AADL models for ROS based applications

Eric SENN, Lucie BOURDON
Lab-STICC
Université de Bretagne Sud
Lorient, France

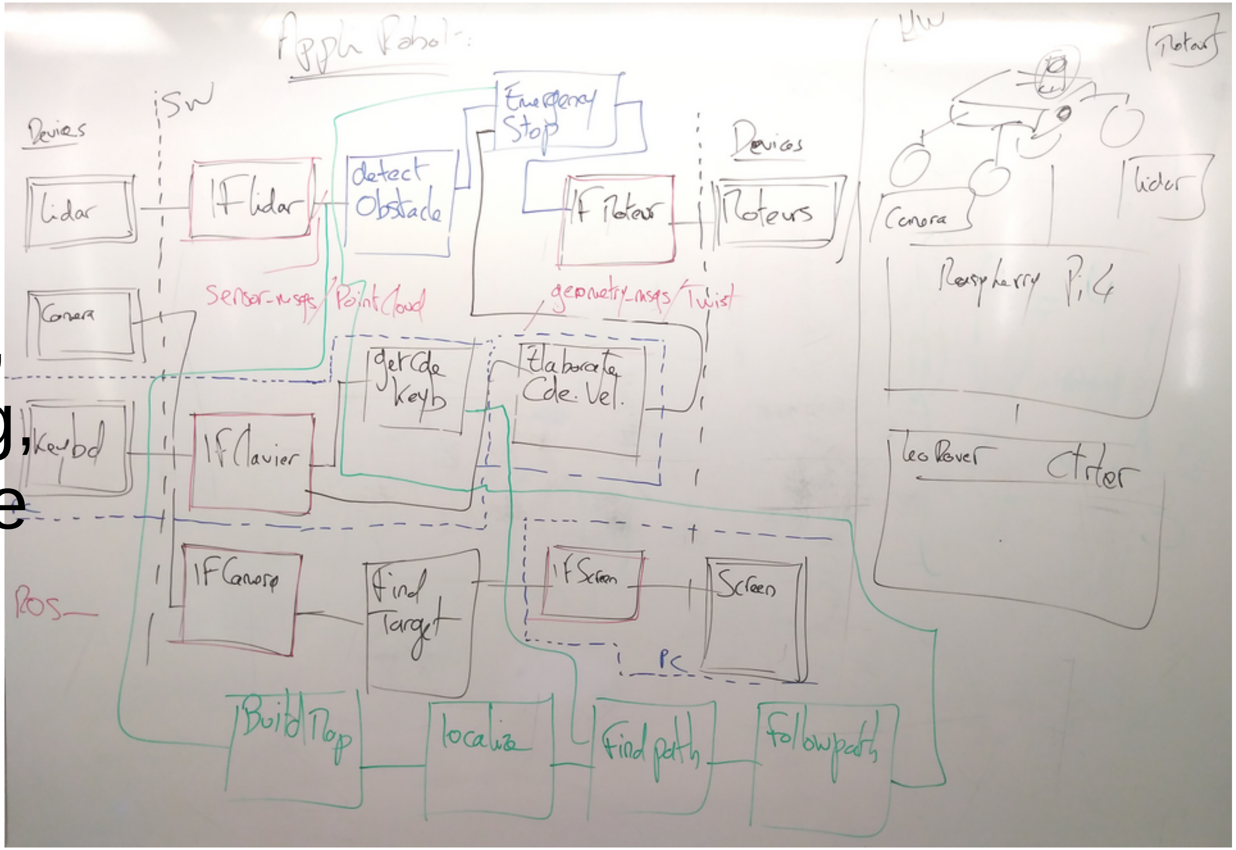
ADEPT Workshop, 17 June 2022

Our applications

- Mobile robots, in industry ~~4.0~~ 5.0

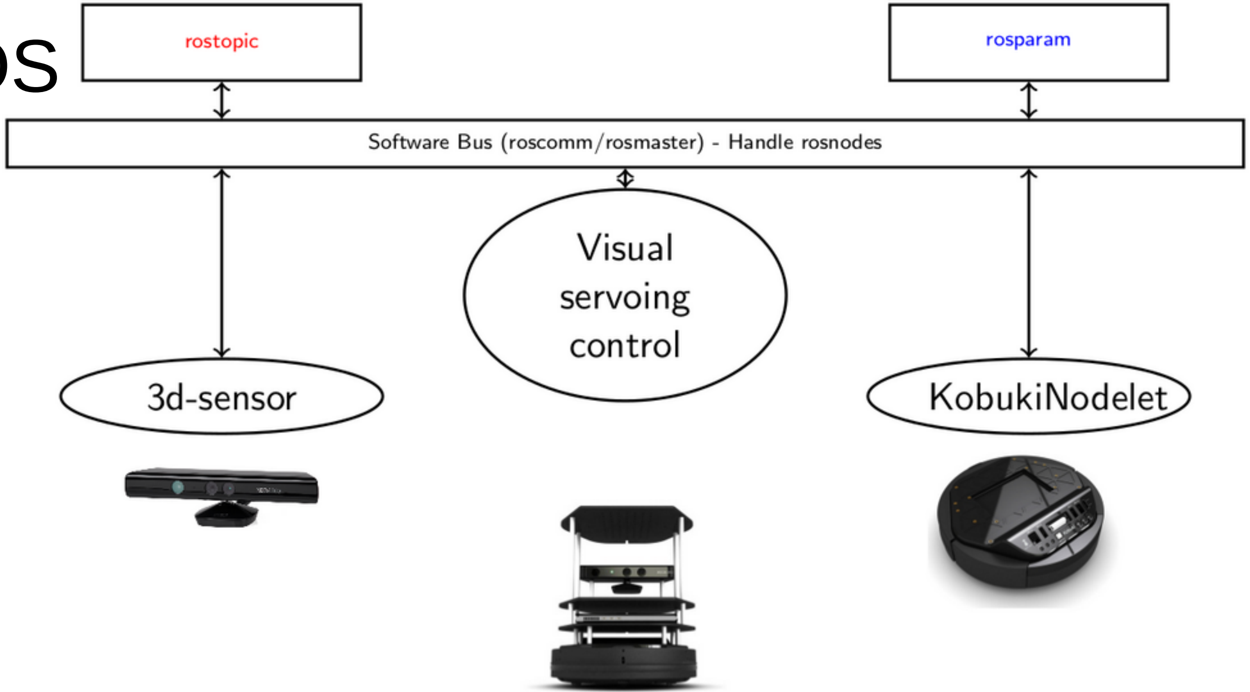
Complex applications

- many services and relations difficult to represent
  - navigation, localization, mapping, path-planning, target tracking, obstacle avoidance, security, reporting ...



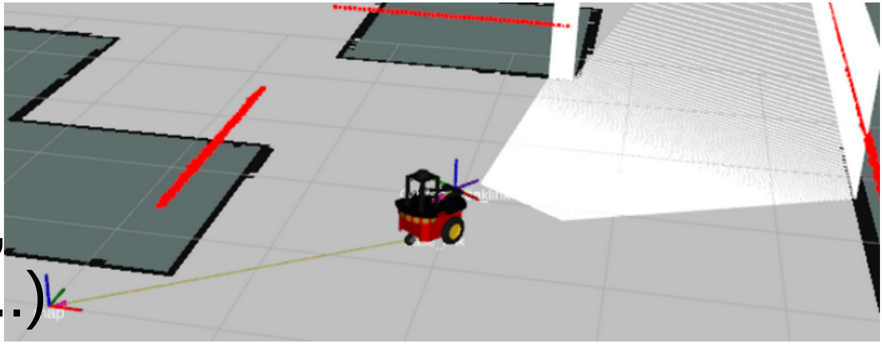
ROS Robot Operating System

- middleware to ease the programming of robots
  - synchronisation and communication mechanisms to hide low-levels OS services
  - multi-platform / multi-OS

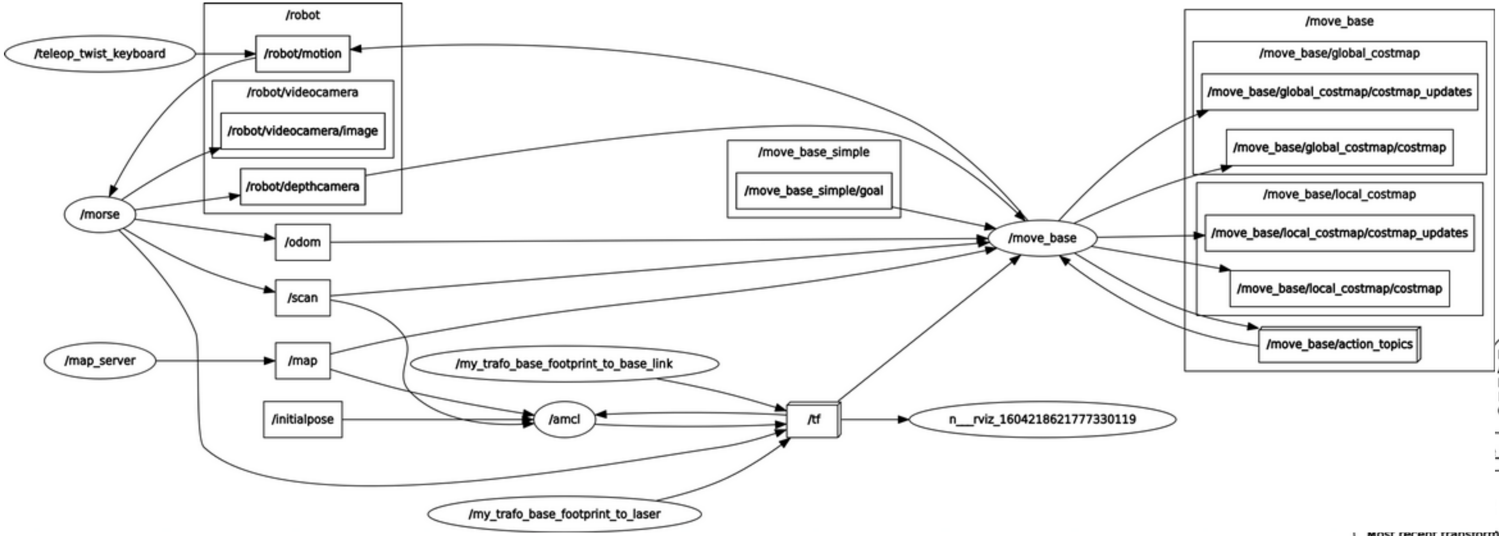


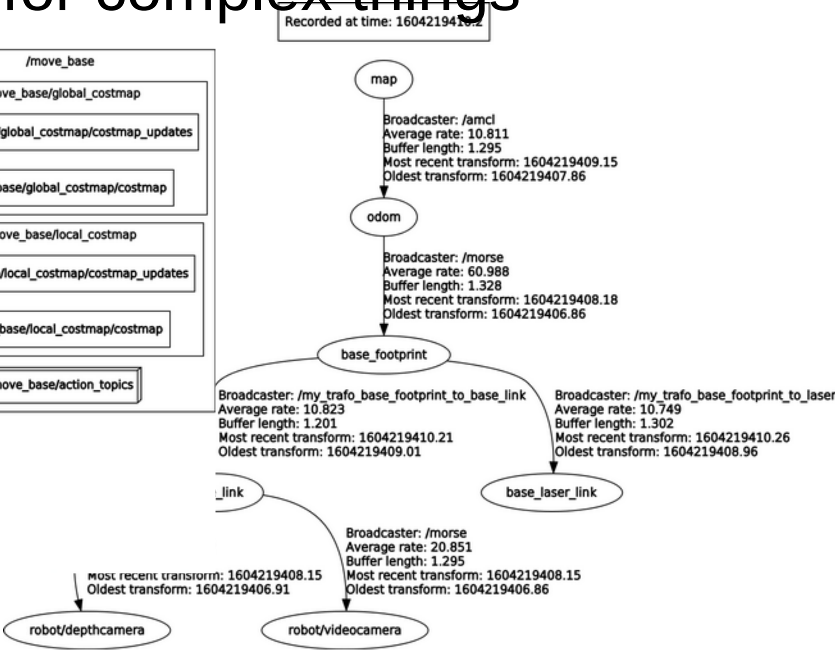
ROS because ...

- set of tools for development, monitoring, debugging, simulation (gazebo, morse ...)

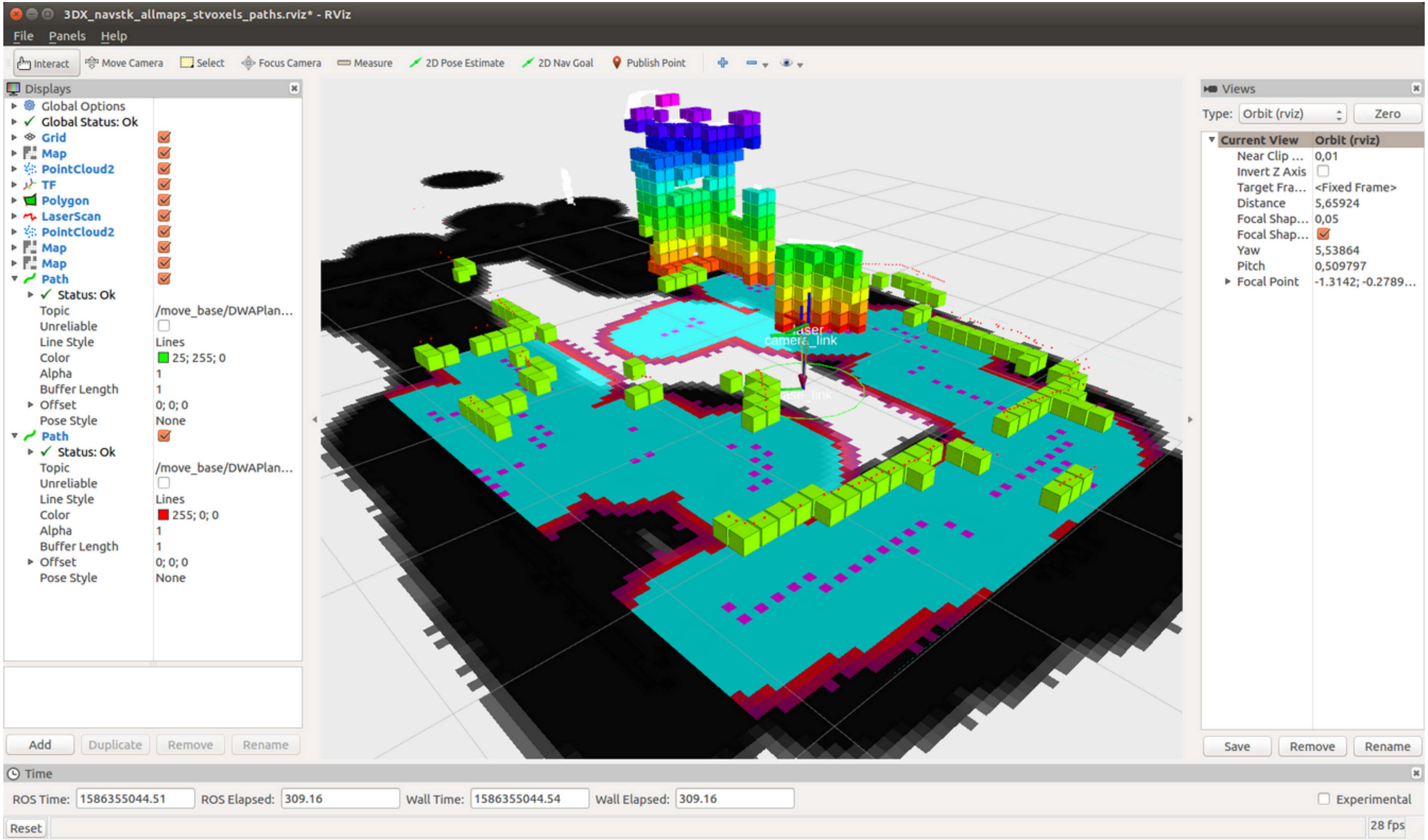


- used in the industry : demand from our ~~clients~~ partners
- we ~~are not smart enough~~ don’t have time for complex things





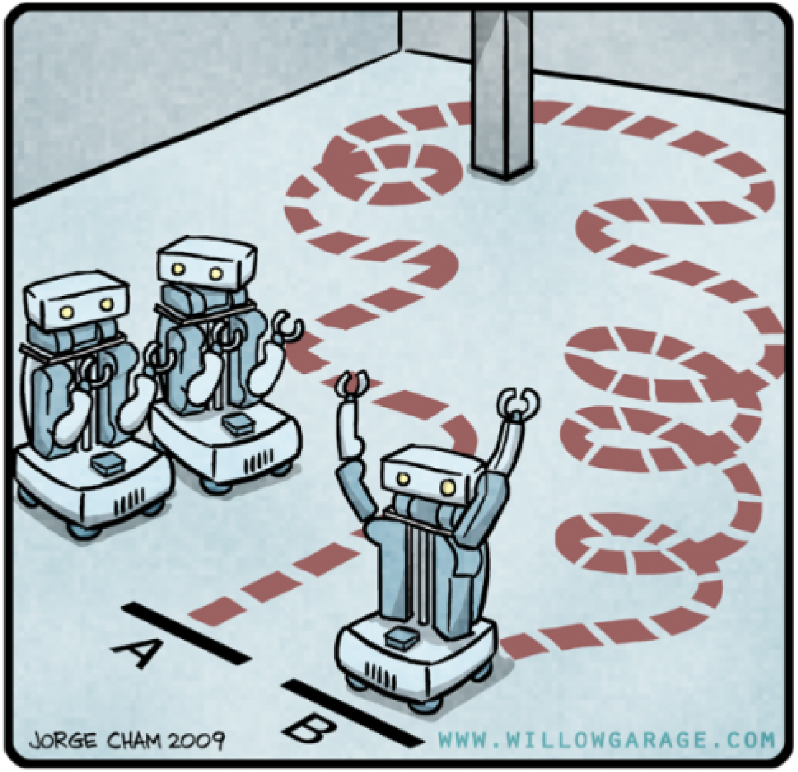
… and it makes nice pictures …



BUT … performance issues !

- Non functioning or malfunctioning robots
  - the robot is too slow, or lose its way, or its target
  - we observe :
    - high CPU load
    - slow communications
    - missed deadlines

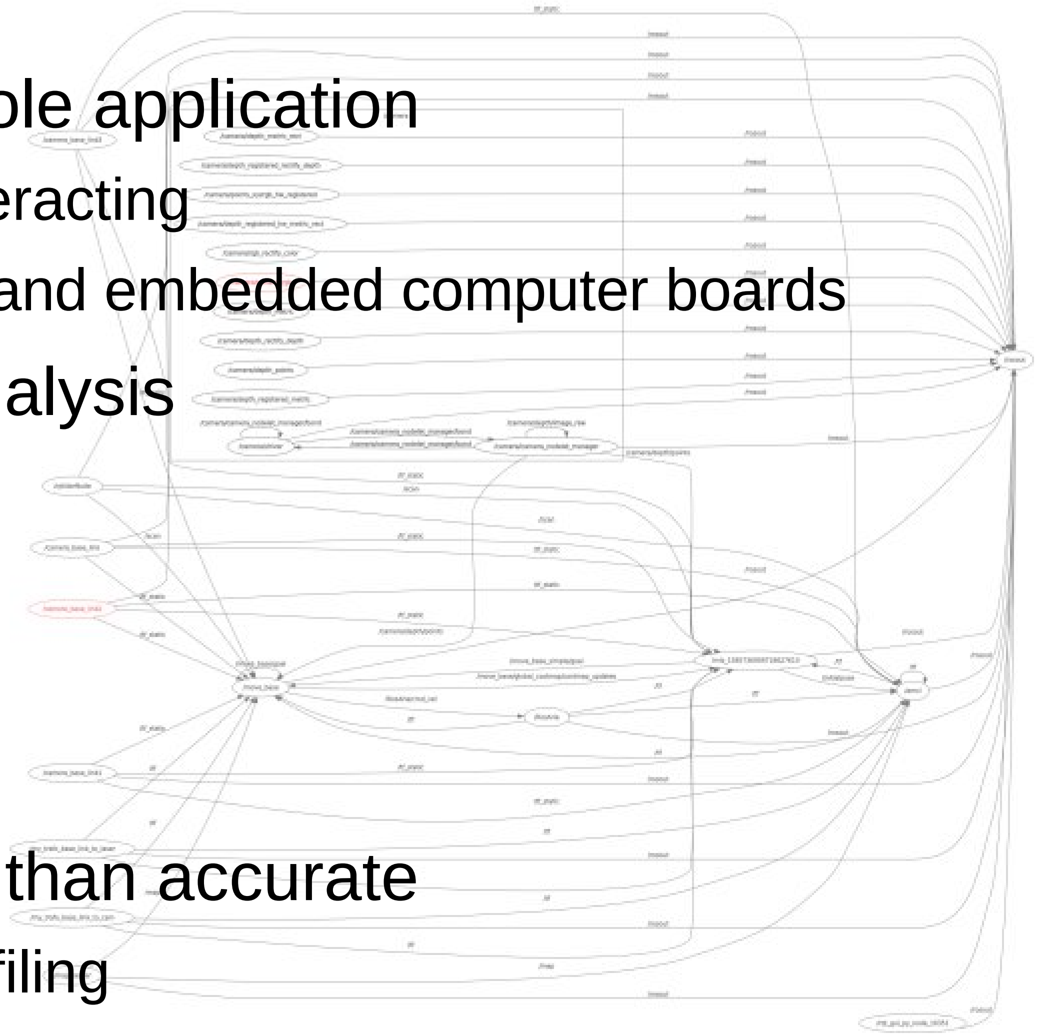
R.O.B.O.T. Comics



"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Our need

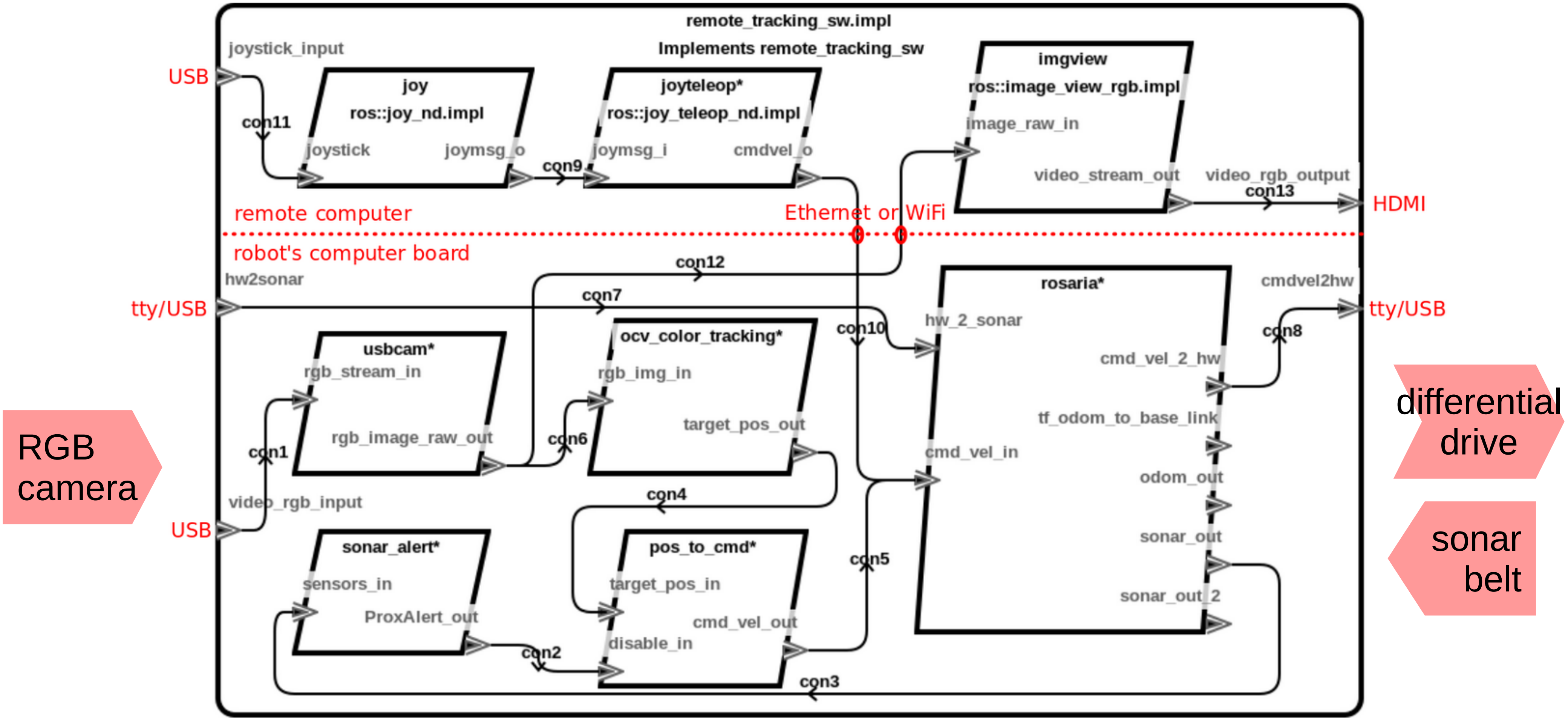
- A comprehensive view of the whole application
  - the software: a set of ROS nodes interacting
  - the hardware: the robot, its sensors, and embedded computer boards
- A tool to perform performance analysis
  - Timing : schedulability & latency
  - CPU load analysis
  - BUS load analysis
- ASAP in the development cycle
- Something more simple and fast than accurate
  - simple & fast modeling, analysis, profiling



Our choice

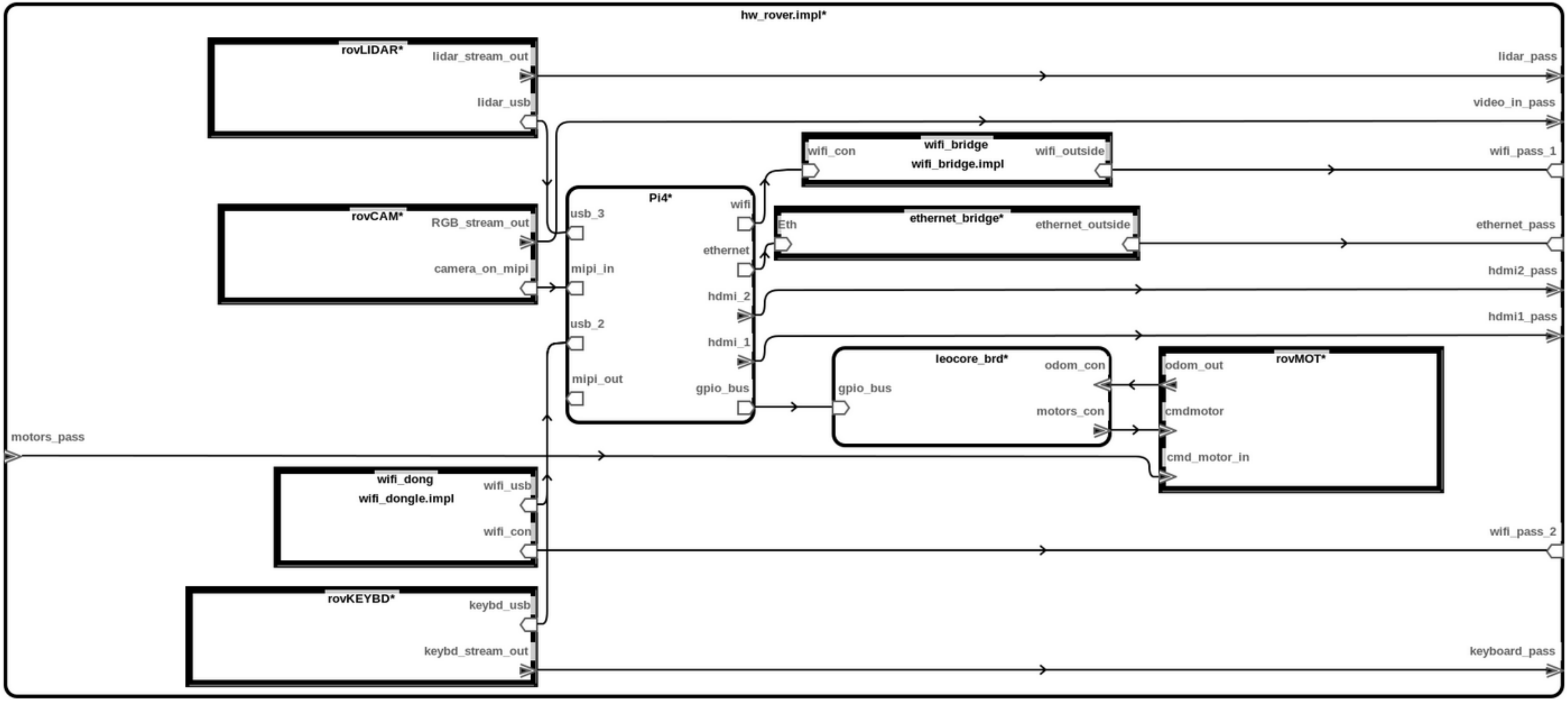
- AADL (Architecture Analysis and Design Language)
  - supporting Model Based Engineering dev. cycle
  - with everything to model real-time embedded systems:
    - software components (process, thread, data, port …)
    - hardware components (processor, bus, memory, devices …)
    - deployment specification with bindings : specify to which HW component(s) a SW component is bound to
- OSATE2 (Open Source AADL Tool Environment)
  - uses AADL properties to carry on proper analysis

Exemple : the application model



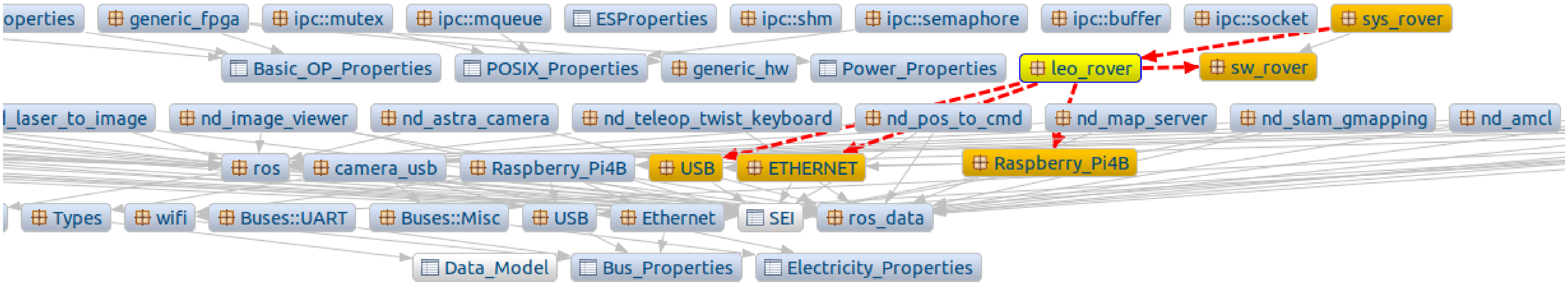
Exemple : the robot

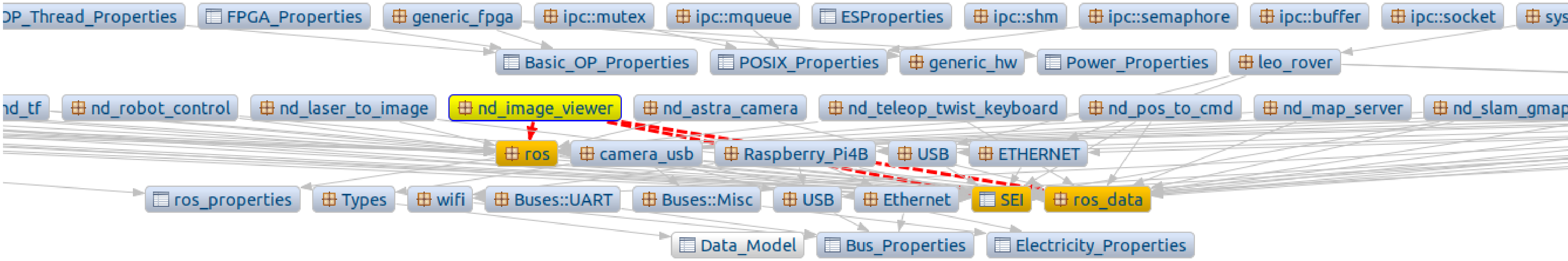
- leo rover: SBC (raspberry Pi4) + leo board (~arduino)



that goes to a Library …

- HW components & SW components …





SW components in ROS

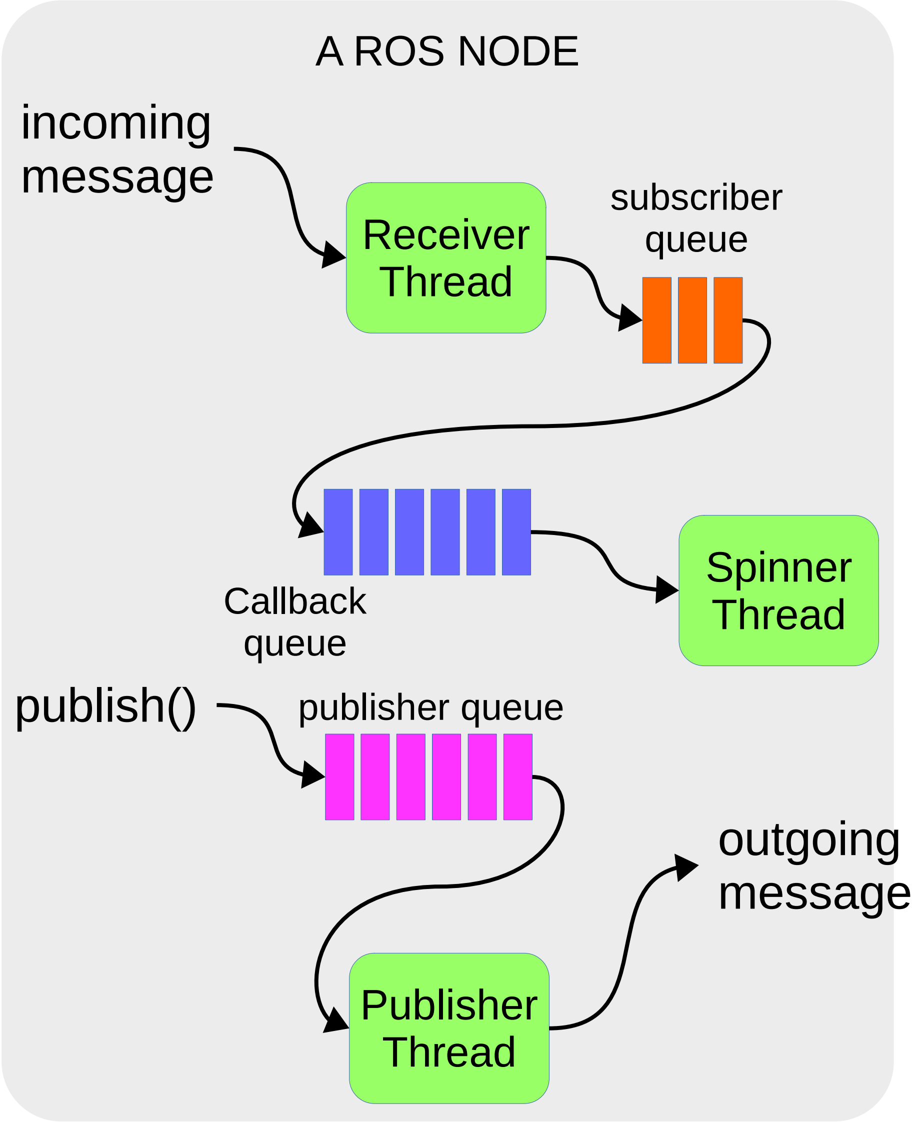
- a ROS node = one process
  - AADL imposes to have inputs / outputs defined in the component declaration
    - in / out event data ports
  - implementation shows what is inside
    - subcomponents and connections

```
process usb_cam_nd extends node
  features
    rgb_stream_in: in event data port video_stream.rgb;
    rgb_image_raw_out: out event data port Image.rgb;
  end usb_cam_nd;

process implementation usb_cam_nd.impl
  subcomponents
    image_broadcaster: thread imagePublisher.impl;
    usbSpinner: thread usbcam_spinner.impl;
  connections
    usb_cam_nd_impl_new_connection: port image_broadcaster.
     pub_msg -> rgb_image_raw_out;
    usb_cam_nd_impl_new_connection2: port rgb_stream_in ->
     usbSpinner.rgb_stream_in;
  end usb_cam_nd.impl;
```

ROS : threads inside

- ros node spawns threads :
  - the receiver (network) thread listens for incoming messages on TCP sockets and pushes them on the callback queue after deserializing them
  - the spinner thread pops messages from the queue and runs the user code : spinner thread run periodically or every time a message is received
  - publisher threads to send messages



Properties for BUS load analysis

- added properties :
  - thread component : ***Period***
  - data component : ***Data_Size***
  - system component :
    - ***Actual_Connection_Binding***
- enabling analysis :
  - bandwidth demand per connection

    = ***Data_Size*** / ***Period***
  - BUS load = Σ demands per connection bound to the bus

```
process implementation usb_cam_nd.impl
  subcomponents
    image_broadcaster: thread imagePublisher.impl;
    usbSpinner: thread usbcam_spinner.impl;
  connections
    con1: port image_broadcaster.pub_msg ->
     rgb_image_raw_out;
    con2: port rgb_stream_in -> usbSpinner.rgb_stream_in;
  end usb_cam_nd.impl;
```

```
thread imagePublisher extends publisher
  features
    pub_msg: refined to out event data port Image.rgb;
  end imagePublisher;

thread implementation imagePublisher.impl
  properties
    Period => 33333 us;--@ 30 images/s
  end imagePublisher.impl;

thread im
     imag
  properti
    Compu
  end ima
```

```
data topic  end topic;
data implementation topic.impl  end topic.impl;

data sensor_msgs extends topic end sensor_msgs;
data implementation sensor_msgs.impl extends topic.impl

data Image extends sensor_msgs end Image;
data implementation Image.impl extends sensor_msgs.impl end
      Image.impl;

data implementation Image.rgb extends Image.impl
  properties
    Data_Size => 921 KByte;
  end Image.rgb;
```

Bandwidth demands vs capacity

- BW demand checked against BW capacity
  - defined in the bus component with the ***BandwidthCapacity*** AADL property

```
system implementation Odroid_XU4.impl
  subcomponents
    Exynos_SOC: system Exynos_5422::Exynos_5422.impl;
    ethernet_bus: bus Ethernet::Ethernet.impl {SEI::
     BandWidthCapacity => 1000.0 MBytesps;};
    usb_bus_1: bus USB::USB.impl {SEI::BandWidthCapacity =>
      480.0 MBytesps;};
    usb_bus_2: bus USB::USB.impl {SEI::BandWidthCapacity =>
      4800.0 MBytesps;};
    usb_bus_3: bus USB::USB.impl {SEI::BandWidthCapacity =>
      4800.0 MBytesps;};
    hdmi_dev: device HDMI.impl;
  connections
    connection1: bus access ethernet -> ethernet_bus;
    connection2: bus access usb_1 -> usb_bus_1;
    connection3: bus access usb_3 -> usb_bus_3;
    connection4: bus access usb_bus_2 -> usb_2;
    connection5: port hdmi_dev.hdmi_port -> hdmi;
end Odroid_XU4.impl;
```

ROS software bus : TCPROS

- to carry messages between nodes running in the same computer
  - actual BW capacity depends on the CPU the nodes are running onto
  - communications impact on the CPU load ⇒ to be checked
  - different hardware targets ⇒ different implementations

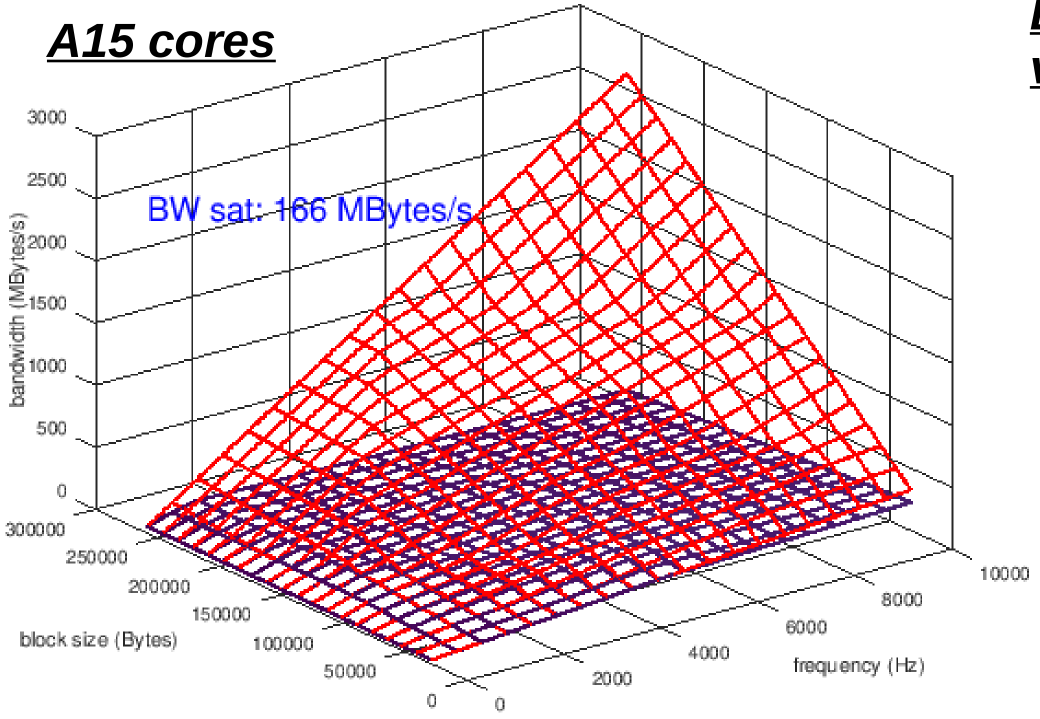
```
bus ros_bus end ros_bus;
bus implementation ros_bus.no_taskset extends ros_bus.impl
  properties
  SEI::BandWidthCapacity => 122.0 MBytesps;
  end ros_bus.no_taskset;

bus implementation ros_bus.A15 extends ros_bus.impl
  properties
    SEI::BandWidthCapacity => 166.0 MBytesps;
  end ros_bus.A15;
```

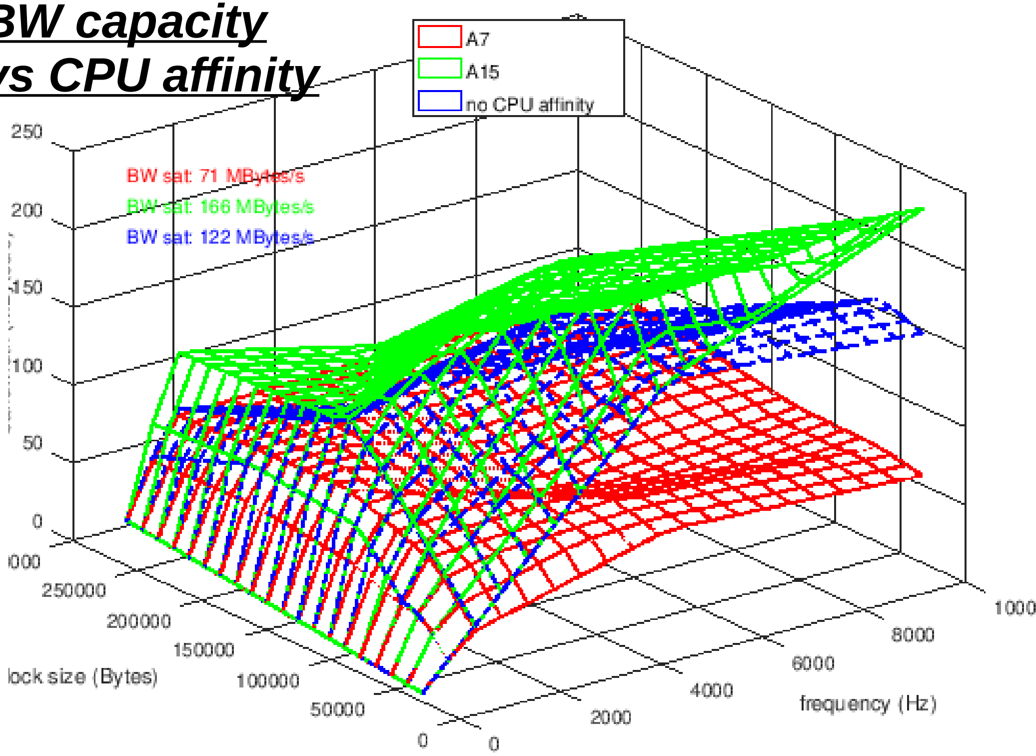
Profiling bus capacities

- experimental setup : producer → listener with growing messages sizes and rates

A15 cores



BW capacity vs CPU affinity



Properties for CPU load analysis

- added properties :
  - ***Compute_execution_time***
  - ***Period***
  - ***MIPSBudget*** (SEI standard)
  - ***Actual_Processor_Bindings***
- enabling analysis :
  - load per thread = compute_execution_time / period
  - total processor load = Σ load per thread bound to the processor
  - MIPS demand for a processor = MIPSCapacity x load

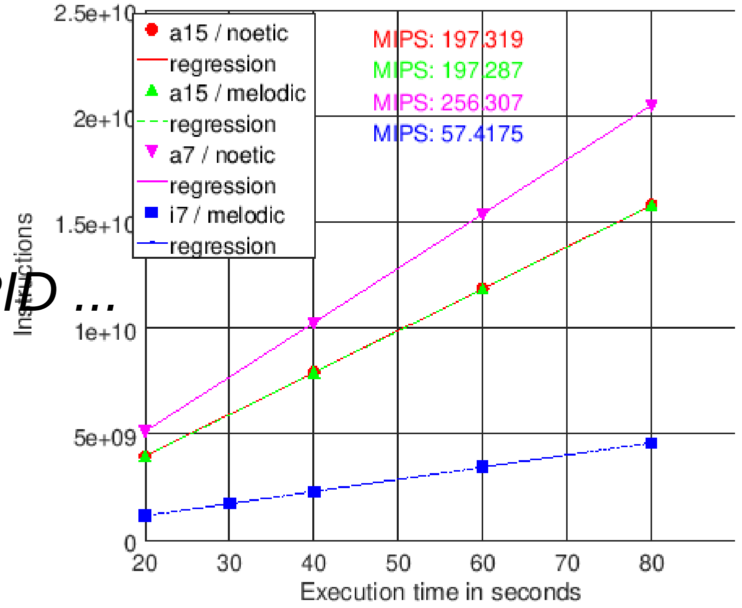
```
process implementation usb_cam_nd.xu4_a15 extends
    usb_cam_nd.impl
  subcomponents
    image_broadcaster: refined to thread imagePublisher.
     xu4_a15;
  properties
    SEI::MIPSBudget => 141.0 MIPS;
  end usb_cam_nd.xu4_a15;

thread implementation imagePublisher.xu4_a15 extends
    imagePublisher.impl
  properties
    Period => 33 Ms;
    Compute_execution_time => 2319 us .. 2319 us;
  end imagePublisher.xu4_a15;
```

```
system implementation color_tracking_dep.i
  subcomponents
    c_t_sw: system color_tracking_sw::color_tracking_sw.
     impl;
    p3dx: system Pioneer3DX::Pioneer3DX.i;
    ROSbus: virtual bus ros::ros_bus.impl;
  Properties
    Actual_Processor_Binding => (reference (p3DX.OdroidXU4
     .Exynos_SOC.big_procs_cluster.big_proc1)) applies to
     c_t_sw.usbcam;
    Actual_Processor_Binding => (reference (p3DX.OdroidXU4
     .Exynos_SOC.big_procs_cluster.big_proc2)) applies to
     c_t_sw.ocv_color_tracking;
    Actual_Processor_Binding=> (reference(p3DX.OdroidXU4.
     Exynos_SOC.little_procs_cluster.little_proc1)) applies
     to c_t_sw.pos_to_cmd;
  end color_tracking_dep.i;
```

Profiling a ROS node

- launch the node, in realistic situation, setting
  - CPU frequency : *cpufreq-set ...*
  - CPU affinity : *launch-prefix="taskset -c 5,6,7" …*
  - Process scheduling policy & priority : *chrt -f -p 15 PID ...*
- record performance for different durations
  - *perf stat -p PID -- sleep duration*
- using scripts (shell, awk …)



number of instr	number of cycles	freq (GHz)	duration	MIPS	MIPS/frame	ipc	cycles/frame	run_time/fra	proc load	core
3947426403	2751780463	1.992	20.00375	197334288	6577810	1.43	4599867	0.00231	6.92751	A15
7899407914	5736961174	1.992	40.00616	197454813	6581827	1.38	4769440	0.00239	7.18289	A15
5121547760	15567239635	1.4	20	256077388	8535913	0.329	25945399	0.01853	55.59728	A7
10244464108	31191572553	1.4	40	256111603	8537053	0.32844	25992977	0.01857	55.69924	A7

Analysis with OSATE2

- performing CPU load & BUS load analysis
  - average error 3,7%
  - modeling effort reasonable
    - fast profiling (a few hours)
    - fast analysis (a few seconds)

ROS node	CPU	MIPS Budget	Execution time (ms)	Estimated load (%)	Measured load (%)	Error %
usb_cam	A7	776	18.5	56	48.7	7.3
	A15	141	2.32	7	9.5	2.5
color_tracking	A7	1705	40	121	low frame rate	
	A15	2205	37	112	93	19
pos_to_cmd	A7	8.4	1	0.63	0.61	0.02
	A15	1.0	0.7	0.44	0.5	0.16
sonar_alert	A7	10	3	9	8	2
	A15	3	1	2	1.3	0.7
rp_lidar	A7	18	4.47	5.8	3	2.8
	A15	16	1.23	1.6	1	0.6
slam_gmapping	A7	462	4448	88.9	90	1.1
	A15	1464	4279	87.8	82.8	5
rosaria	A7	9.5	1.40	4.20	0.85	3.35
	A15	6.5	1.19	3.57	0.66	2.91

Our ROS library of AADL models

- We provide a library of models for software components
  - Organized in packages according to ROS based applications
    - ROS nodes and complex services (SLAM, navigation stacks …) from mainstream ROS distributions
    - ROS data types and messages
    - ROS synchronisation and communication mechanisms
- We provide a library of models for hardware components
  - SBC : Jetson Xavier, Nano, Odroid XU4, Raspberry Pi4, Pi3 …
  - SoC : Exynos 5422, Broadcom BCM2711 ...
  - SoPC : Xilinx, Altera with hardcores/softcores (PowerPC, µBlaze, NIOS …)
  - Robots : (Pioneer3DX, LeoRover, TurtleBot ...)

Our ROS library of AADL models

- includes dedicated properties to allow for multiple analysis from OSATE2
  - Ressource allocation analysis :
    - CPU load / Bus load
    - Memory capacities / Power consumption / Weights
  - Timing and scheduling analysis
    - Schedulability, scheduling analysis
    - Flow latency analysis
- offers tools for Design Space Exploration in Model Based Engineering
  - choosing hardware targets & software architectures
  - exploring binding solutions / balancing between CPU vs Bus load
  - to guarentee reaction time for robotic applications