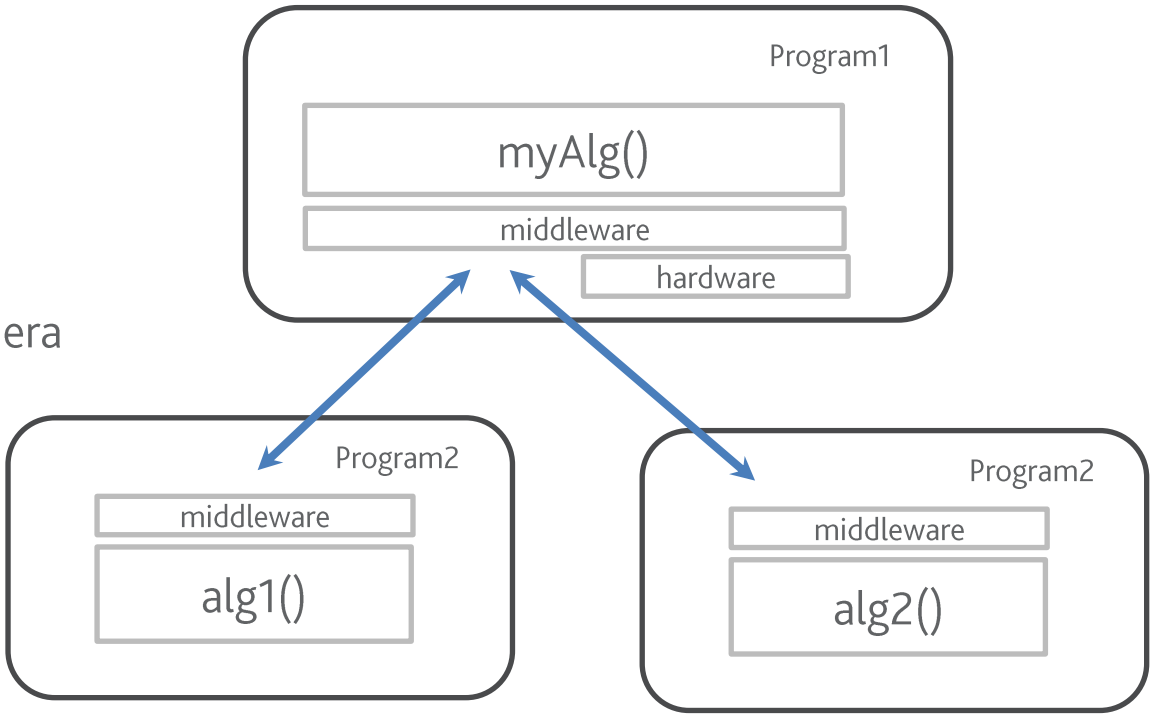
Application dependent

Component driven software development

```
output myAlgorithm(input)
{
 ...
 out = call alg1(in)
 …
 out = call alg2(in)
 getImage() // from usb camera
 …
}

output alg1(input)
{code}

output alg2(input)
{code}
```

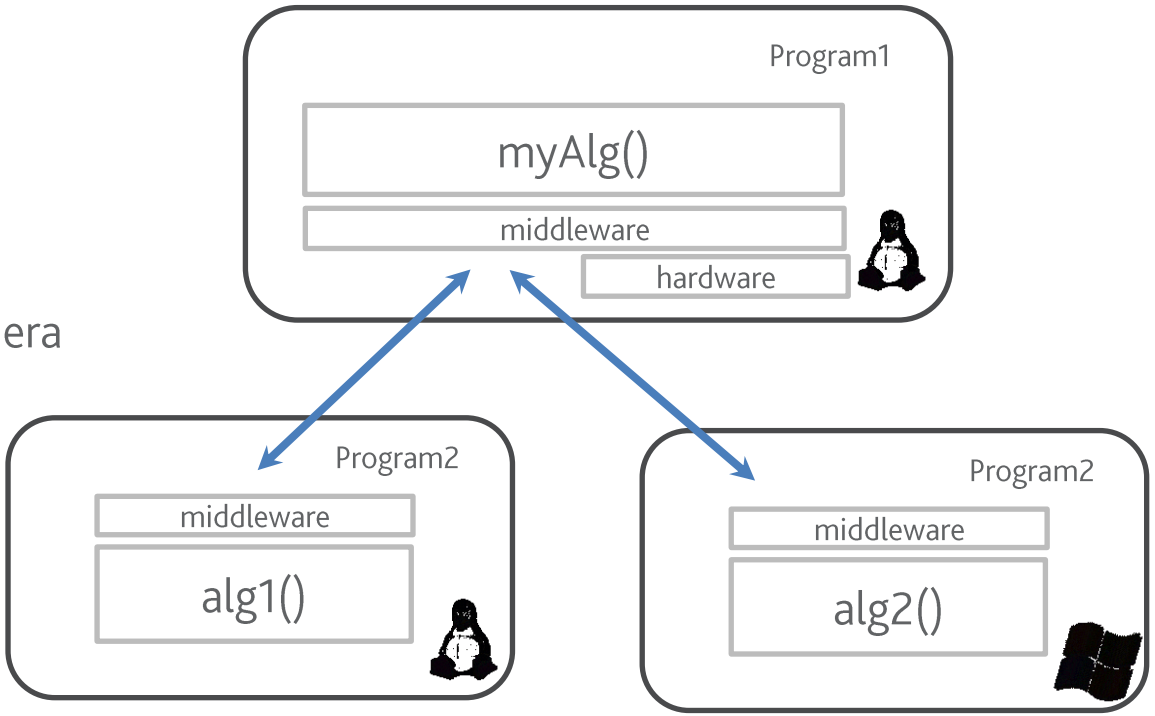
Component driven software development

```
output myAlgorithm(input)
{
 ...
 out = call alg1(in)
 ...
 out = call alg2(in)
 getImage() // from usb camera
 ...
}

output alg1(input)
{code}

output alg2(input)
{code}
```



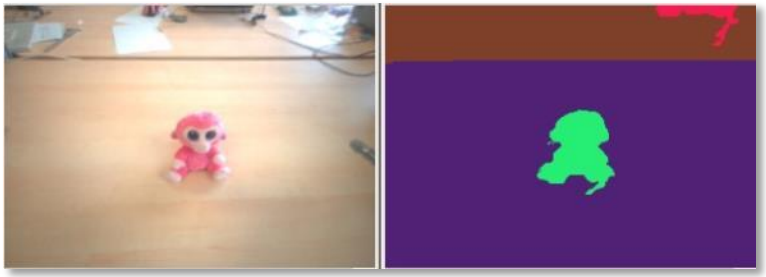
Component driven software development

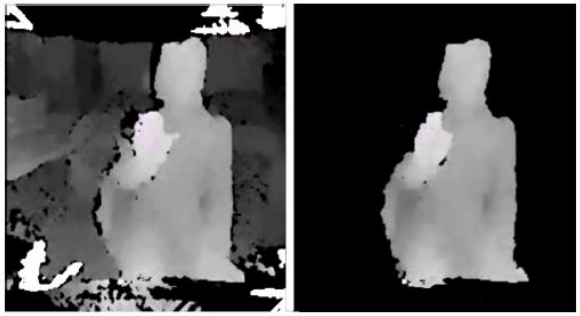
```
output myAlgorithm(input)
{
 ...
 out = call alg1(in)
 ...
 out = call alg2(in)
 getImage() // from usb camera
 ...
}

output alg1(input)
{code}

output alg2(input)
{code}
```

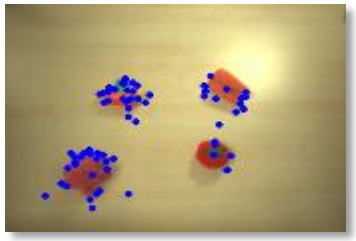


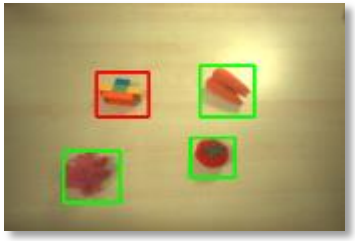
Components: some examples from the iCub repository

- 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)

Communication paradigm

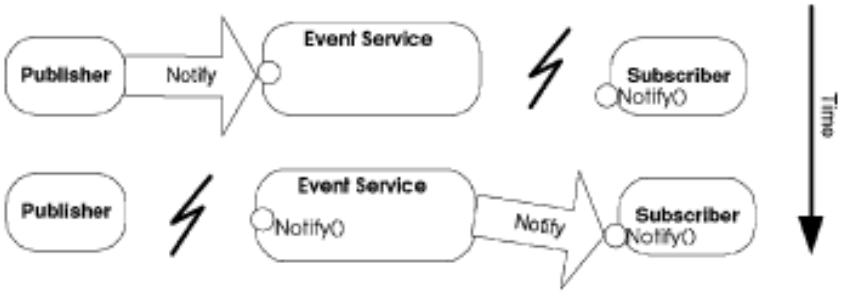
- Publish/Subscribe
  - Space, time, synchronization decoupling
- Remote Procedure Calls (RPC)
  - Remote invocation of object
  - Synchronous nature (although variant exists)

Publish/Subscribe

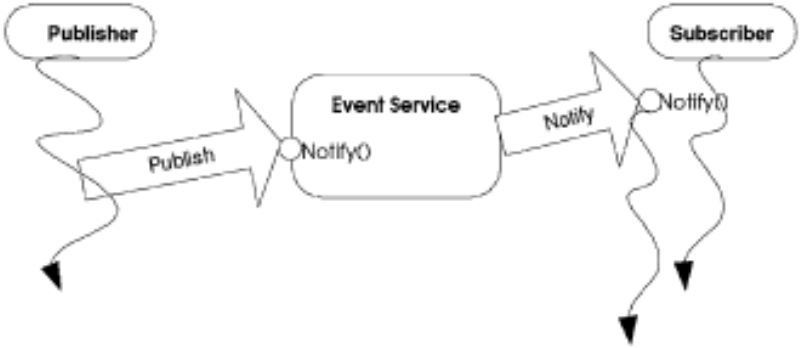
- **Space decoupling**: the interaction parties do not need to know each other; publisher publish events through an event service and the subscribers get these events indirectly through the event service
- **Time decoupling**: interaction parties do not need to be actively participating in the interaction 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

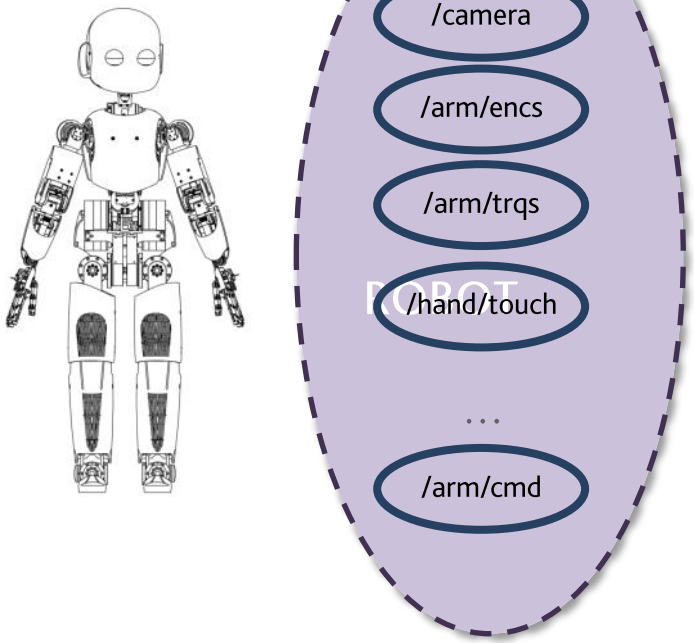
From: Eugster, et al., The Many Faces of Publish/Subscribe, 2003.

YARP approach

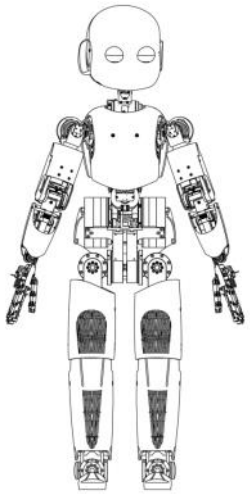
- Limited form of publish-subscribe
- *Observer patter*: subscribers register their interest directly with publishers, which mange subscriptions and sends events
- Publishers notify subscribers synchronously or asynchronously
- Space decoupling (from user perspective)
- Time coupling
- Dynamic (re)configuration

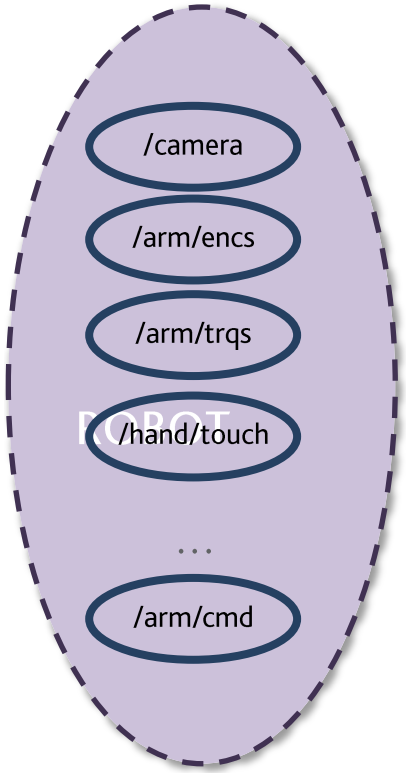
See also:
YARP: Yet Another Robot Platform, G. Metta, P. Fitzpatrick, L. Natale, 2006
Stewart et al., Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects, 1997

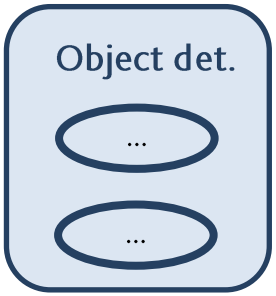
iCub software architecture

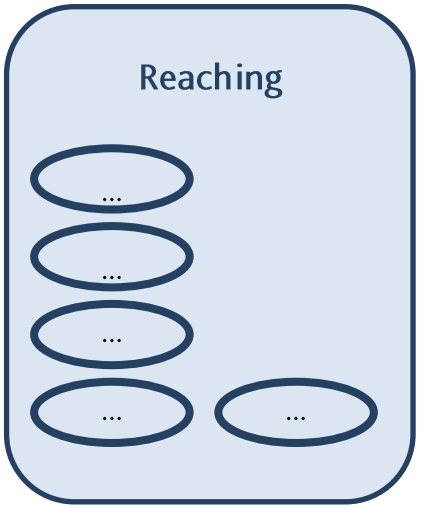


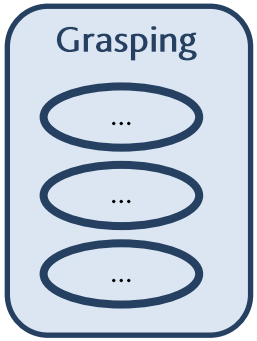
iCub software architecture



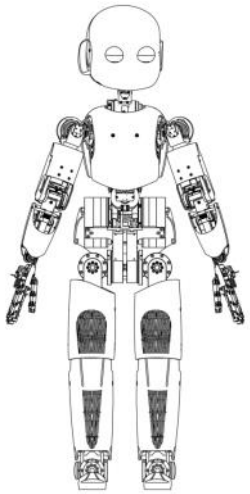


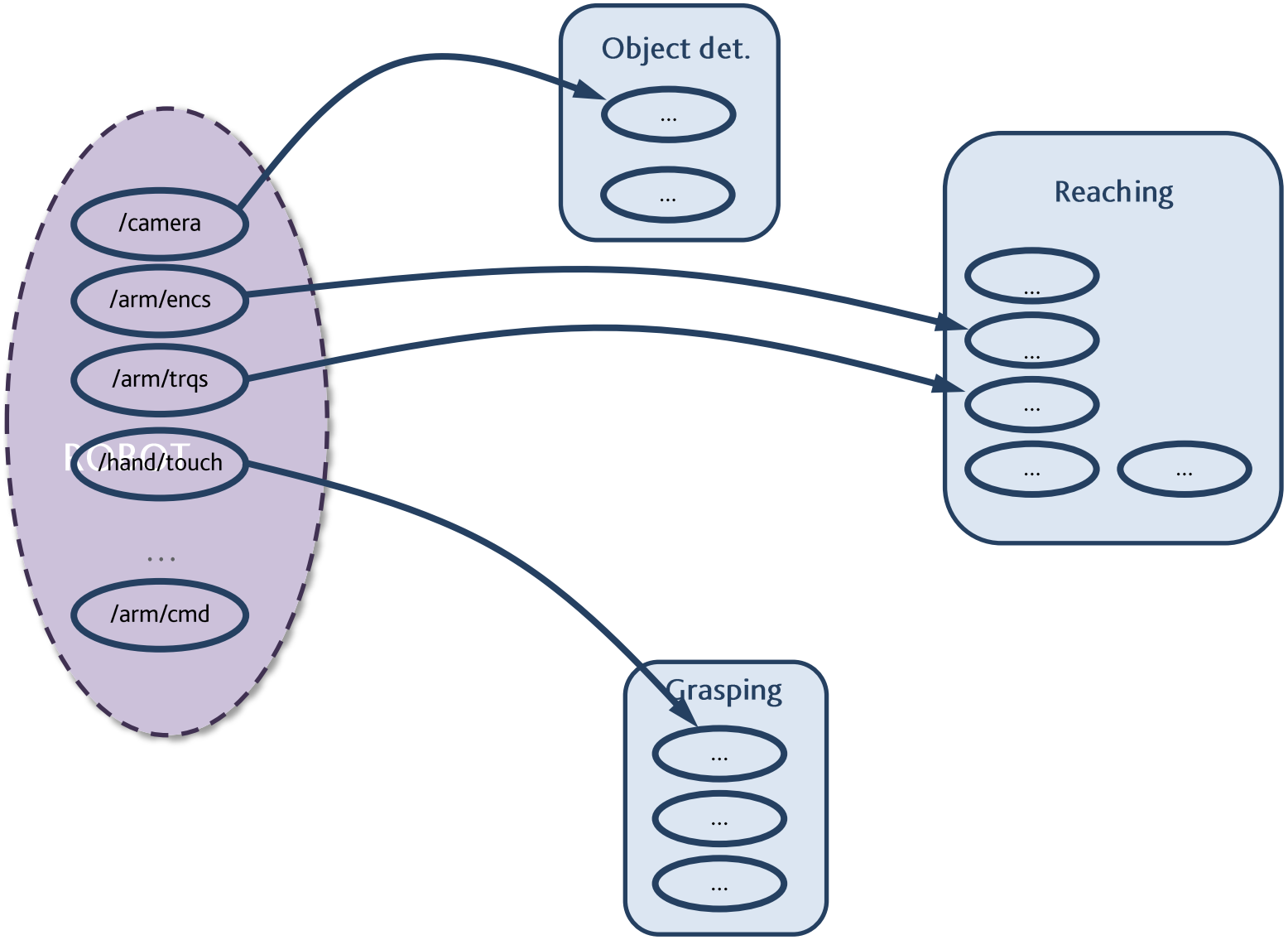




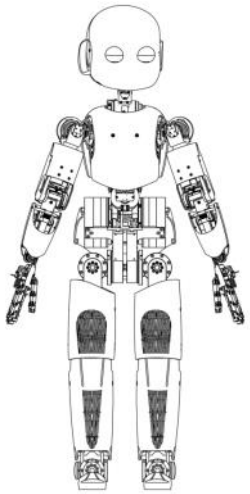


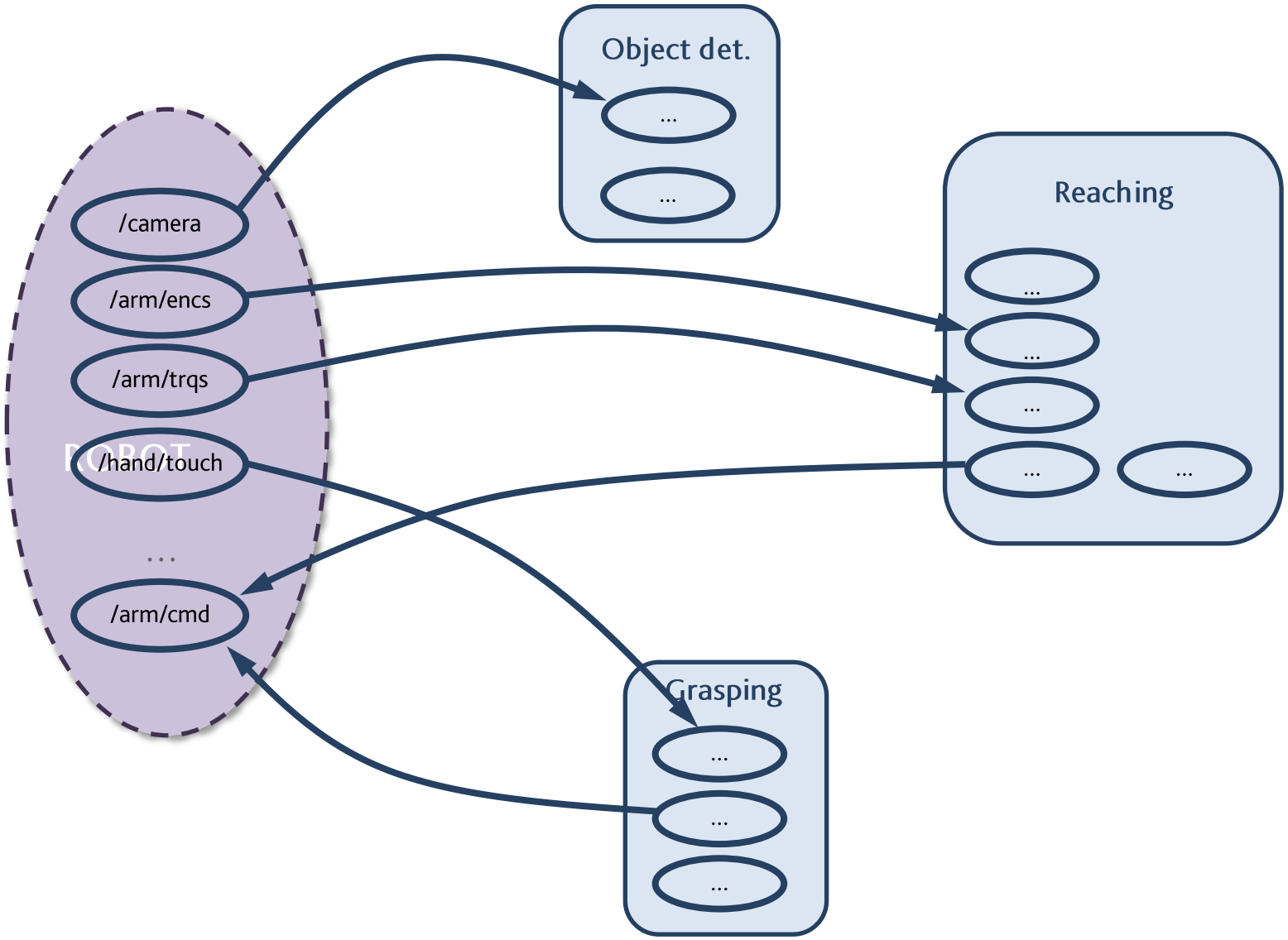
iCub software architecture



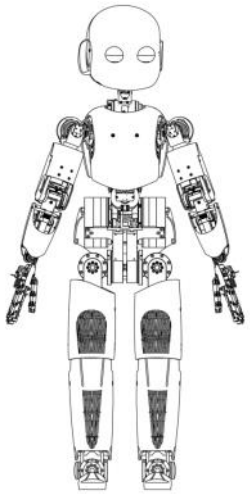


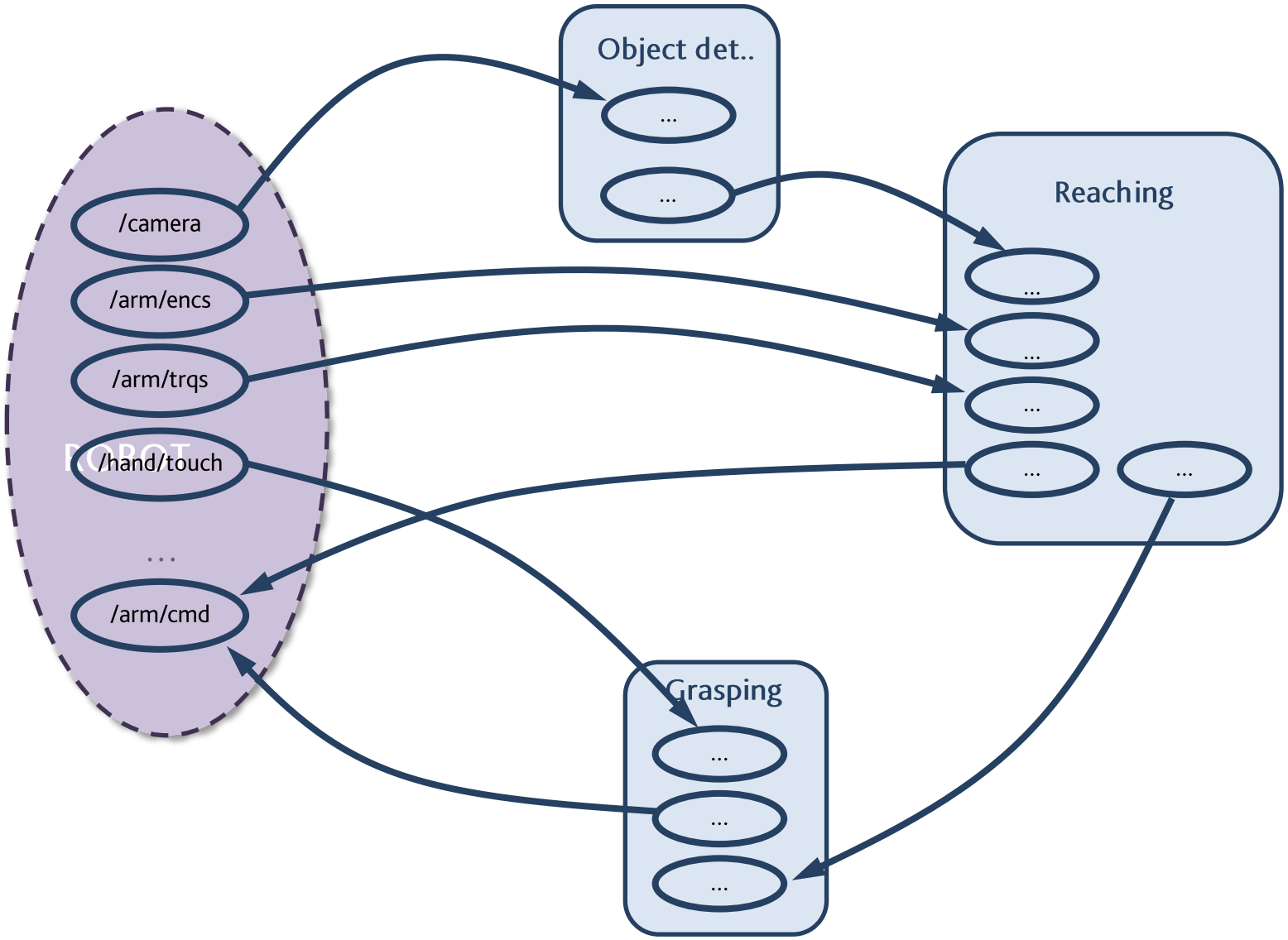
iCub software architecture



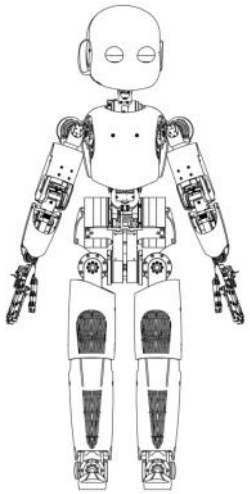


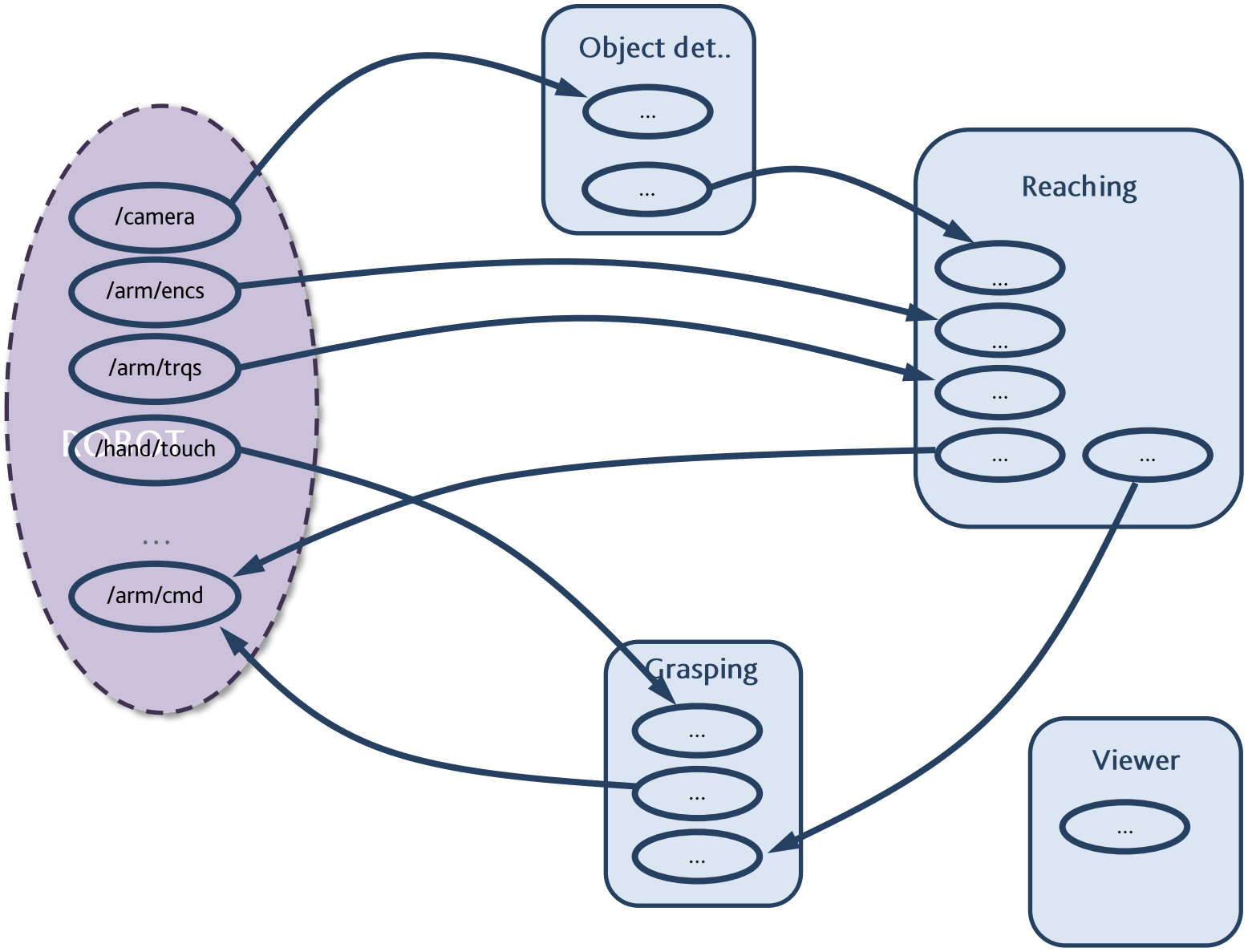
iCub software architecture



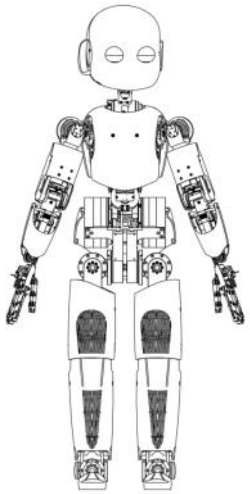


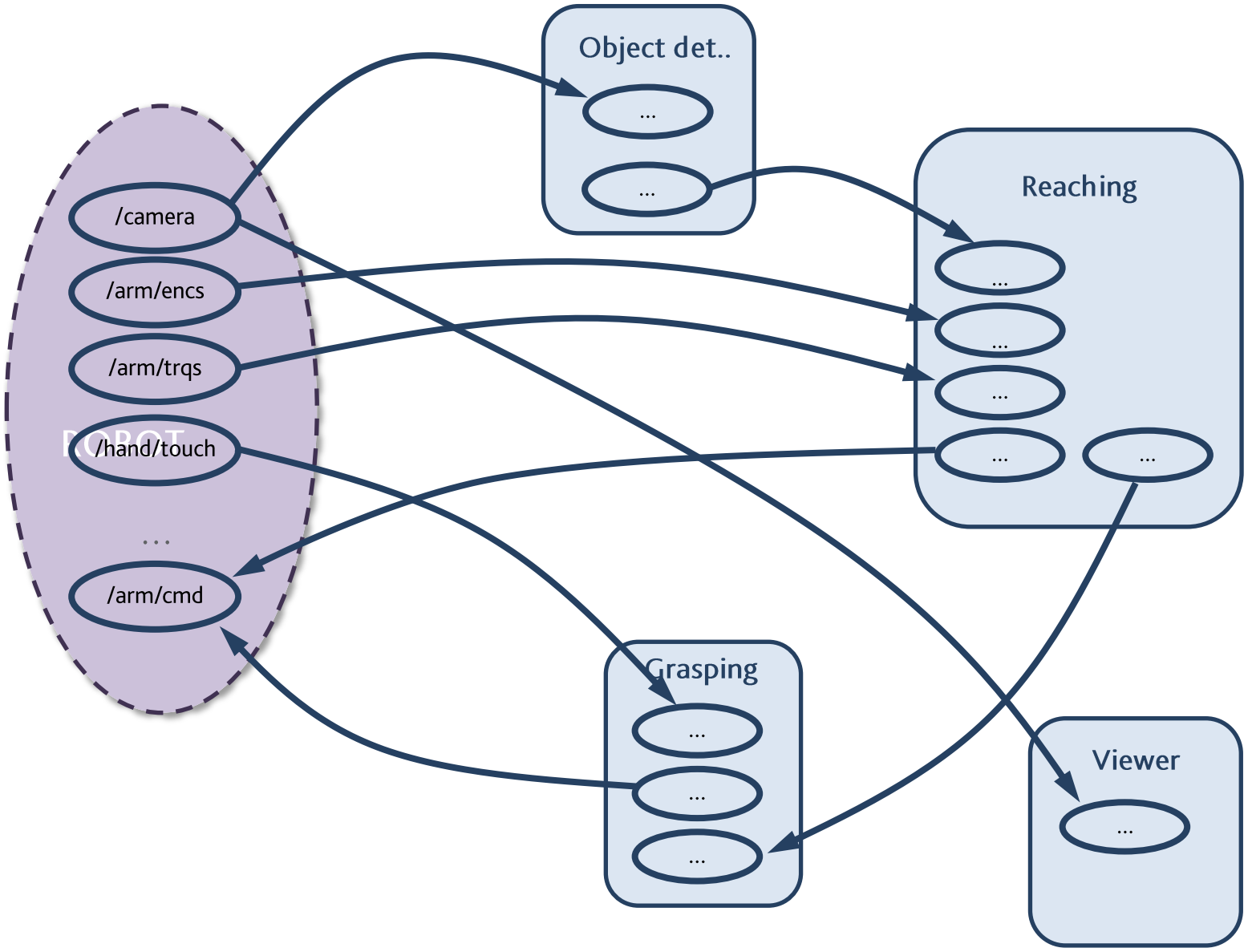
iCub software architecture



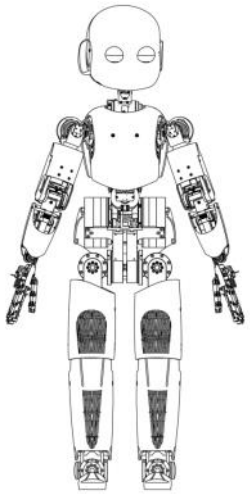


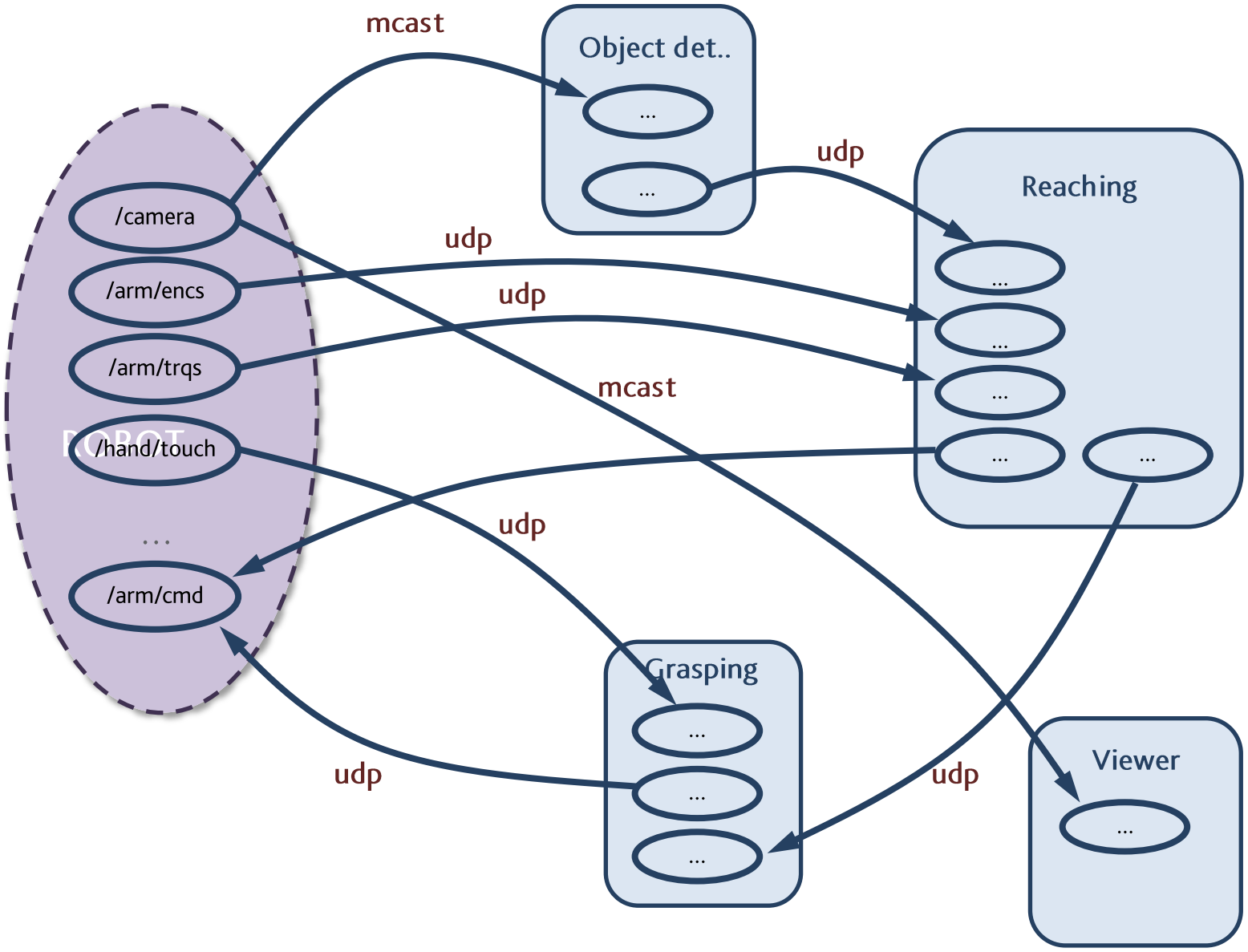
iCub software architecture



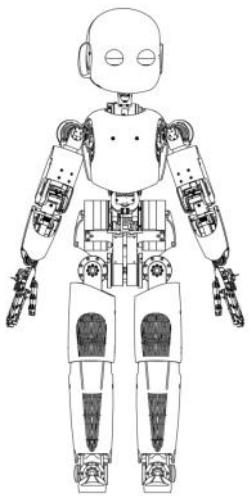


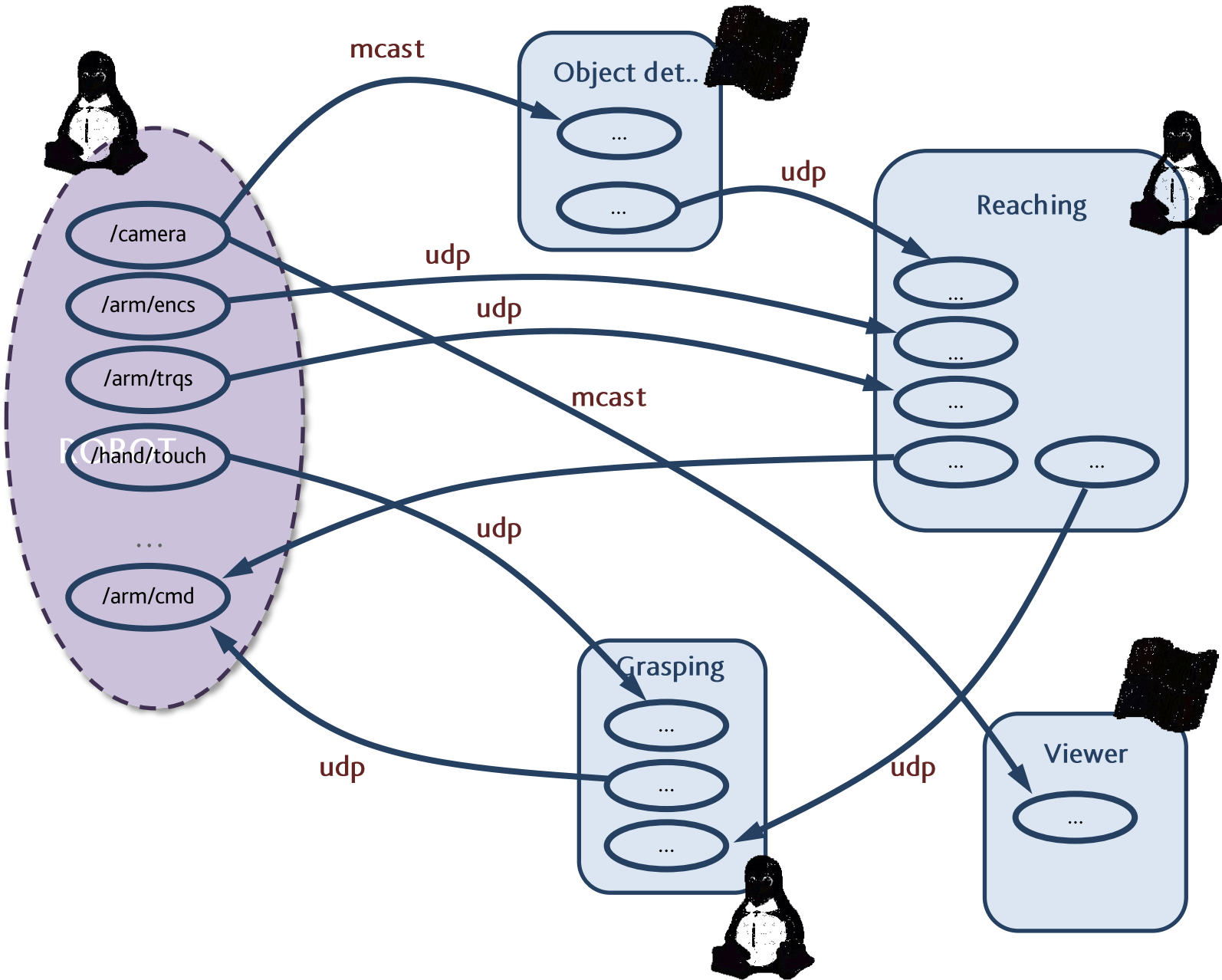
iCub software architecture





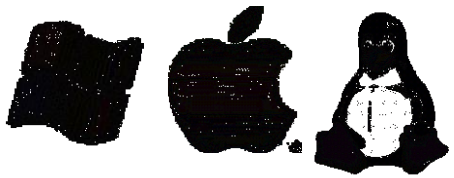
iCub software architecture





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..



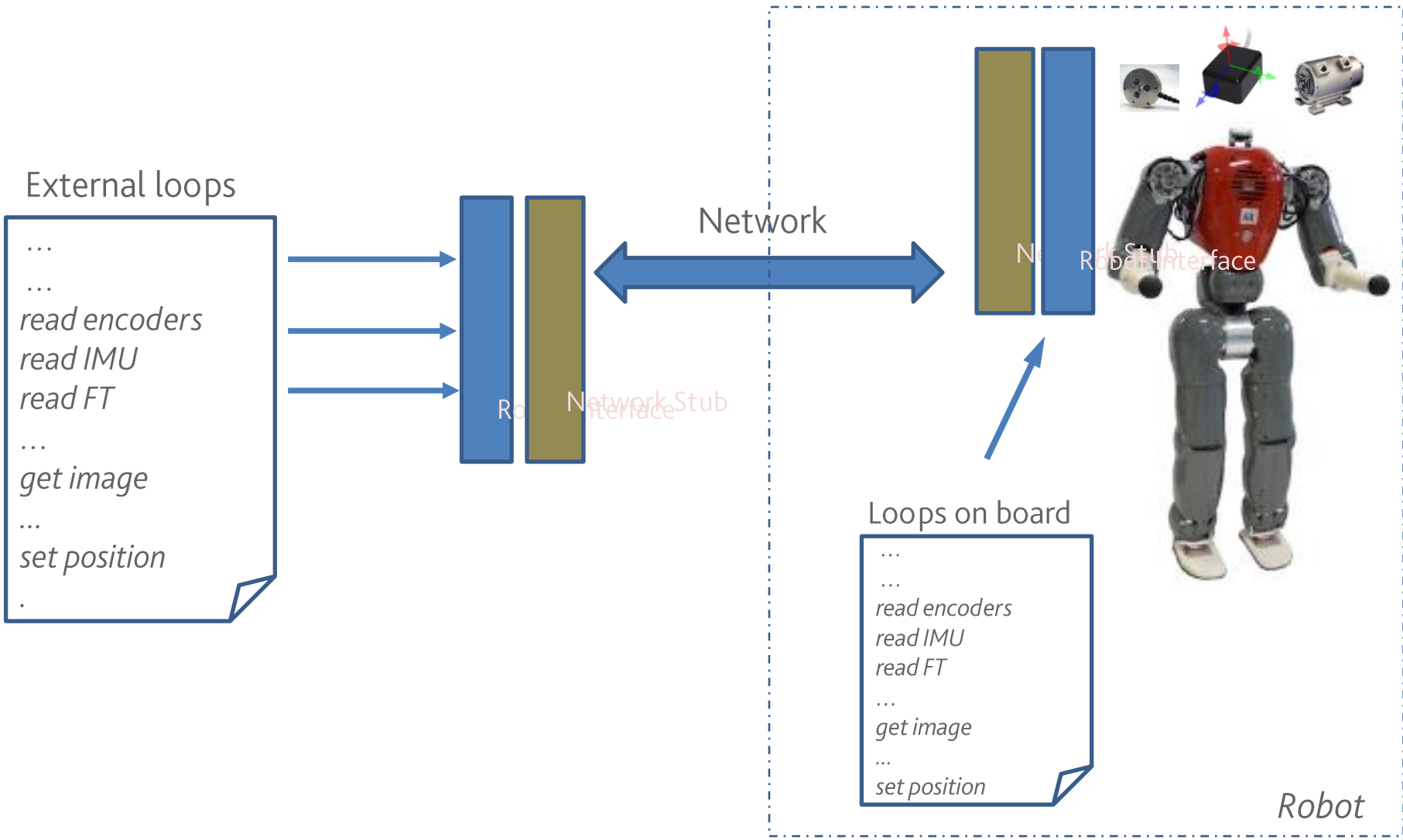
CMake
Cross-platform Make

SWIG

Interfaces

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

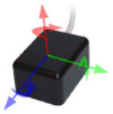
Interfaces


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

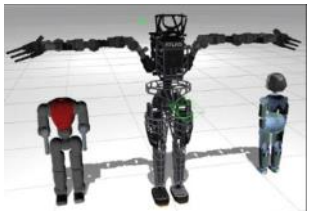
Interfaces

Control loops

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

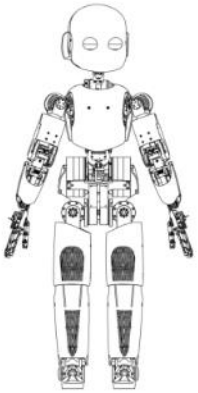


Robot Interface



Gazebo

COMAN



iCub

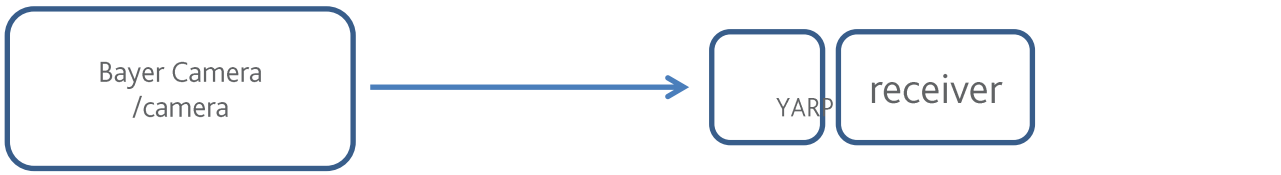
Walkman

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

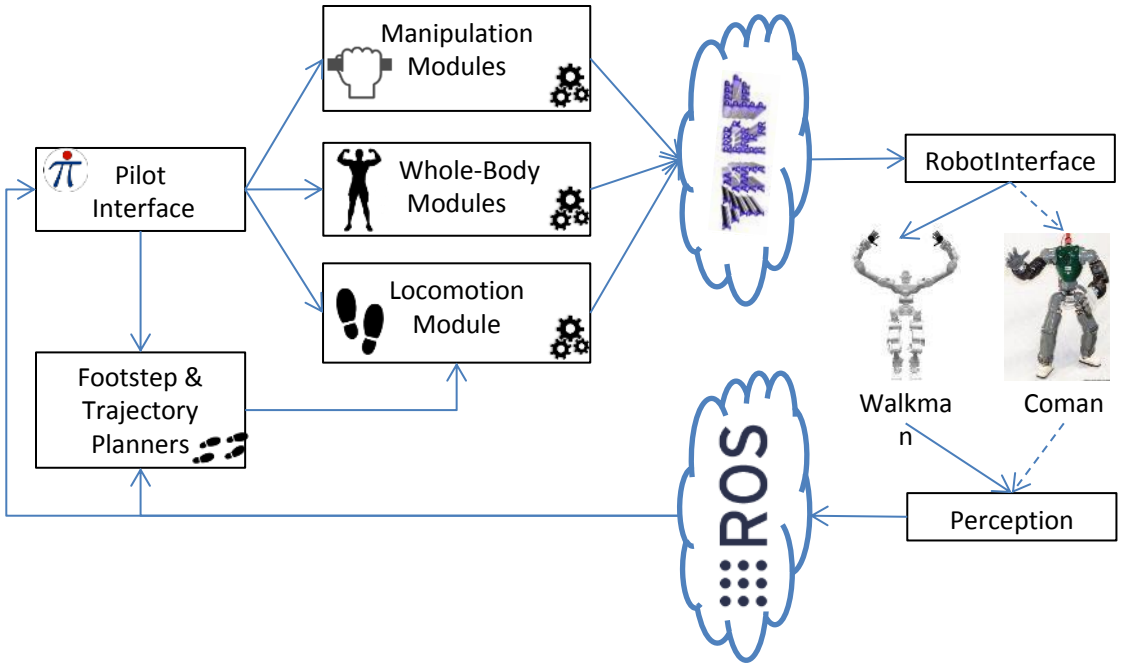
Examples

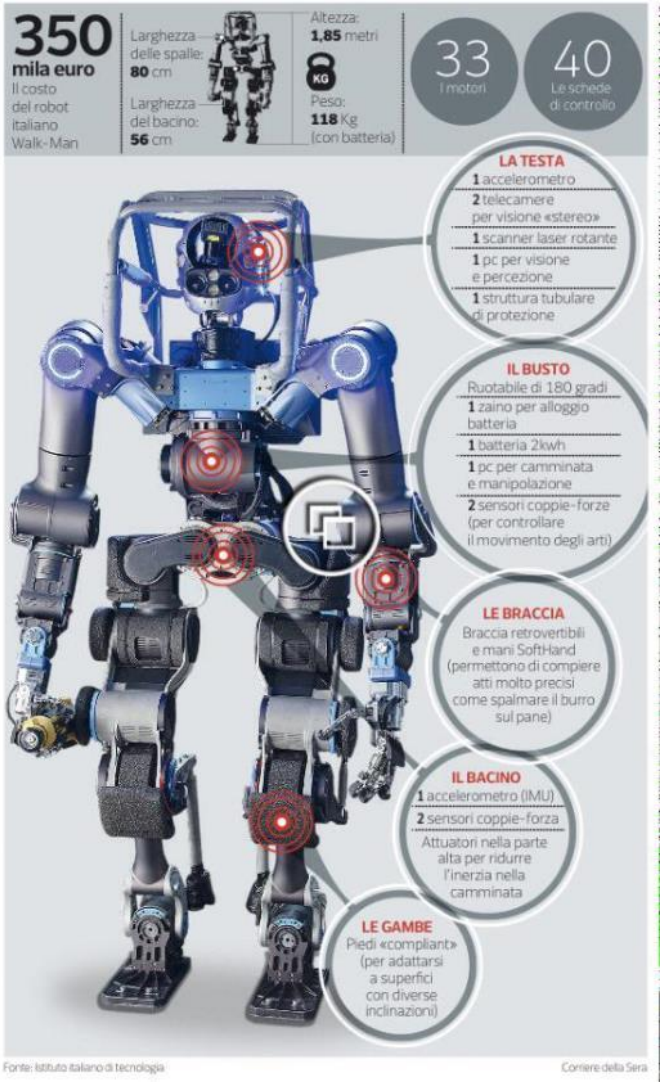
yarp connect /camera /receiver rec.bayer

yarp connect /65.52.88.202:5159 /receiver mjpeg

yarp connect /image@/camera /receiver

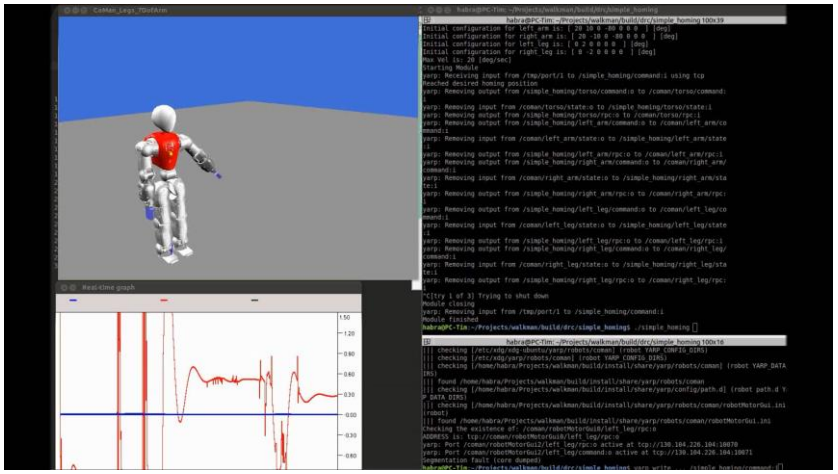
Success story



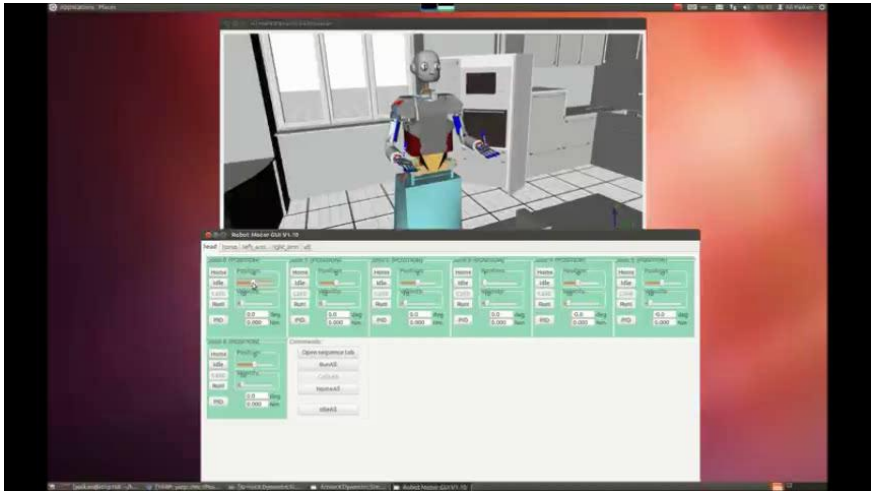


Simulators & robots

- 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

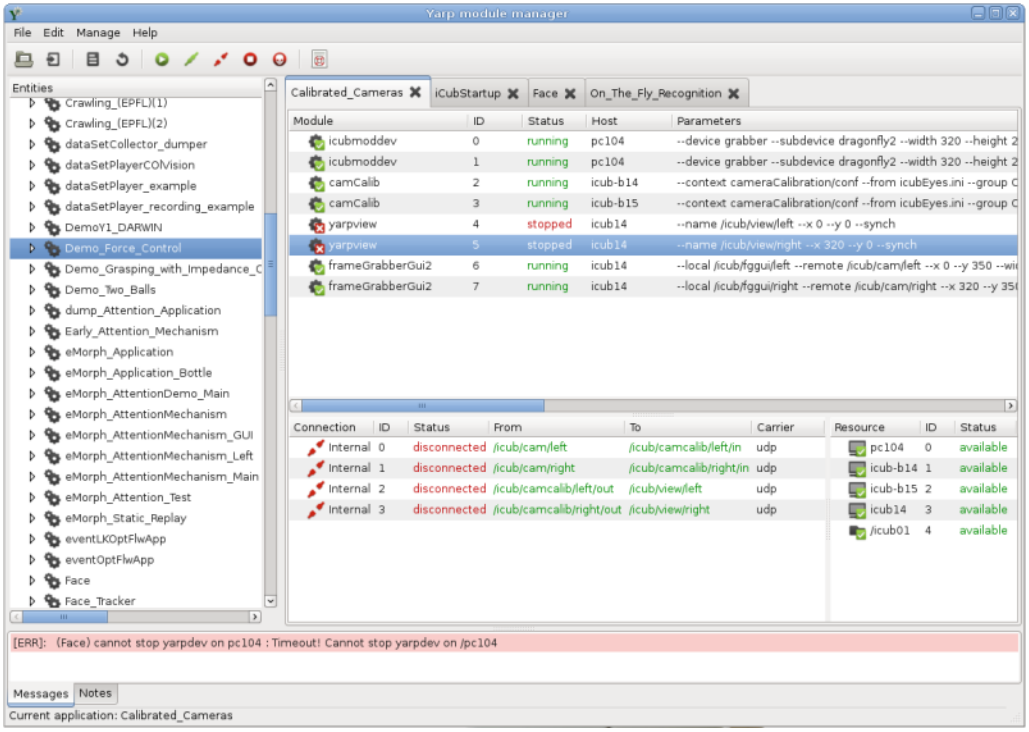
In a modular system integration becomes an issue:

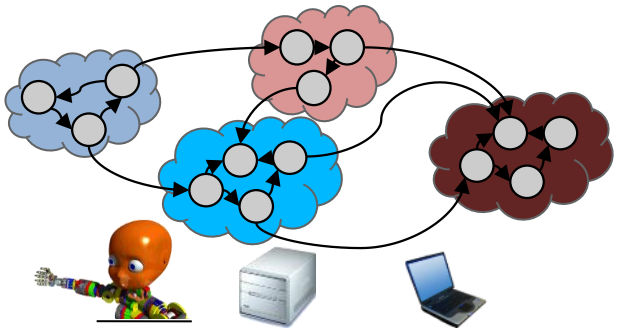
- Execution and monitoring
- Development
- Coordination

Execution and monitoring: YARP manager

Required modules connections nodes resources

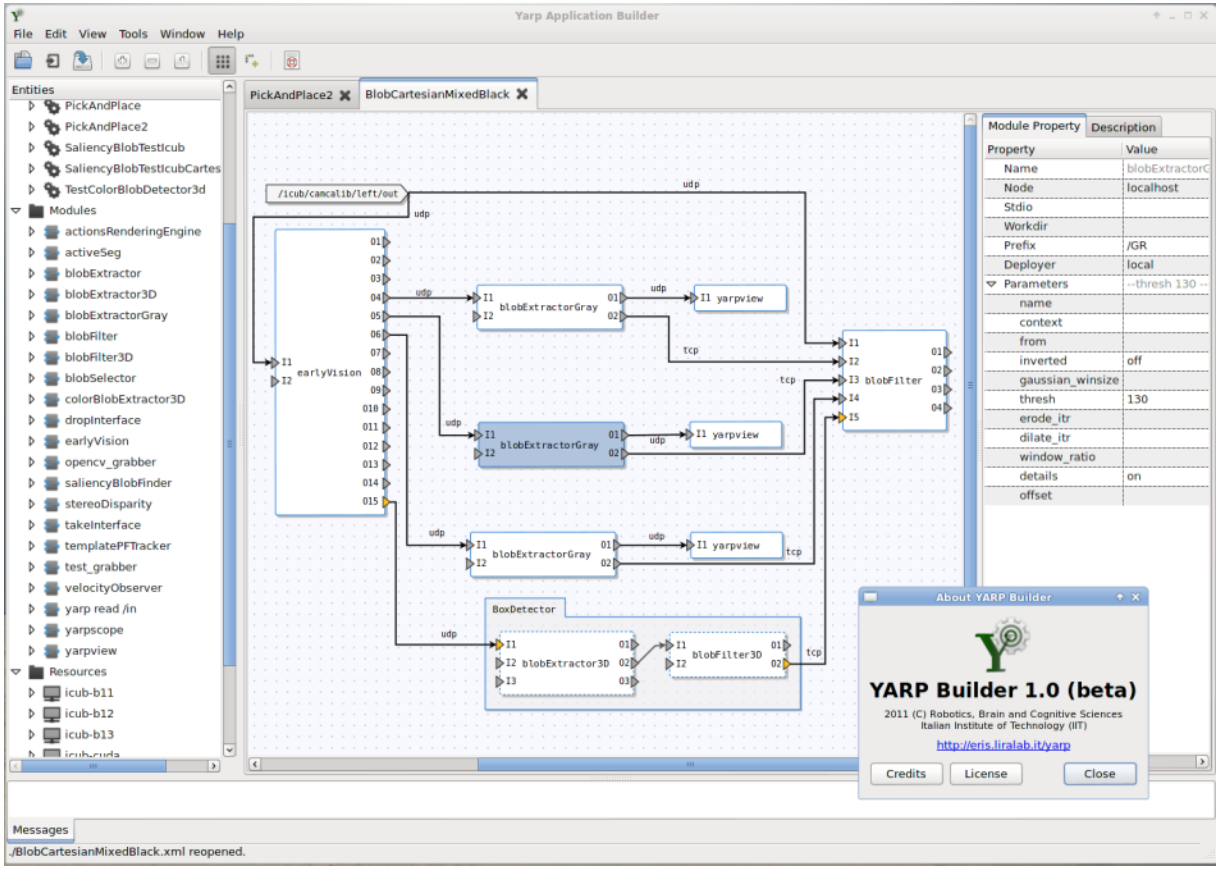
Available resources



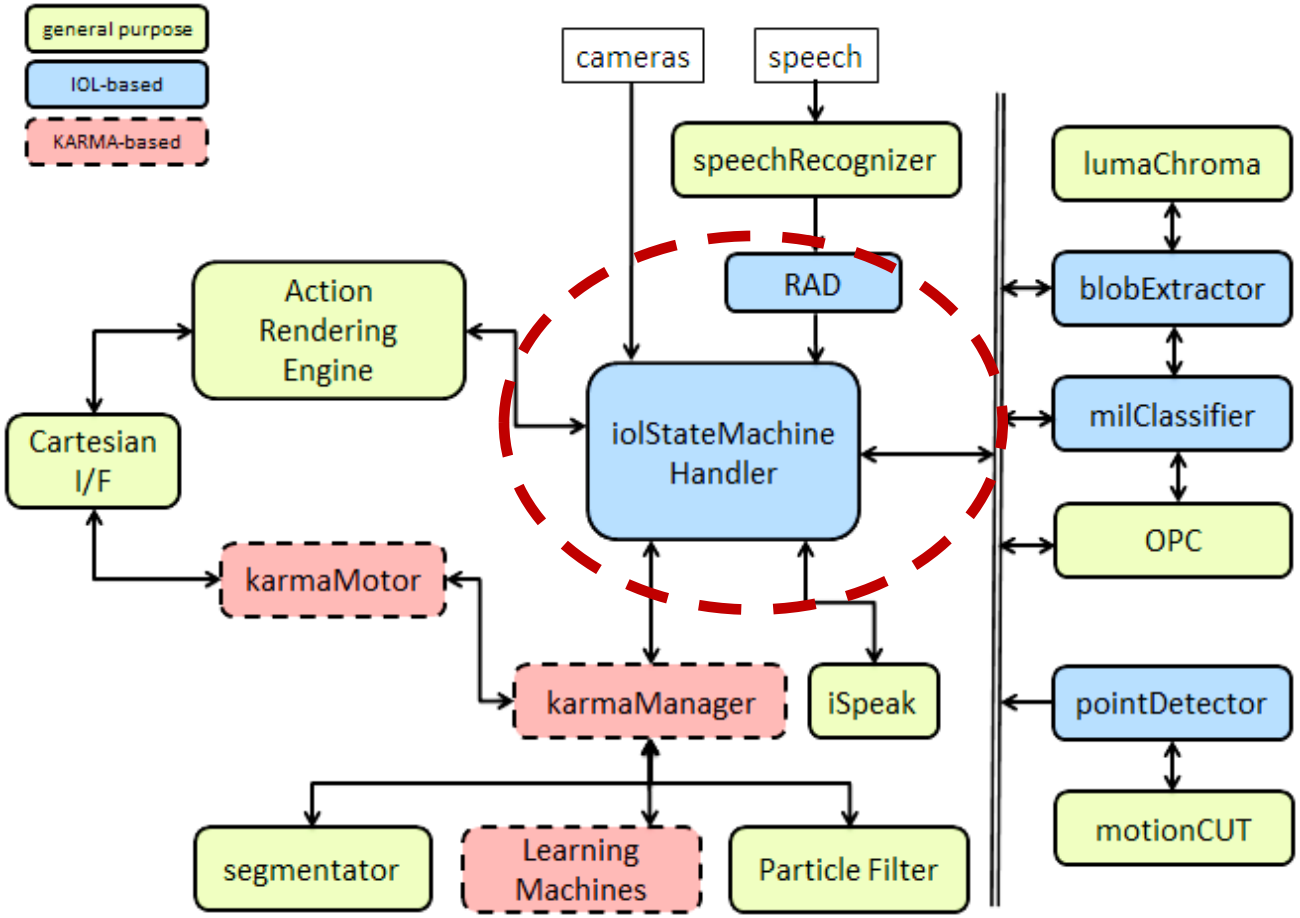


Tools for rapid development

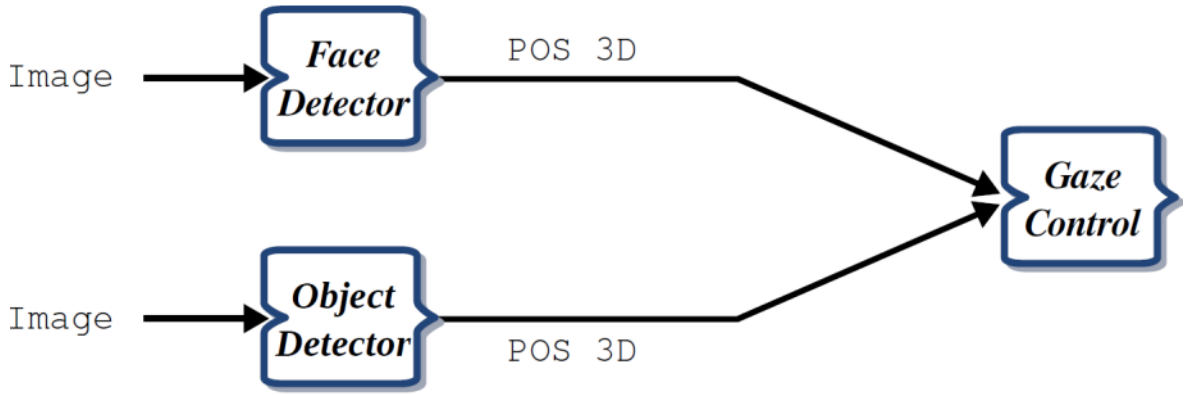
- YARP builder: graphical tool to design application
- Interface Definition Language (IDL):
  - formalization of types and interfaces between modules
  - automatic generation of message handlers



Coordinating modules

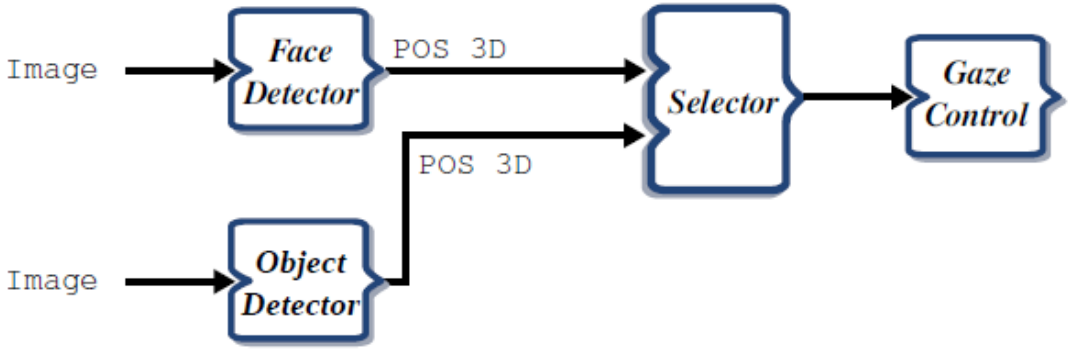


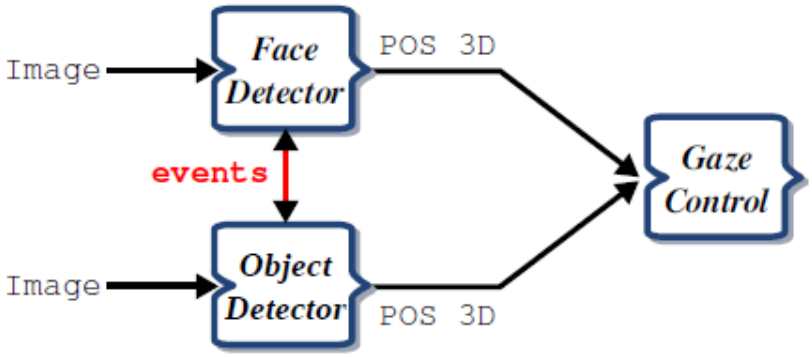
Arbitration and coordination

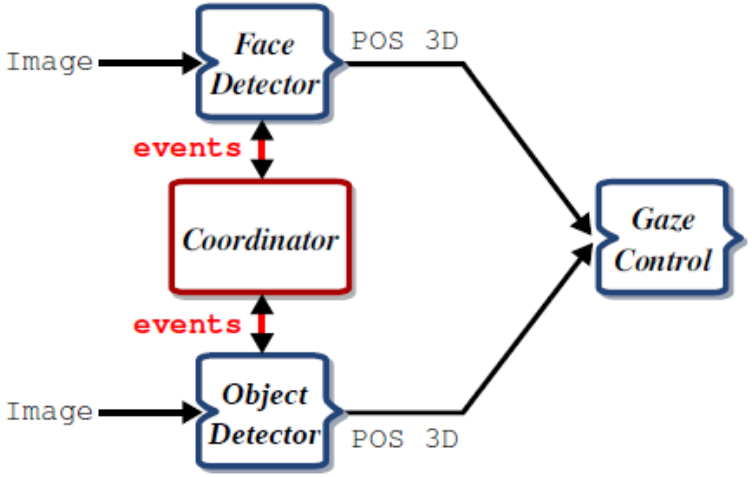


Paikan, A., et. al., *Data Flow Port's Monitoring and Arbitration*, Joser 2014.

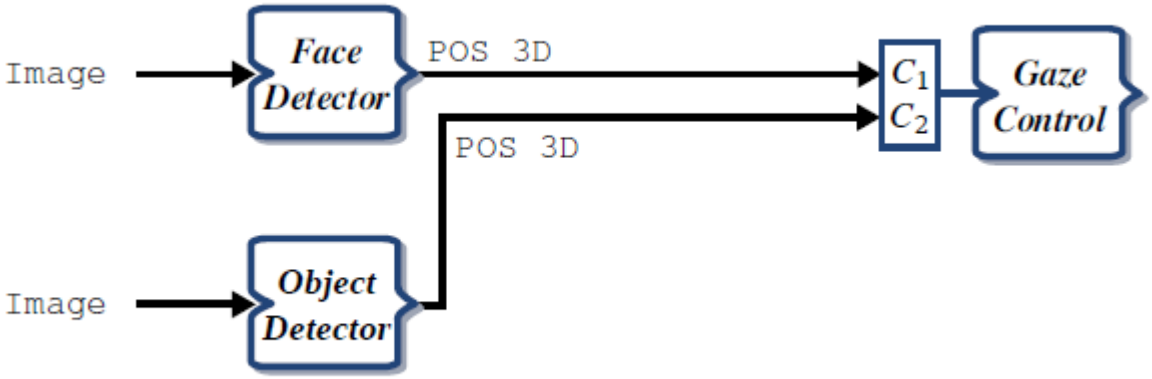
Arbitration and coordination



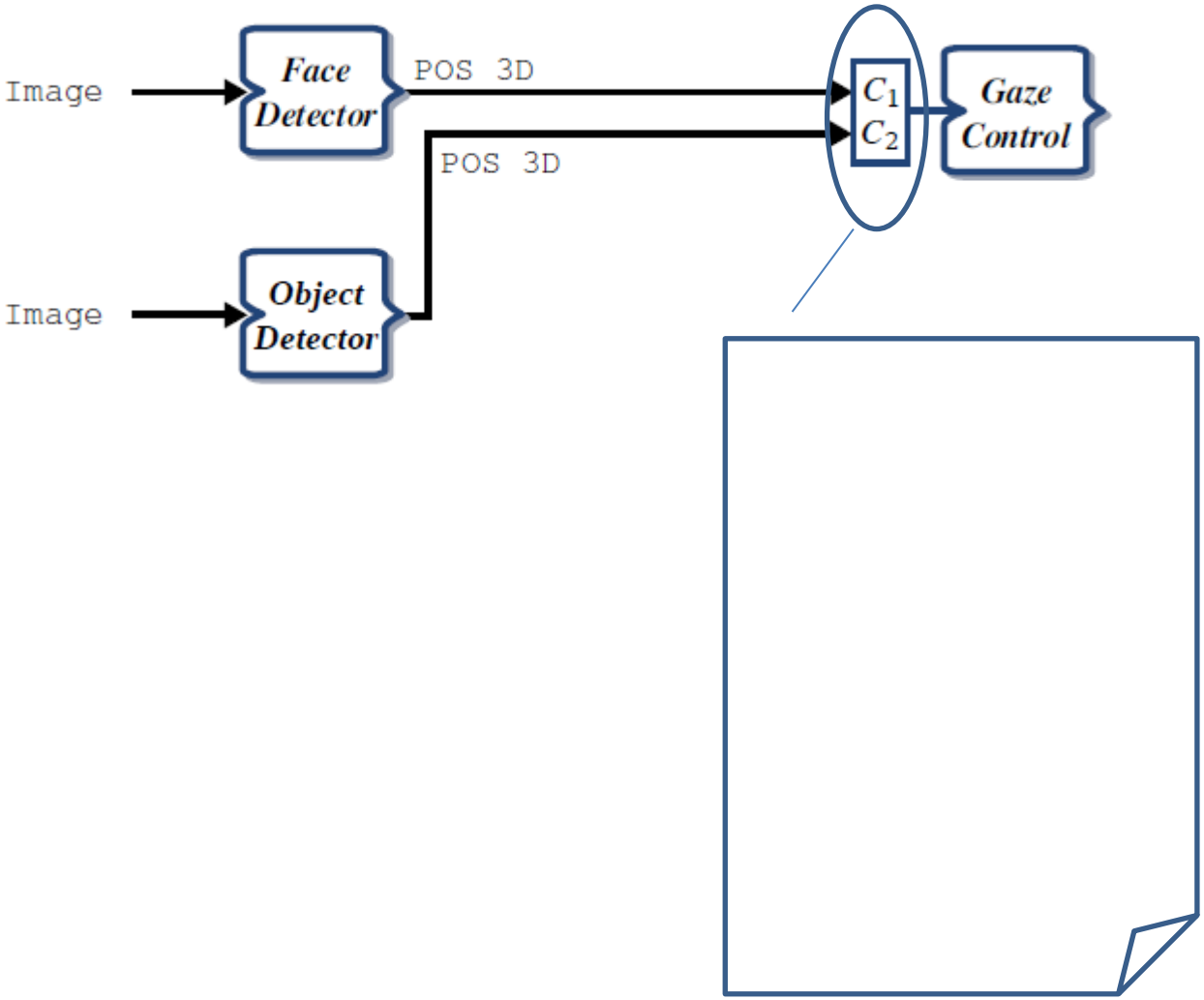




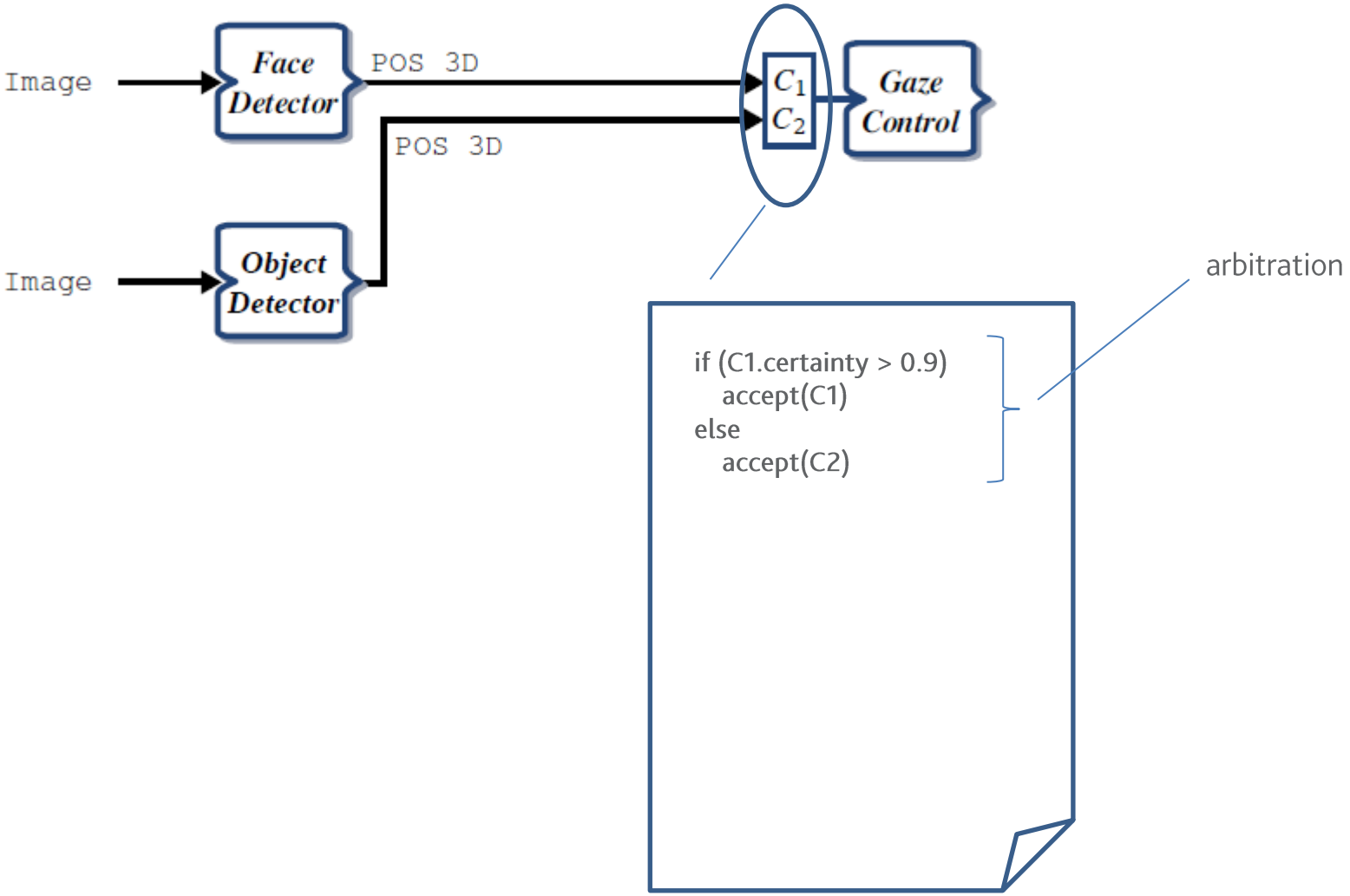
Our solution: Port Monitor



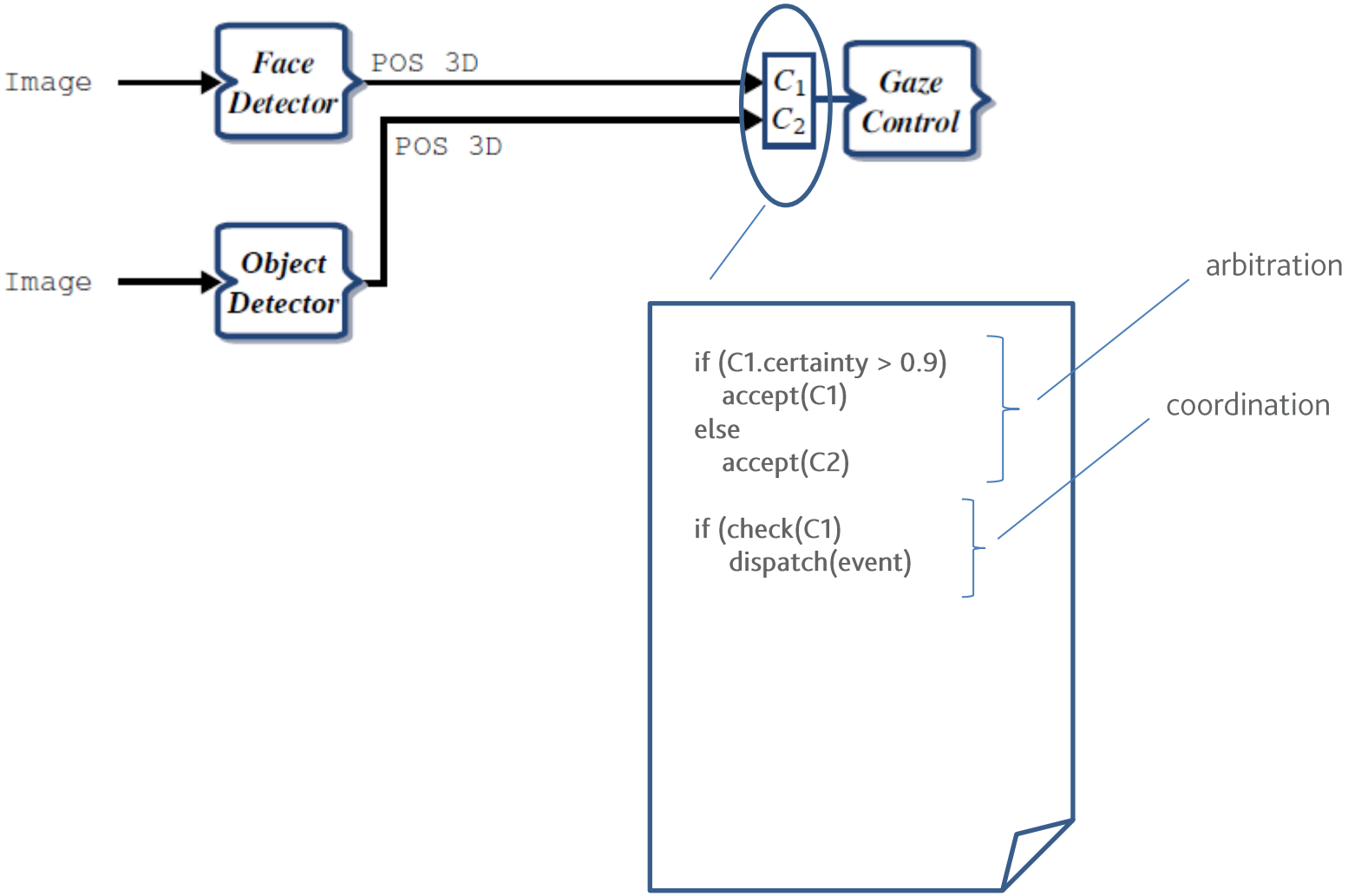
Our solution: Port Monitor



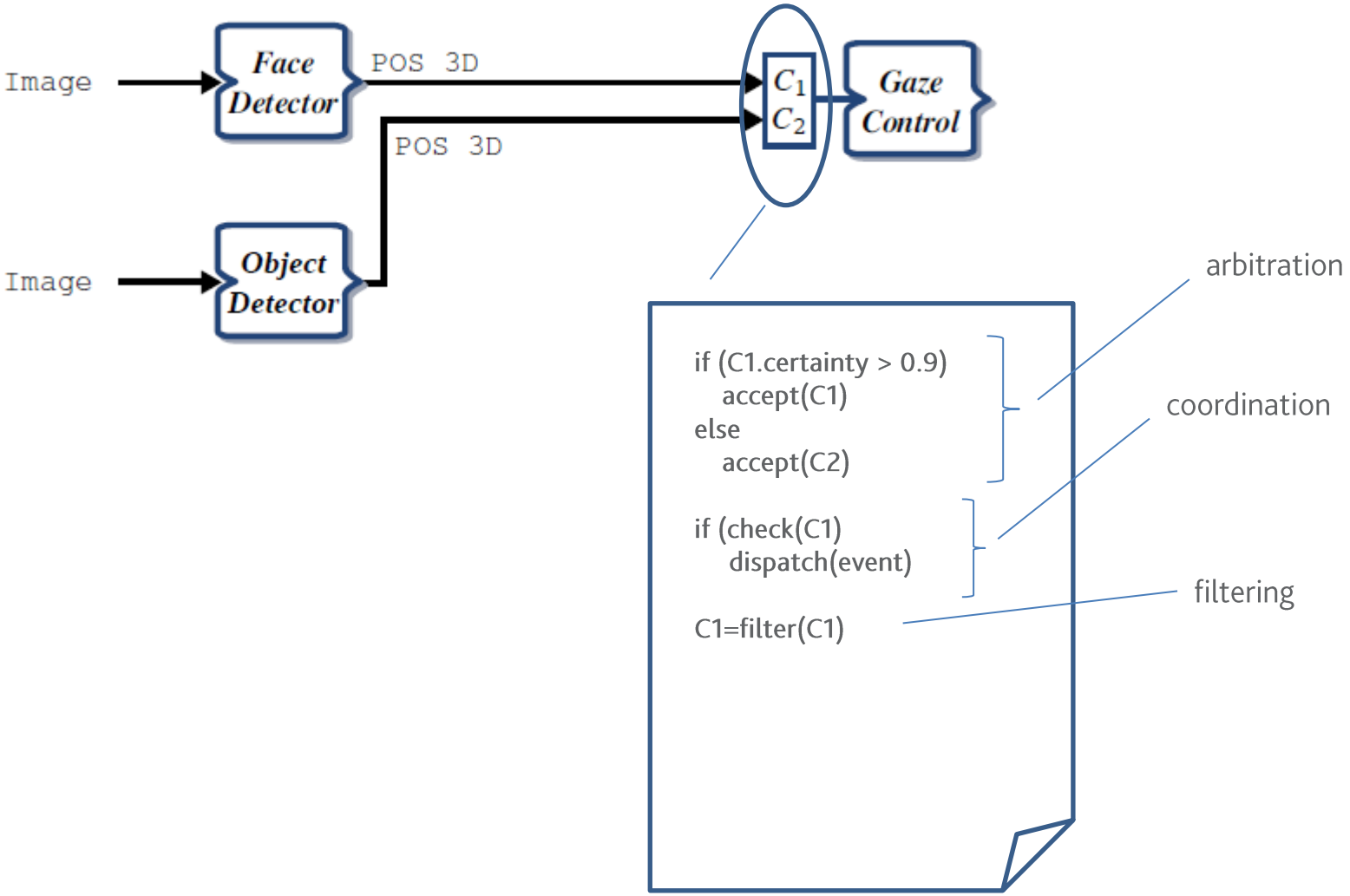
Our solution: Port Monitor



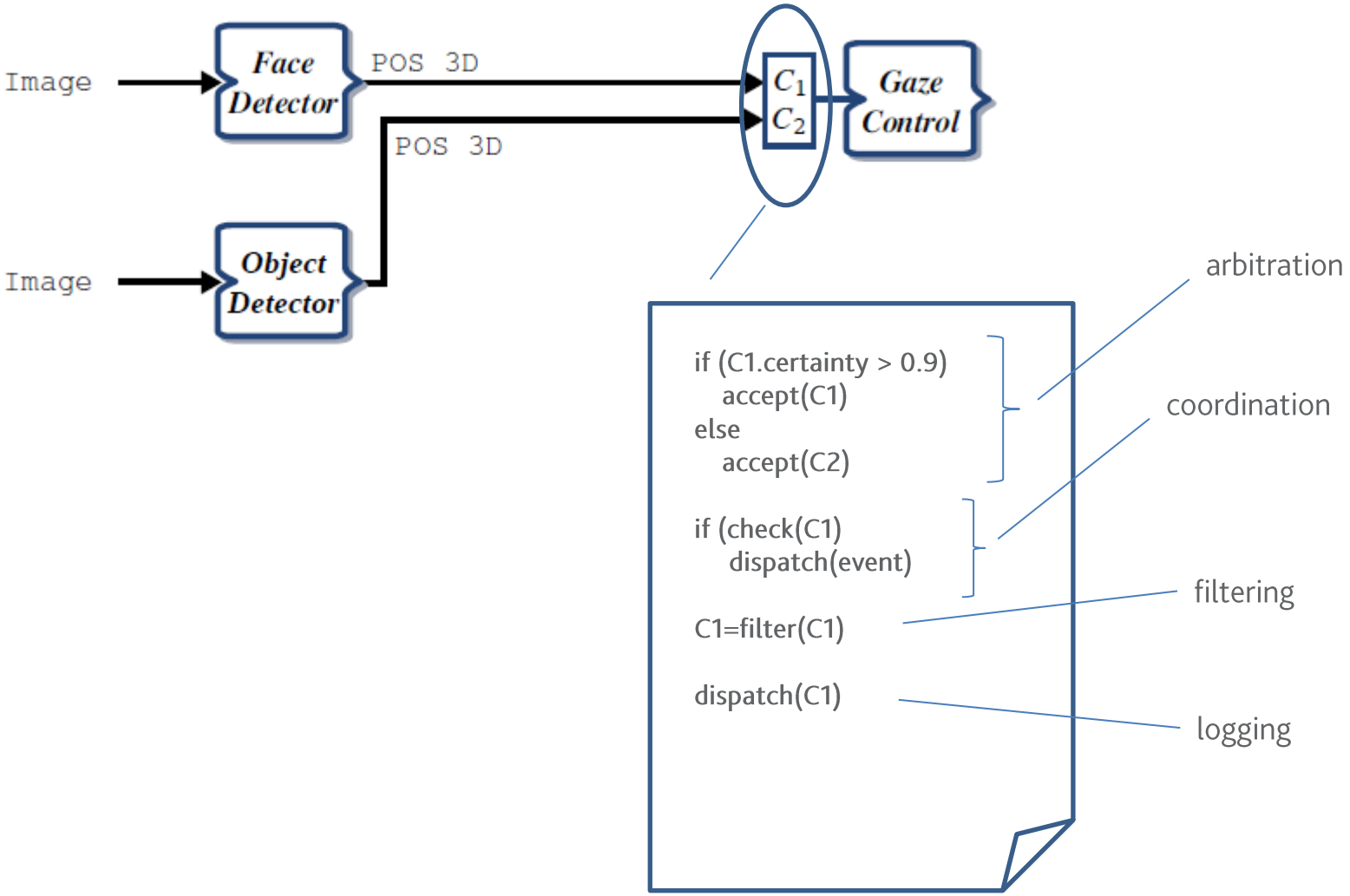
Our solution: Port Monitor



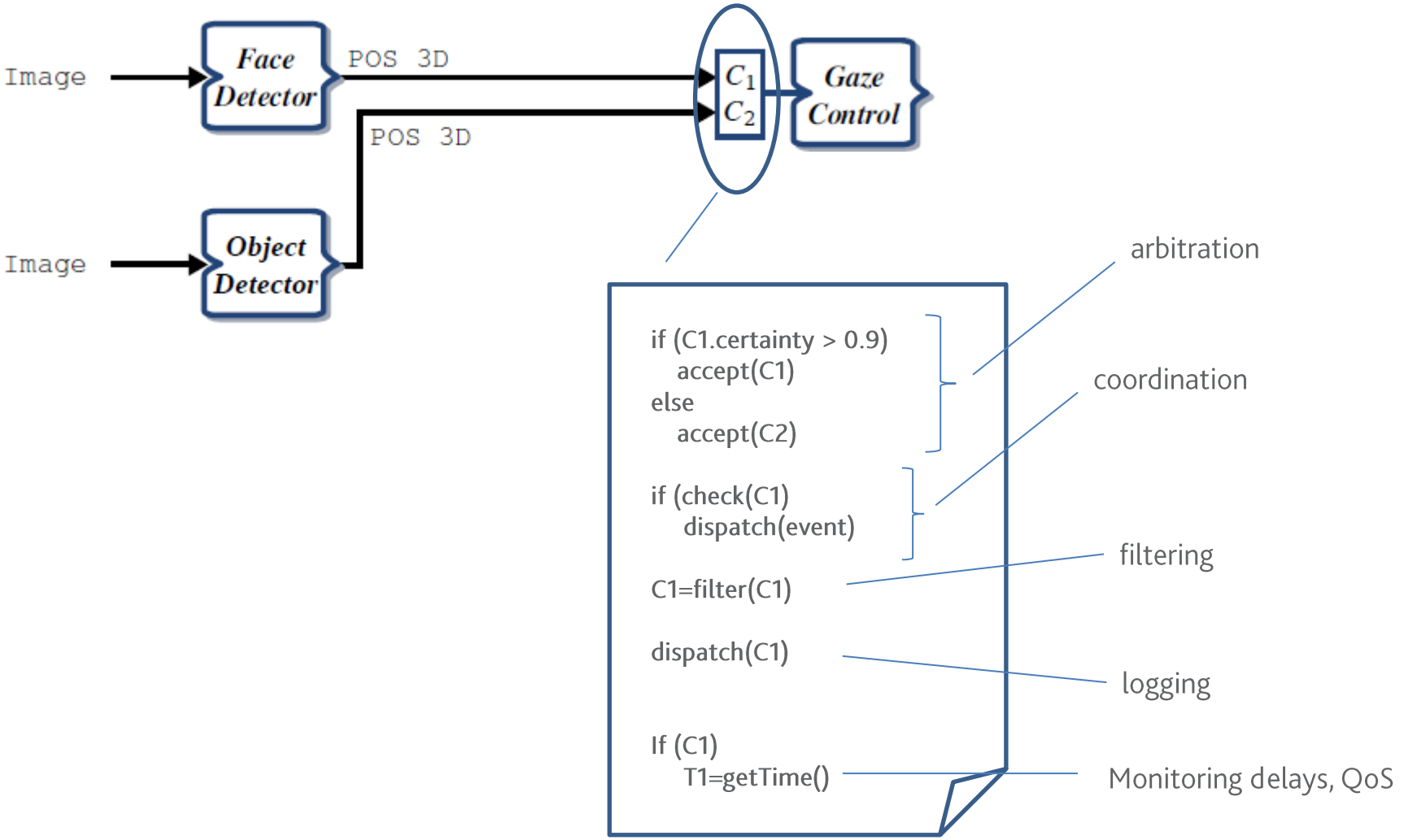
Our solution: Port Monitor

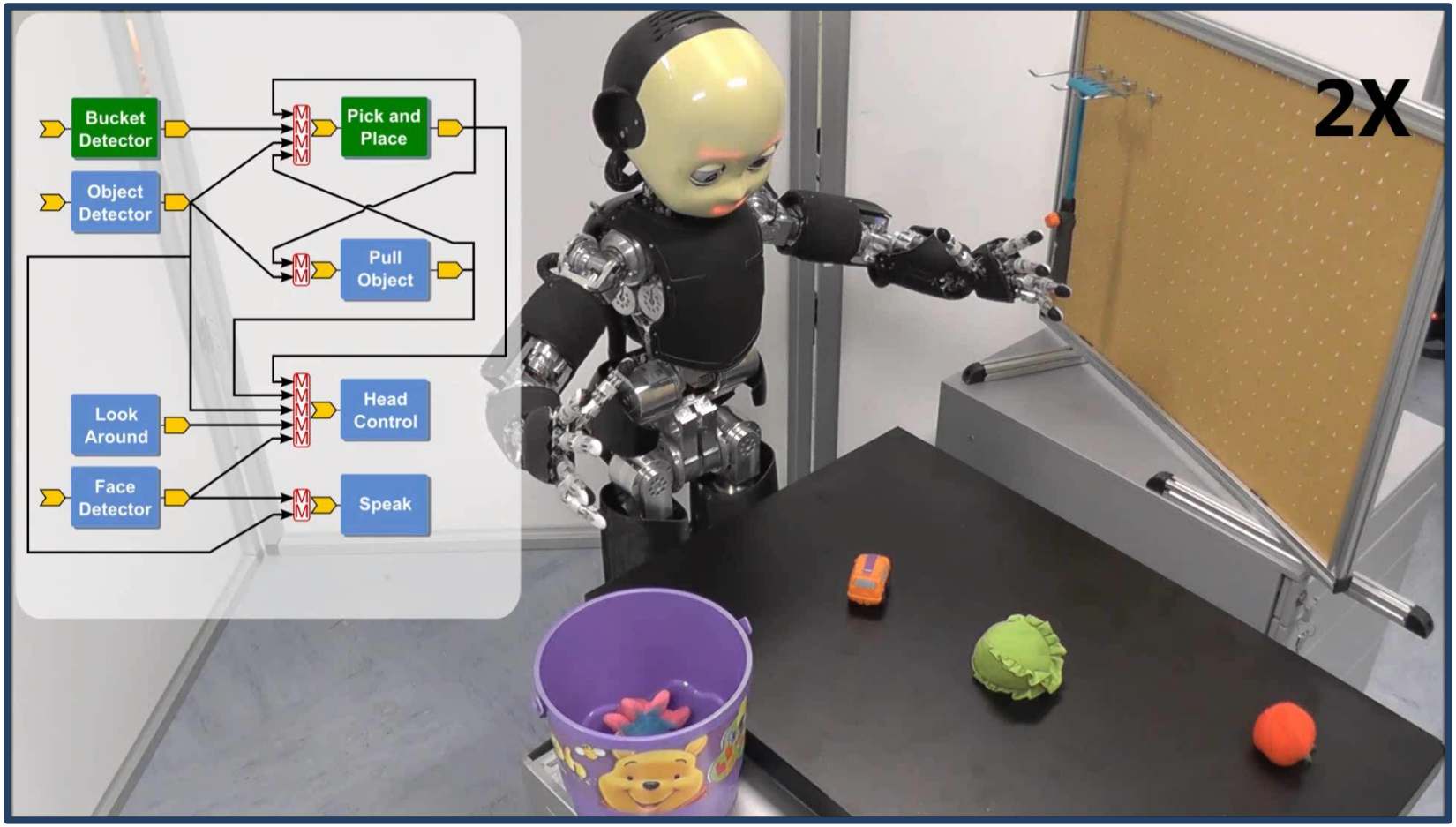


Our solution: Port Monitor



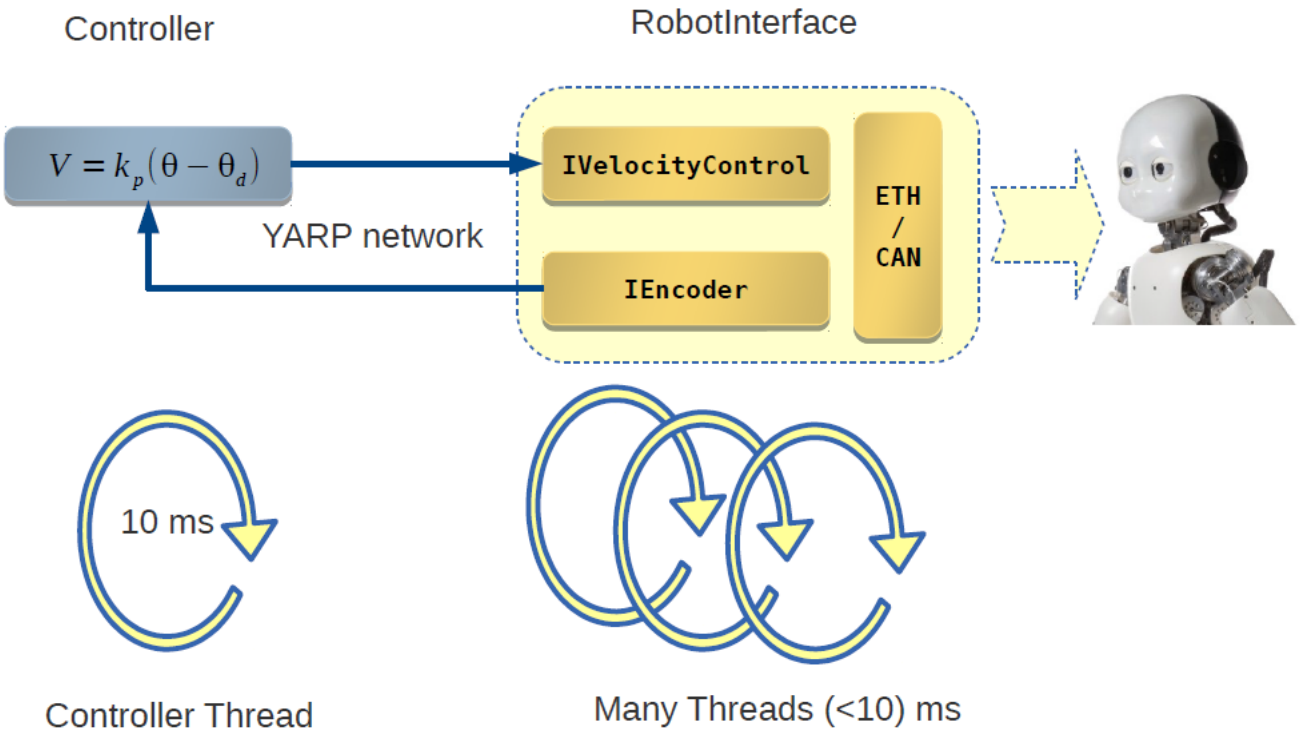
Our solution: Port Monitor





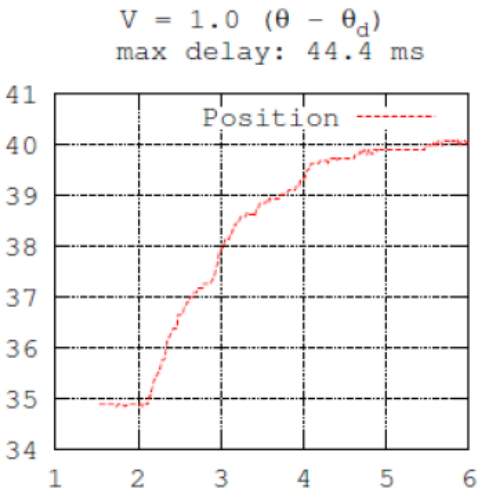
Paikan, A., et al., *Enhancing software module reusability using port plug-ins: an experiment with the iCub robot*, IROS 2014.

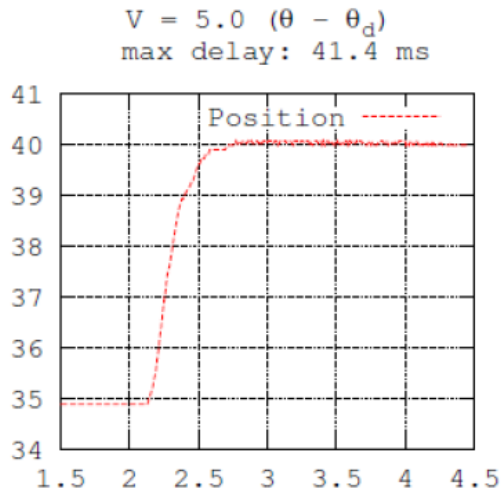
Improving determinism

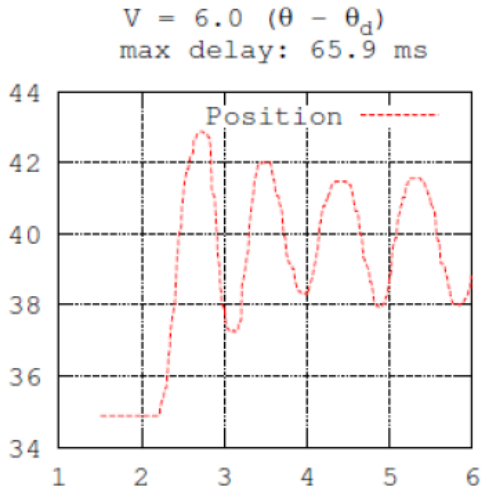


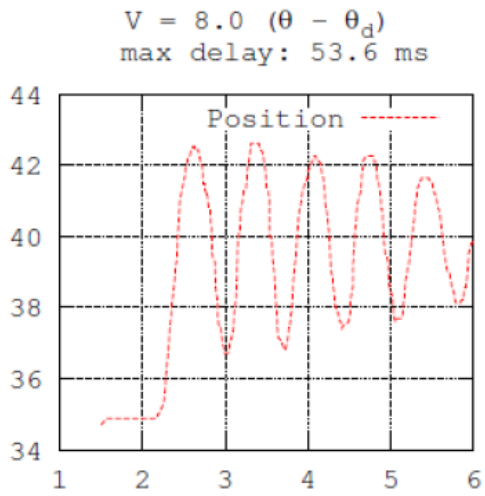
QoS

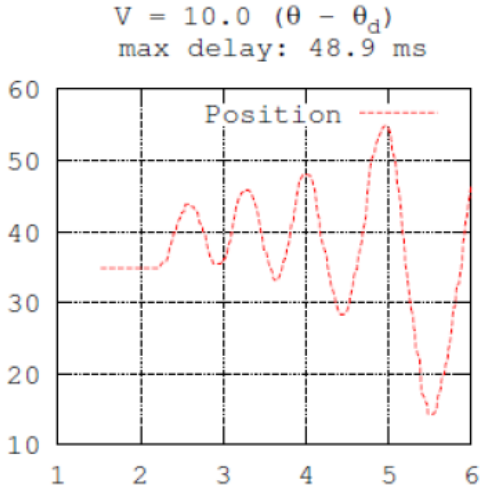
Quality of Service

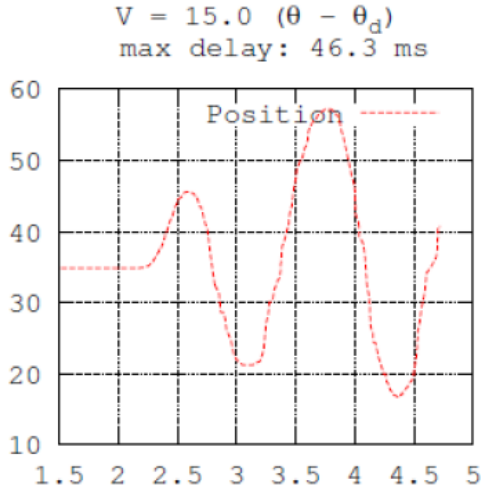

V = 1.0 (θ − θd)
max delay: 44.4 ms
Position
41
40
39
38
37
36
35
34
1
2
3
4
5
6


V = 5.0 (θ − θd)
max delay: 41.4 ms
Position
41
40
39
38
37
36
35
34
1.5
2
2.5
3
3.5
4
4.5

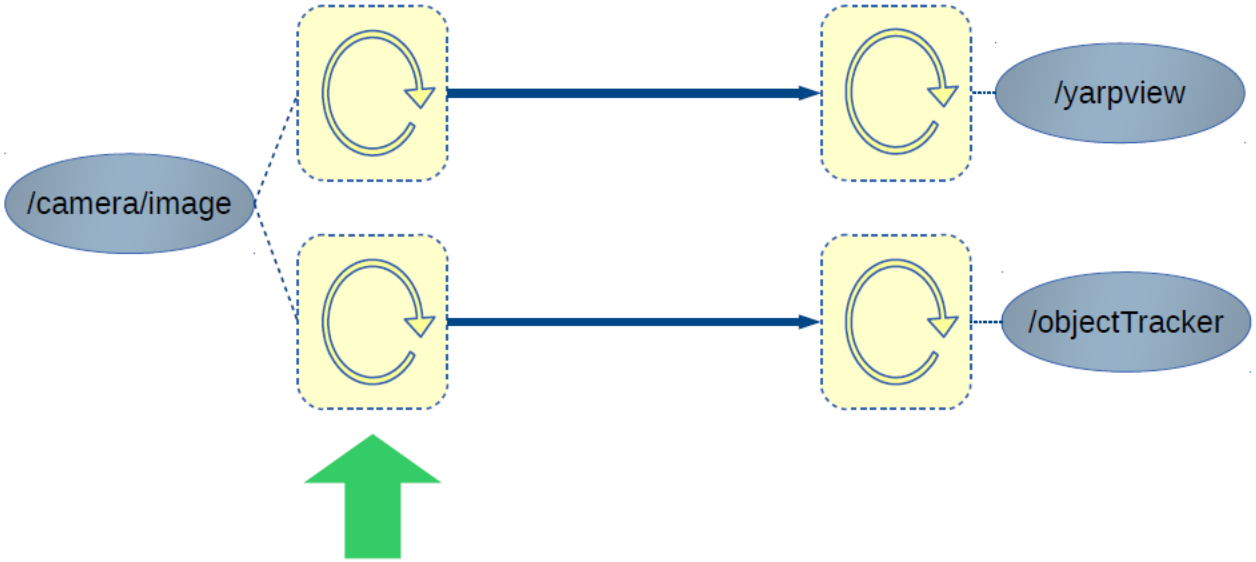

V = 6.0 (θ − θd)
max delay: 65.9 ms
Position
44
42
40
38
36
34
1
2
3
4
5
6


V = 8.0 (θ − θd)
max delay: 53.6 ms
Position
44
42
40
38
36
34
1
2
3
4
5
6

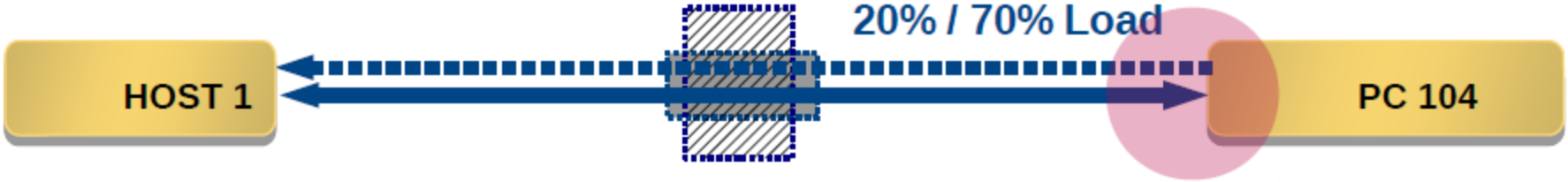

V = 10.0 (θ − θd)
max delay: 48.9 ms
Position
60
50
40
30
20
10
1
2
3
4
5
6

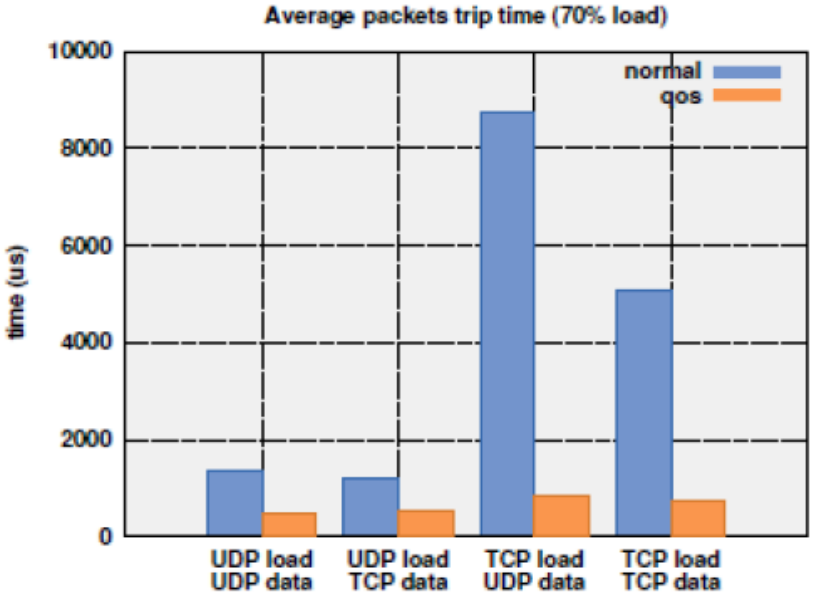

V = 15.0 (θ − θd)
max delay: 46.3 ms
Position
60
50
40
30
20
10
1.5
2
2.5
3
3.5
4
4.5
5

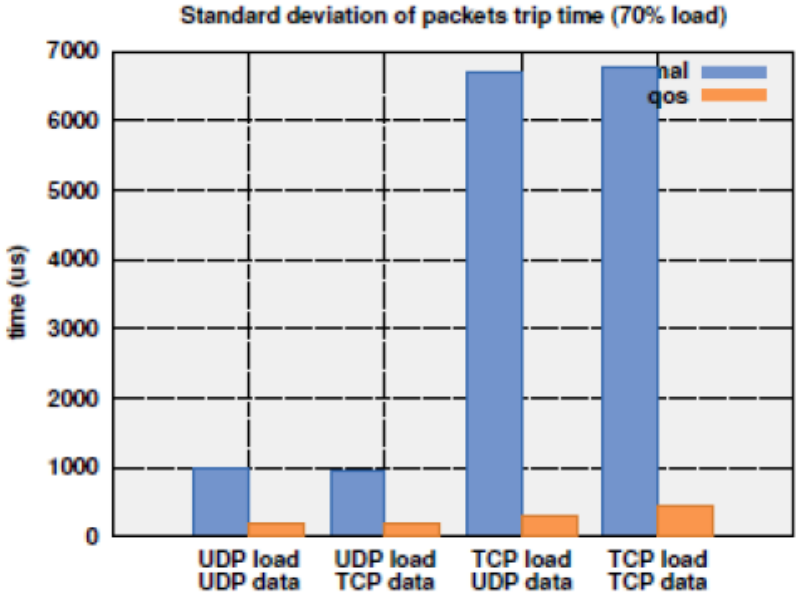
- Improve determinism by increasing *thread priorities* and reducing network bottlenecks using *QoS:*




```
> prop set /objectTracker sched policy 1 priority 30
> prop set /objectTacker qos priority HIGH
```


20% / 70% Load
HOST 1
PC 104


Average packets trip time (70% load)
time (us)
10000
8000
6000
4000
2000
0
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)
7000
6000
5000
4000
3000
2000
1000
0
qos
UDP load UDP data
UDP load TCP data
TCP load UDP data
TCP load TCP data

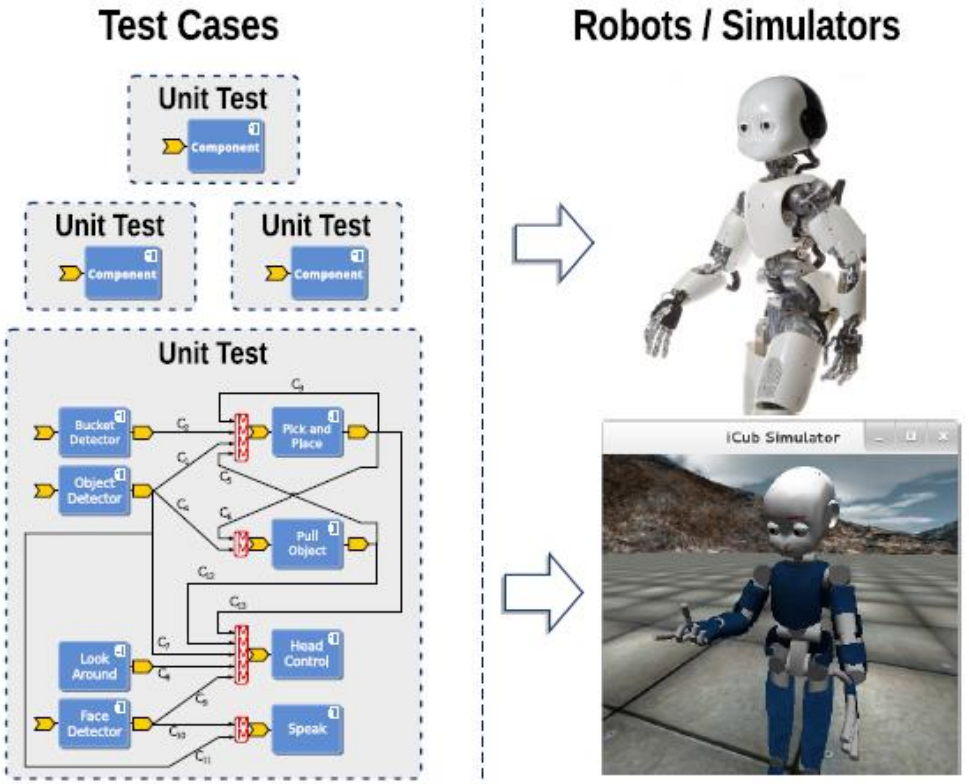
Different levels of testing

Testing of Code: CTest/CDash

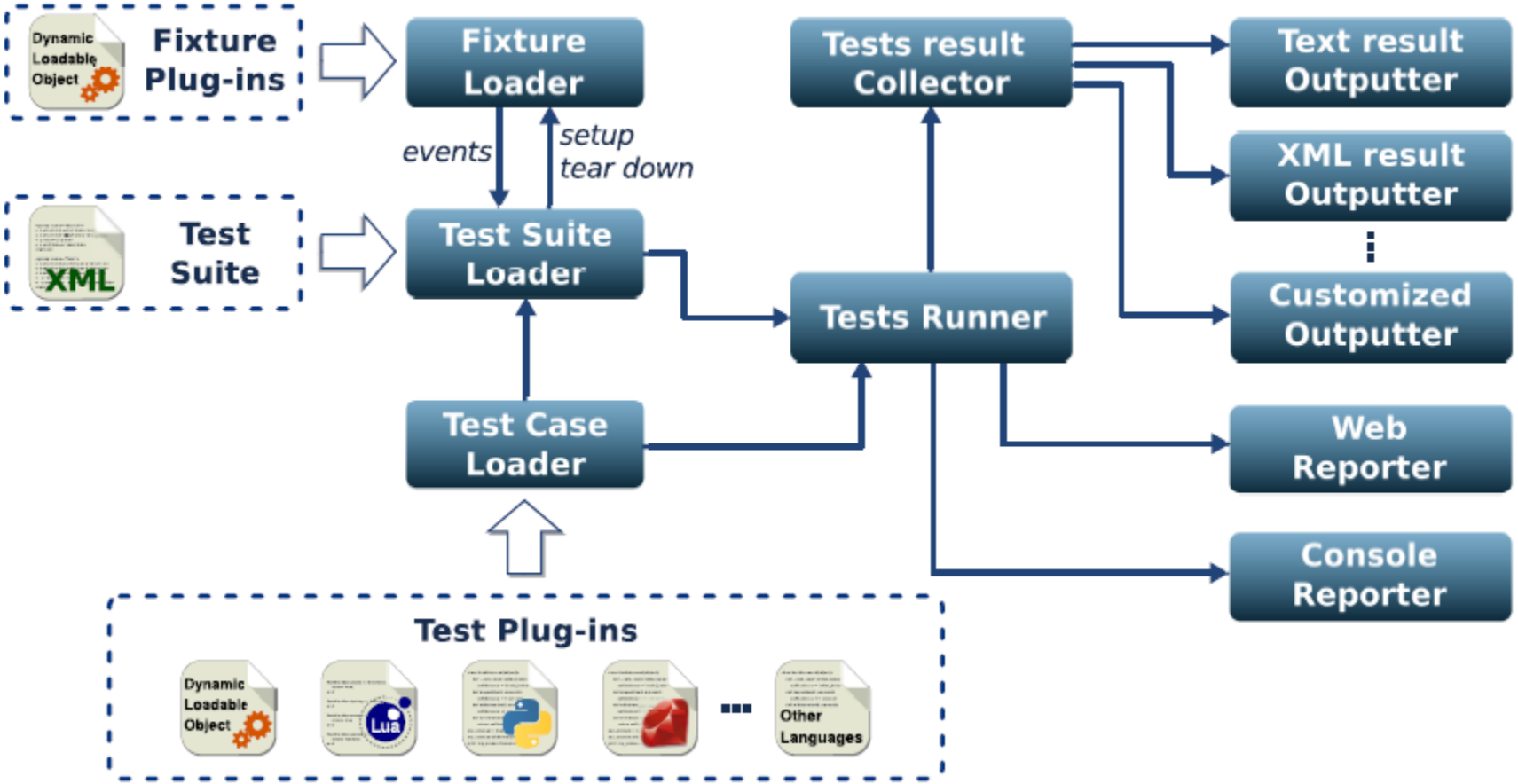
A testable unit for a robot *system*:

- Piece of code
- Driver/Hardware component
- Software component
- Application

- Hardware or software dependencies (real or simulated robot, other software components)



Testing framework



Fixture manager

- Setting up resources
  - Execute simulator
  - Execute preparatory routines (parking/calibrator etc)
- Monitoring resources
  - Ensure they remain functional
  - Handle failure appropriately
  - Restarting/parking the robot

More: https://github.com/robotology/robot-testing

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.