Why use/not use ROS?

Pros

- Large base of users, programs, web site, even company using ROS on their robots
- Tutorials and courses
- Learning curve is reasonable
- multi-cpu
- Rich environment & ecosystem

Cons

- roscore single point of failure
- Performance issues
- Not hard real-time
- Hardly model based software development
- Some high level paradigm (actions), smach, or even T-ReX (but hardly used and now obsolete)

ROS 2

- No more Master (**roscore**) (single point of “failure”)
- ROS Comm is replaced with **DDS** (Data Distribution Service)
  - DDS, better implementation? (some commercial)
- The concepts remain the same
- Move from multi platform to one platform automatically (former nodelet shared mem)
- **ACTION**s are now “first class citizen” (like **topics** and **services**)

- A lot of legacy software still under ROS 1…
- … but ROS 1 is not maintained anymore!

Some ROS 2 commands

- For developper: **catkin** replaced by **colcon**
- All ROS 2 commands are prefixed with "`ros2`"
  - `ros2 {node|topic|interface|service|action|…}`
- <tab> is your friend… `-h or --help also`
- Env variables to close and limit your deployment setup:
  - `ROS_LOCALHOST_ONLY` and `ROS_DOMAIN_ID`

PoCoLibs (LAAS)

POrtability and COmmunication LIBraries

- Initially developed for VxWorks, excellent real-time performance and memory footprint (portLib ensures compatibility of other Unixes with the VxWorks primitives)
- data sharing with shared memory (strongly typed)
  - proper lock mechanism
- **client/server** communication with mailbox
  - each component has an in/out mailbox (message queue)
  - receive request, reply with reports

https://git.openrobots.org/projects/pocolibs/gollum/index

PoCoLibs (LAAS)

PortLib (portability using Posix.4, (see your RT programming courses))

- Tasks (threads) management
- Semaphores (P & V)
- Watchdogs

PoCoLibs (LAAS)

ComLib

- Shared Memory (Poster)
- MailBox and csMailBox (cs Client Server)
  - attach callbacks to the Server side
  - send/receive request
  - send/receive reply
- Timers
- Posters, find, read, write, etc (properly locked)
- Events (inter process event com), wait on an event, etc

Why use/not use PoCoLibs

Pros

- Good real-time capabilities (first developed for VxWorks a real time OS)
- No memory allocation at run-time
- Good memory footprint
- It just does what is expected from a middleware, nothing more
- hardly used directly

Cons

- support for multi-cpu of posters require an additional component (PosterServ)
- used at LAAS and a few other places…

Other middleware

- DDS (used in ROS2)
- ZeroMQ
- Protocol Buffer / gRPC (used by Boston Dynamics)
- YARP
- RT Middleware (OpenRTM AIST)
- Orocos RTT (used by Orocos)

Above middleware

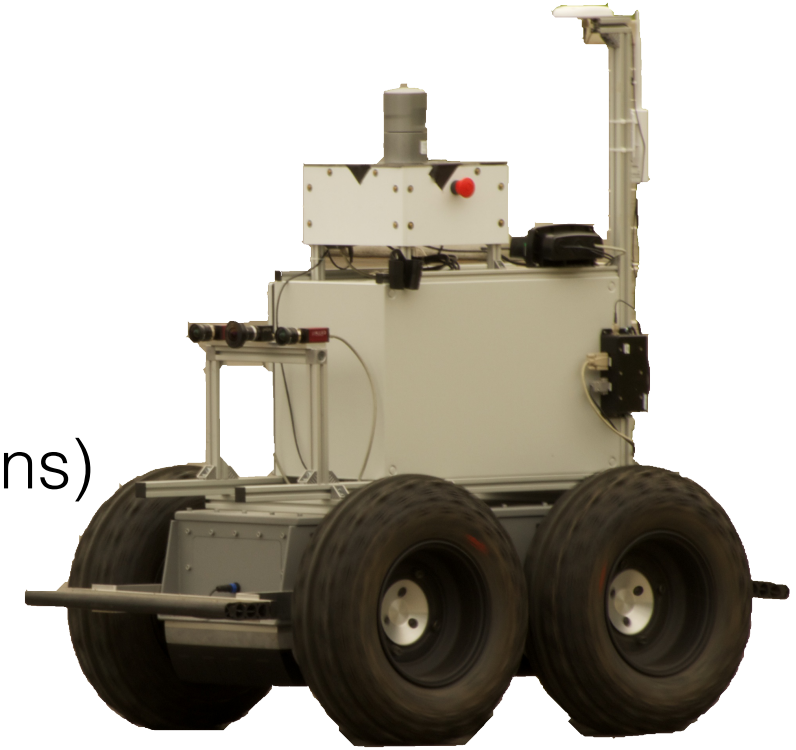
- Modeling tools relying on middleware
  - ROS Action lib (already mentionned)
  - GenoM (to be presented)
  - Orocos (real time, arms control)
  - BRICS
  - ROCK (based on Orocos RTT)
  - MARIE
  - MAUVE (ONERA)
  - etc

All have pros and cons. We focus on GenoM -> V&V

This is “Minnie”

Segway RMP 440
R&D platform… not an autonomous car but… definitely not a toy robot

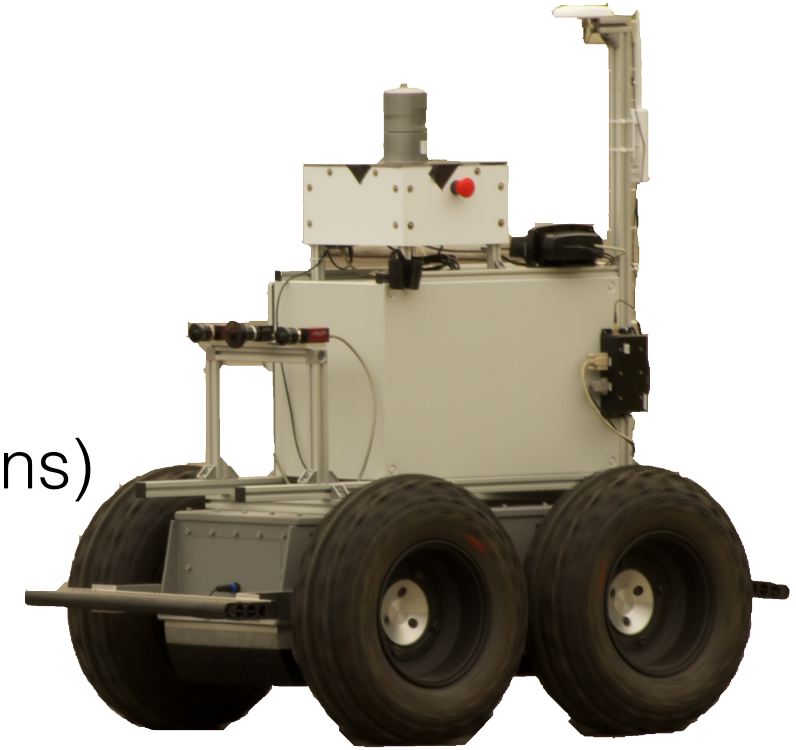
- Fast (up to 8 m/s)
- GPS
- Gyro (measures theta/wz)
- IMU (angular velocities and accelerations)
- 2 recent CPUs (but I only use one)



This is “Minnie”

Segway RMP 440
R&D platform… not an autonomous car but… definitely not a toy robot

- Fast (up to 8 m/s)
- GPS
- Gyro (measures theta/wz)
- IMU (angular velocities and accelerations)
- 2 recent CPUs (but I only use one)

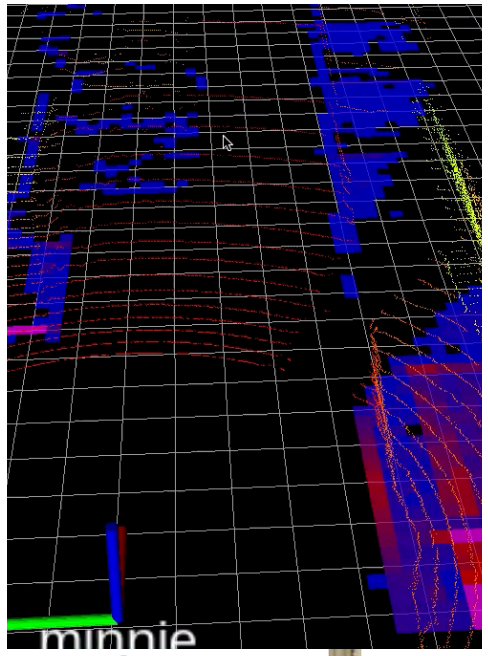


Let’s focus on Minnie’s velodyne

- Velodyne lidar sensor
  - they are used on most "autonomous cars” (except Tesla)
  - Point Cloud: 64K 3D points at 10 Hz…
  - … at 5 m/s, Minnie has moved 50cm while scanning
  - you need to register where you are while scanning
  - rebuild the corrected scan

KEY FEATURES

- Dual Returns
- ± 2 cm accuracy
- 1kg (plus 0.3kg for cabeling)
- 32 Channels
- 80m-100m Range
- Up to ~1.39 Million Points per Second
- 360° Horizontal FOV
- +10° to -30° Vertical FOV
- Low Power Consumption
- Rugged Design

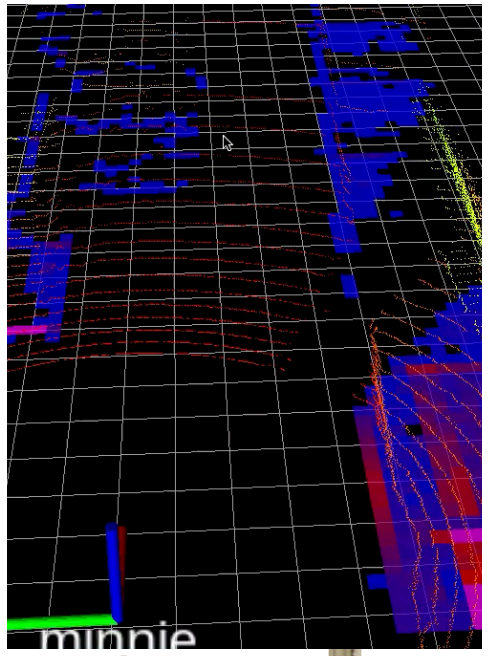


Let’s focus on Minnie’s velodyne

- Velodyne lidar sensor
  - they are used on most "autonomous cars” (except Tesla)
  - Point Cloud: 64K 3D points at 10 Hz…
  - … at 5 m/s, Minnie has moved 50cm while scanning
  - you need to register where you are while scanning
  - rebuild the corrected scan

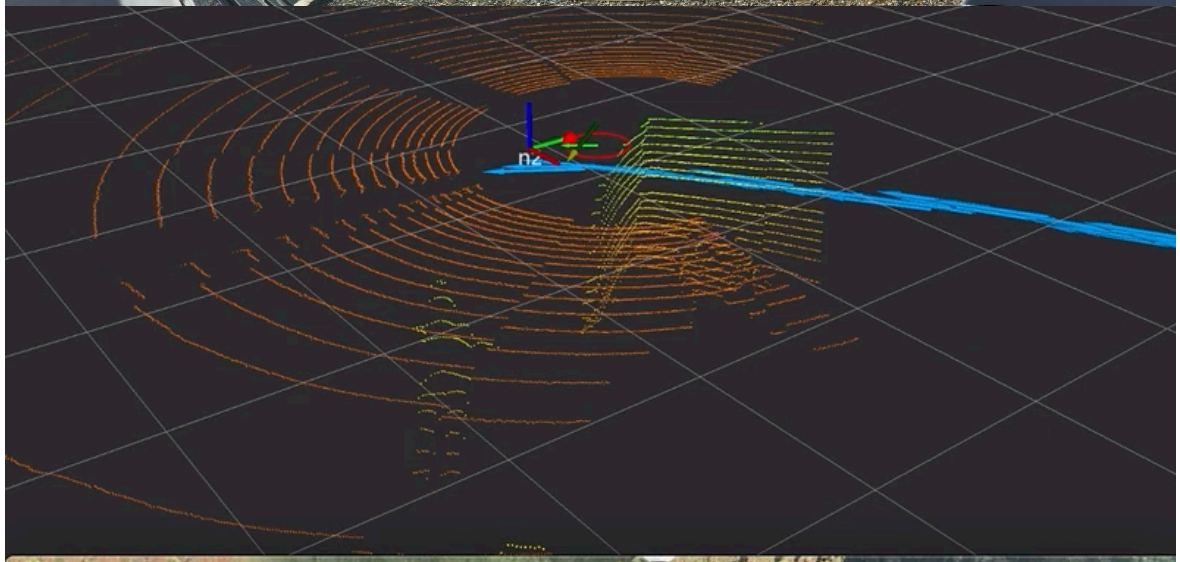
KEY FEATURES

- Dual Returns
- ± 2 cm accuracy
- 1kg (plus 0.3kg for cabeling)
- 32 Channels
- 80m-100m Range
- Up to ~1.39 Million Points per Second
- 360° Horizontal FOV
- +10° to -30° Vertical FOV
- Low Power Consumption
- Rugged Design



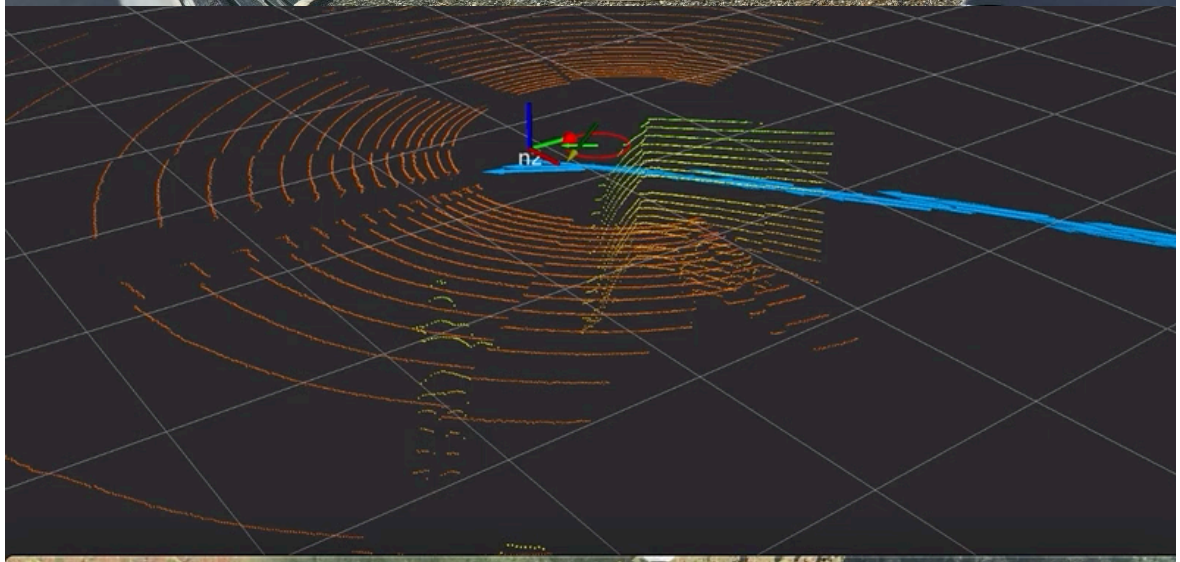
What do we use Minnie for?

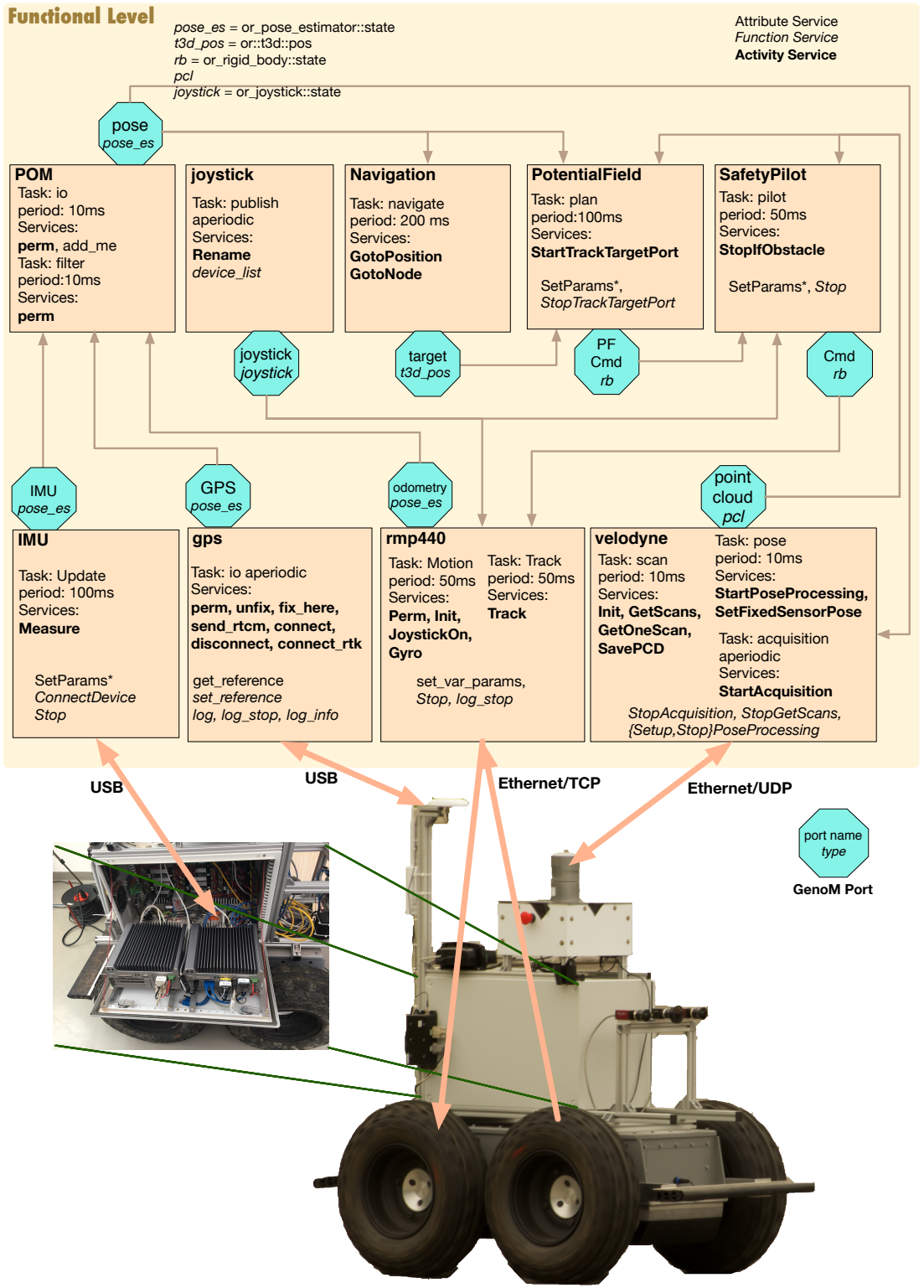
- Patrolling the parking lot in the lanes/ crossings graph
- without scratching my colleague's car (parked or moving)
- without bumping in people

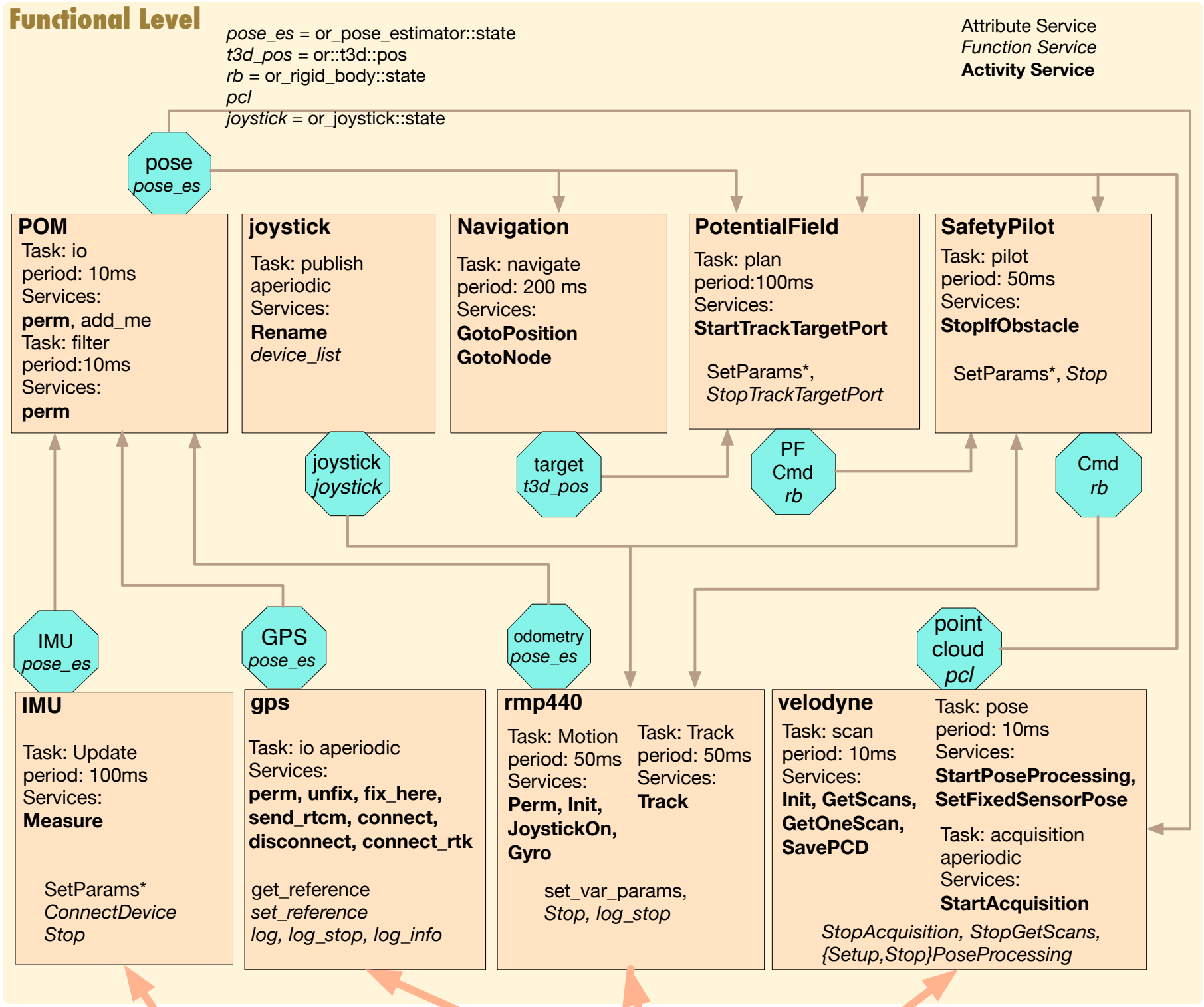


What do we use Minnie for?

- Patrolling the parking lot in the lanes/ crossings graph
- without scratching my colleague's car (parked or moving)
- without bumping in people



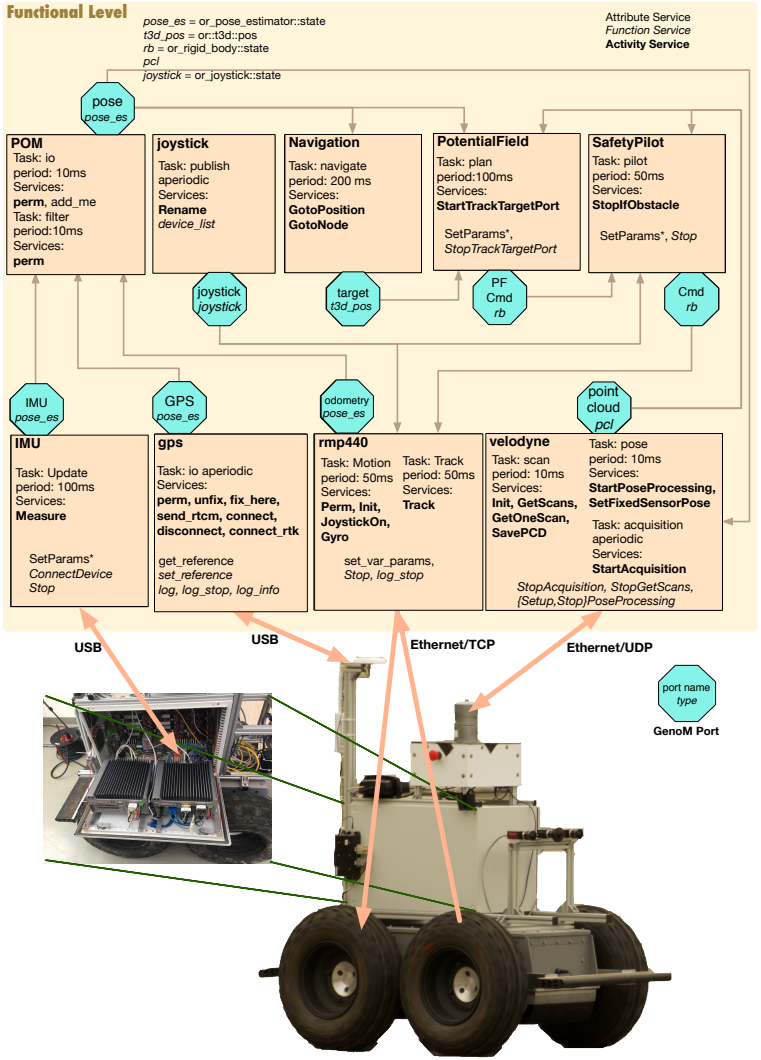

Functional Level
pose_es = or_pose_estimator::state
t3d_pos = or::t3d::pos
rb = or_rigid_body::state
pcl
joystick = or_joystick::state
Attribute Service
Function Service
Activity Service
pose
pose_es
POM
Task: io
period: 10ms
Services:
perm, add_me
Task: filter
period:10ms
Services:
perm
joystick
Task: publish
aperiodic
Services:
Rename
device_list
Navigation
Task: navigate
period: 200 ms
Services:
GotoPosition
GotoNode
PotentialField
Task: plan
period:100ms
Services:
StartTrackTargetPort
SetParams*,
StopTrackTargetPort
SafetyPilot
Task: pilot
period: 50ms
Services:
StopIfObstacle
SetParams*, Stop
joystick
joystick
target
t3d_pos
PF
Cmd
rb
Cmd
rb
IMU
pose_es
GPS
pose_es
odometry
pose_es
point
cloud
pcl
IMU
Task: Update
period: 100ms
Services:
Measure
SetParams*
ConnectDevice
Stop
gps
Task: io aperiodic
Services:
perm, unfix, fix_here,
send_rtcm, connect,
disconnect, connect_rtk
get_reference
set_reference
log, log_stop, log_info
rmp440
Task: Motion
period: 50ms
Services:
Perm, Init,
JoystickOn,
Gyro
Task: Track
period: 50ms
Services:
Track
set_var_params,
Stop, log_stop
velodyne
Task: scan
period: 10ms
Services:
Init, GetScans,
GetOneScan,
SavePCD
Task: pose
period: 10ms
Services:
StartPoseProcessing,
SetFixedSensorPose
Task: acquisition
aperiodic
Services:
StartAcquisition
StopAcquisition, StopGetScans,
{Setup,Stop}PoseProcessing
USB
USB
Ethernet/TCP
Ethernet/UDP
port name
type
GenoM Port


Functional Level
pose_es = or_pose_estimator::state
t3d_pos = or::t3d::pos
rb = or_rigid_body::state
pcl
joystick = or_joystick::state
Attribute Service
Function Service
Activity Service
pose
pose_es
POM
Task: io
period: 10ms
Services:
perm, add_me
Task: filter
period:10ms
Services:
perm
joystick
Task: publish
aperiodic
Services:
Rename
device_list
Navigation
Task: navigate
period: 200 ms
Services:
GotoPosition
GotoNode
PotentialField
Task: plan
period:100ms
Services:
StartTrackTargetPort
SetParams*,
StopTrackTargetPort
SafetyPilot
Task: pilot
period: 50ms
Services:
StopIfObstacle
SetParams*, Stop
joystick
joystick
target
t3d_pos
PF
Cmd
rb
Cmd
rb
IMU
pose_es
GPS
pose_es
odometry
pose_es
point
cloud
pcl
IMU
Task: Update
period: 100ms
Services:
Measure
SetParams*
ConnectDevice
Stop
gps
Task: io aperiodic
Services:
perm, unfix, fix_here,
send_rtcm, connect,
disconnect, connect_rtk
get_reference
set_reference
log, log_stop, log_info
rmp440
Task: Motion
period: 50ms
Services:
Perm, Init,
JoystickOn,
Gyro
Task: Track
period: 50ms
Services:
Track
set_var_params,
Stop, log_stop
velodyne
Task: scan
period: 10ms
Services:
Init, GetScans,
GetOneScan,
SavePCD
Task: pose
period: 10ms
Services:
StartPoseProcessing,
SetFixedSensorPose
Task: acquisition
aperiodic
Services:
StartAcquisition
StopAcquisition, StopGetScans,
{Setup,Stop}PoseProcessing
USB
Ethernet/TCP

Functional components specification

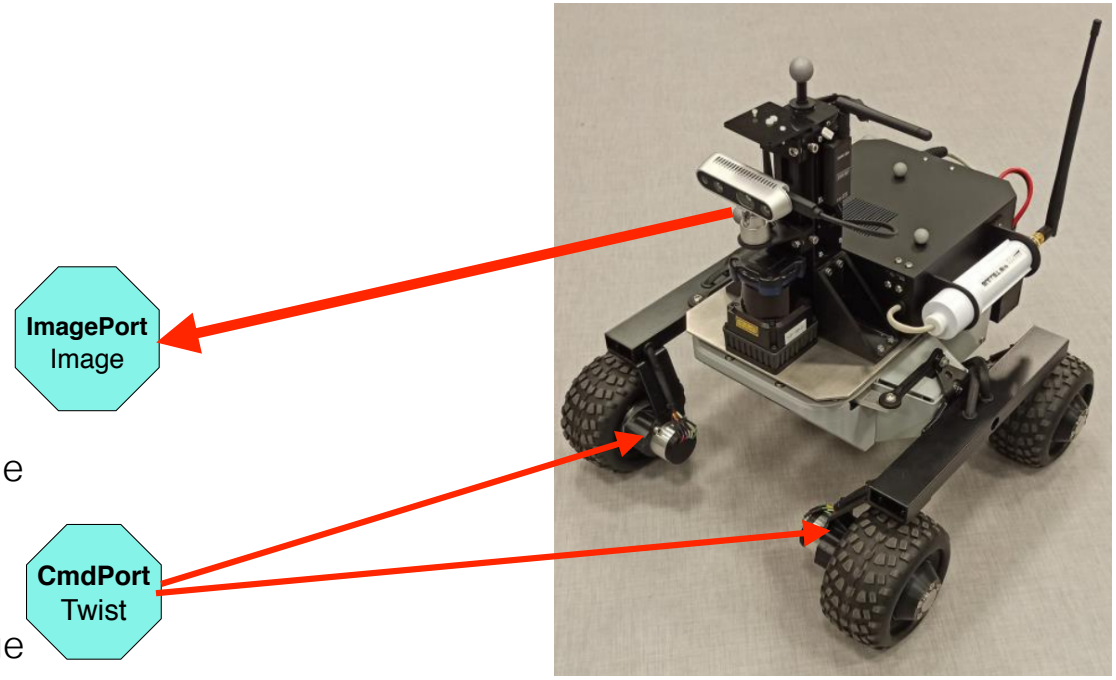
- Functional level : GenoM
- Modules
  - Services (control flow)
  - Ports (data flow)

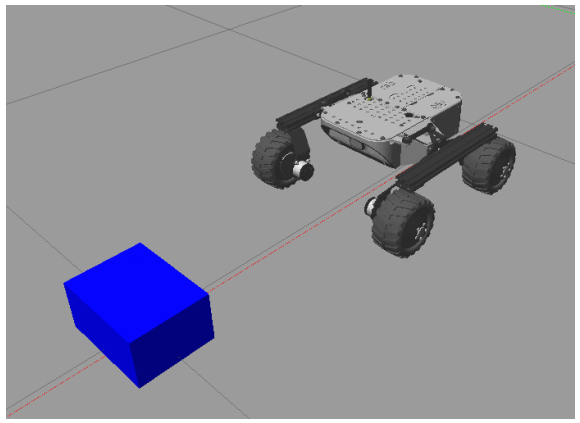
Specification: Model-Driven Software Engineering



GenoM3, simple example, ColorTrack Robot: CT_robot

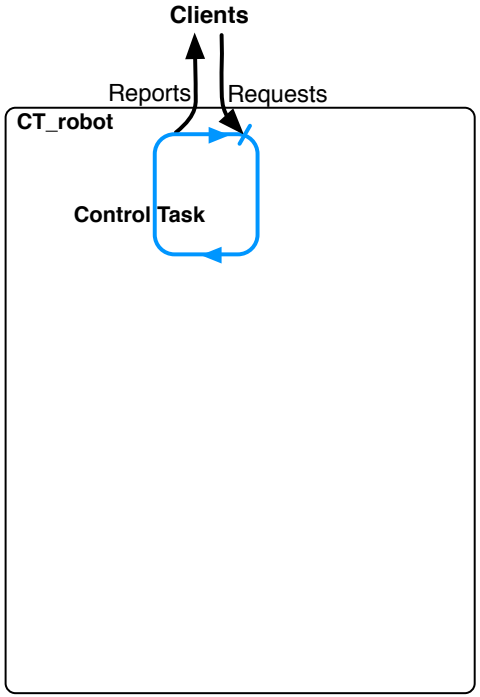
- CT_robot component (node):
  - has access to an **image** in **ImagePort**
  - provides a **ColorTrack** service to track a given **color (rgb)** in the **image** with a modifiable **threshold**
  - OpenCV simple primitives to find the **x,y** position of the barycenter of the **color** in the **image**
  - computes a **speed** command (vx, wz) to keep the **x, y** position centered in the image
  - exports the **speed** in a **CmdPort**
  - uses a modifiable **patrol_speed** to search the **color** when lost
  - the **ColorTrack** service reports the number of time we **lost** the **color**





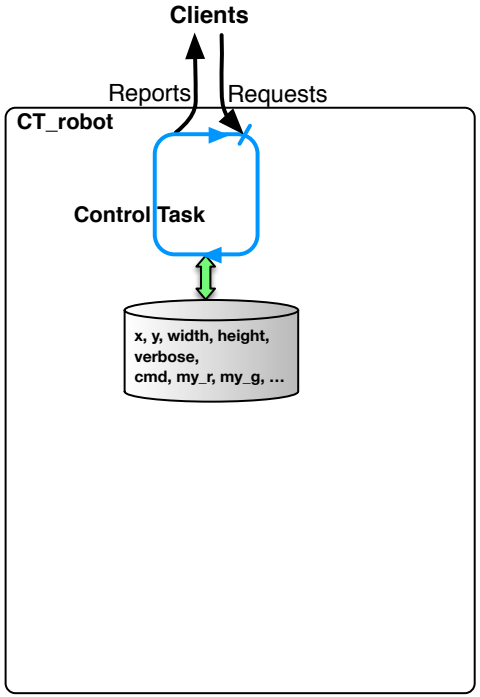
CT_robot .gen

```
/*
 * Copyright (c) 2019-2021 LAAS/CNRS
 *
 * Author: Felix Ingrand - LAAS/CNRS
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include "geometry.idl"    // Twist definition ROS masquerade geometry/Twist
#include "sensor.idl"      // Image definition ROS masquerade sensor/Image

/* ------------------------- MODULE DECLARATION ------------------------- */
component CT_robot {
  version "1.0";
  email       "felix@laas.fr";
  lang        "c";
  doc         "This module illustrates a simple GenoM module for the CT_robot ISAE UPSSITECH BE.";

  codels-require "roscpp,geometry_msgs,nav_msgs,opencv,cv_bridge";

  exception bad_image_port, bad_cmd_port, opencv_error, e_mem;
```

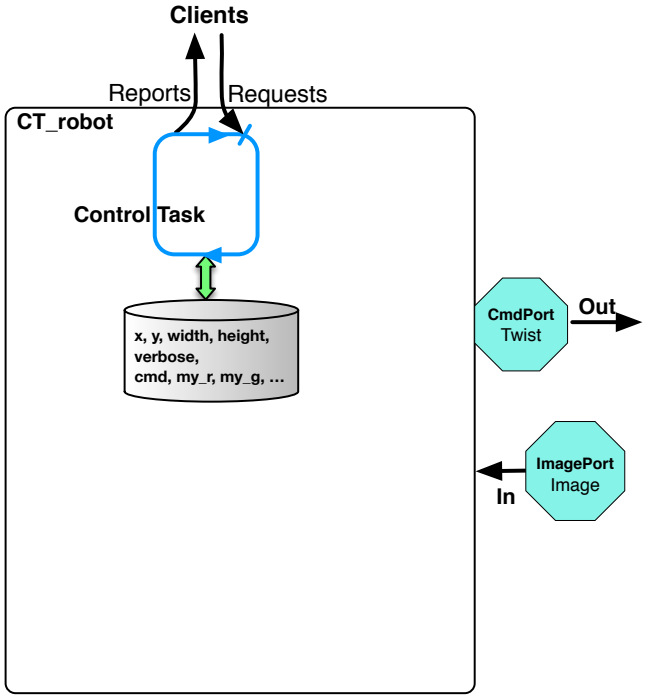


CT_robot .gen

```
struct cmd_s{
    double vx;       // The internal speed struct declaration
    double wz;
};
ids {
  long x,y;              // Position of the center of orange object in the image
  long width,height; // Size of the image
  long verbose;        // For logging verbosity
  cmd_s cmd;           // Internal speed command computed

  long my_r;           // Various values used by the image analysis algo.
  long my_g;
  long my_b;
  long my_seuil;
};
```

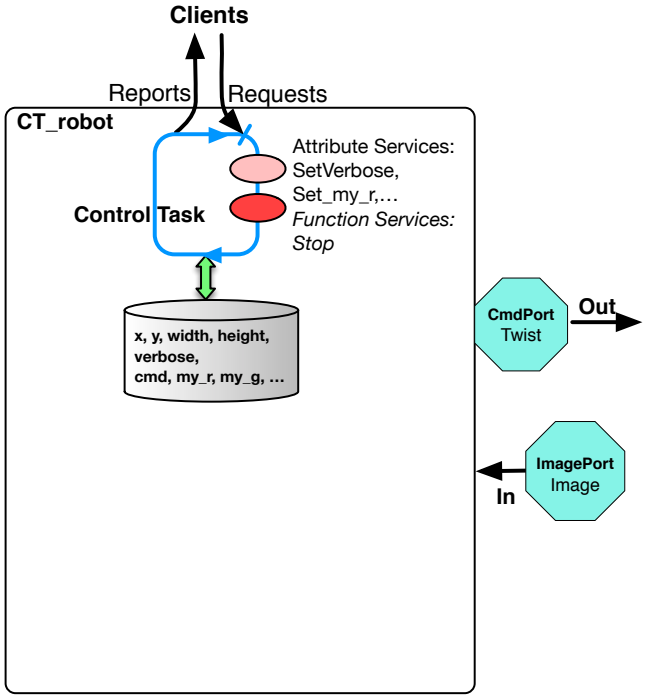


CT_robot .gen



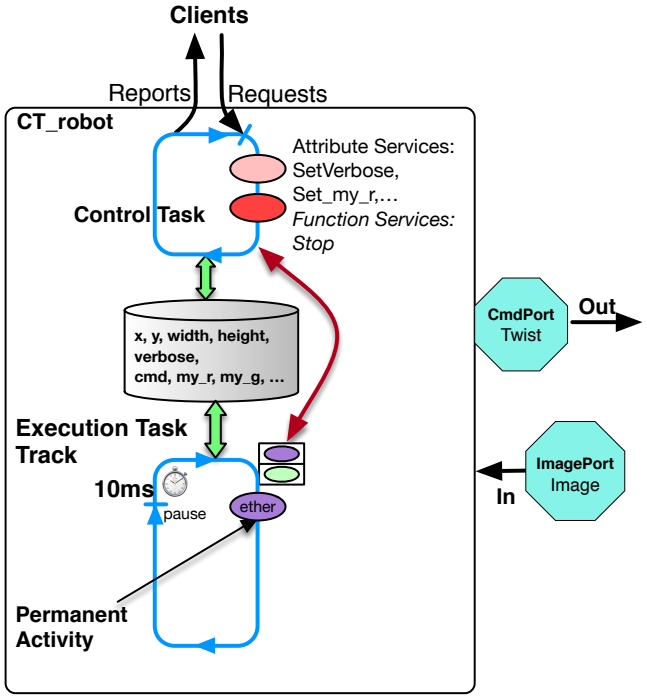

```
/* ------------ DEFINITION OF PORTS ------------ */
port in sensor::Image ImagePort {
  doc "The port ImagePort containing the image from the camera.";
};

port out geometry::Twist CmdPort { // CmdPort is the speed command port
                                   // (cmd (see above), in lower case, is the ids field)
  doc "The port CmdPort in which we put the speed at which we drive the robot.";
};
```

CT_robot .gen




```
/* ----------------- SERVICES DEFINITION: The attributes ---------------------- */
attribute SetVerbose(in verbose = 0 : "Verbose level")
{
  doc             "Set the verbose level.";
};

attribute Set_my_r(in my_r);
attribute Set_my_g(in my_g);
attribute Set_my_b(in my_b);
attribute Set_my_seuil(in my_seuil);

/* ----------------- SERVICES DEFINITION: The Functions ---------------------- */
function Stop()
{
  doc             "Stop the tracking.";
  codel    StopTrack(in verbose);  // This codel does not do anything... just here as an example.

  interrupts     ColorTrack;  // This field will force the transition to the stop codel in the
                              // ColorTrack activity automata
};
```

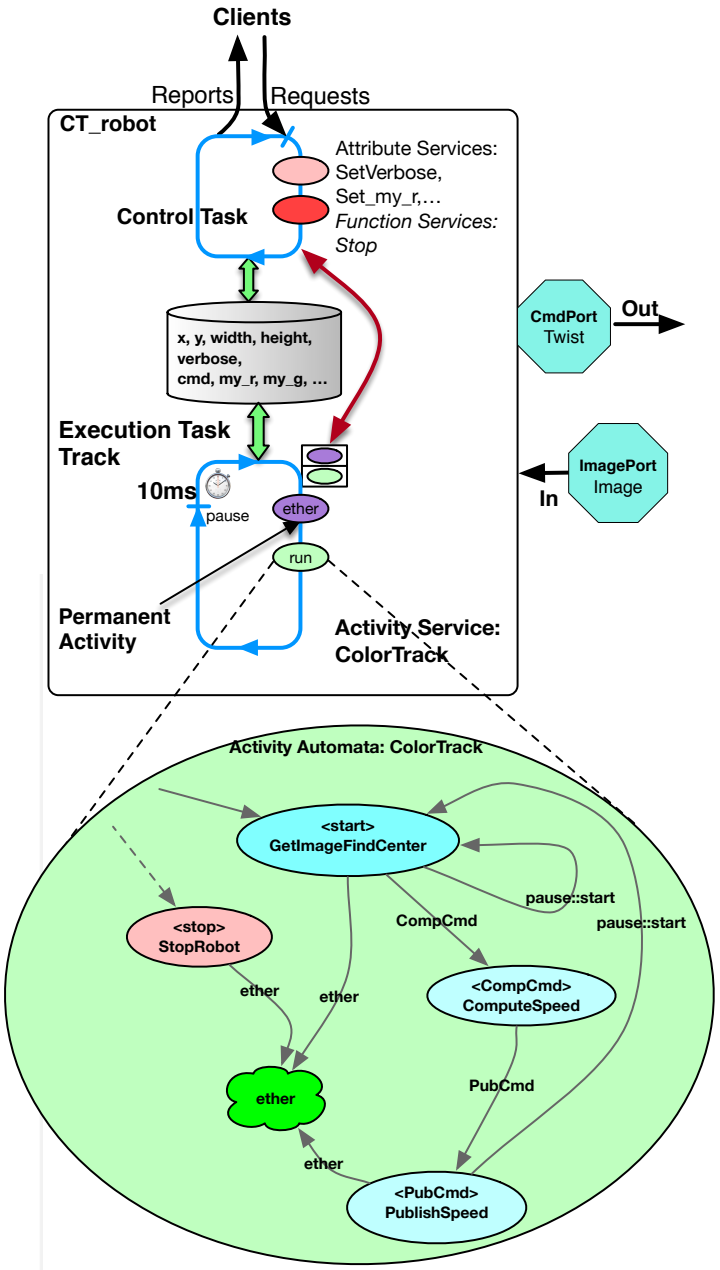
CT_robot .gen




```
/* ------------------ TASK DEFINITION ---------------------- */
task track {
  period              10 ms;      // fast, but we only process the image when it is new.
  codel <start>       InitIDS(port out CmdPort, ids out cmd, ids out x, ids out y) yield ether;
  codel <stop>        CleanIDS(port out CmdPort) yield ether;
};
```

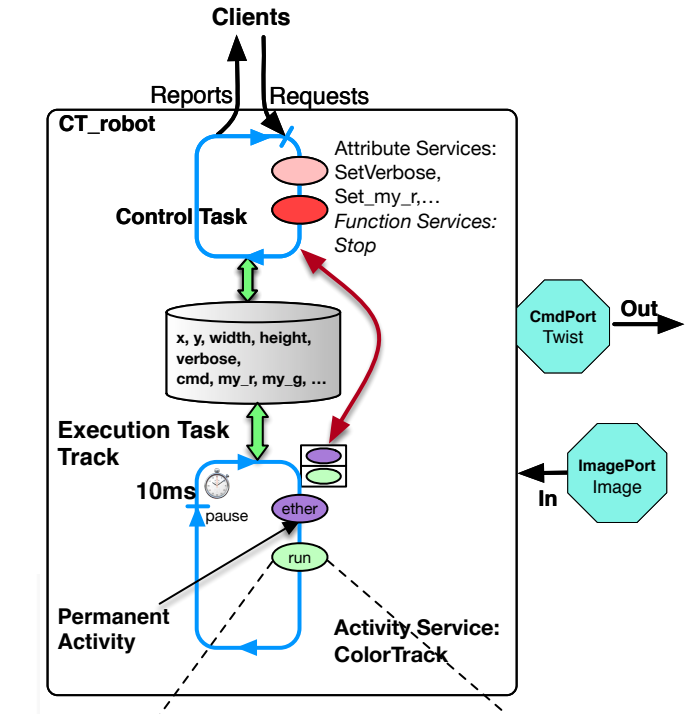
CT_robot .gen

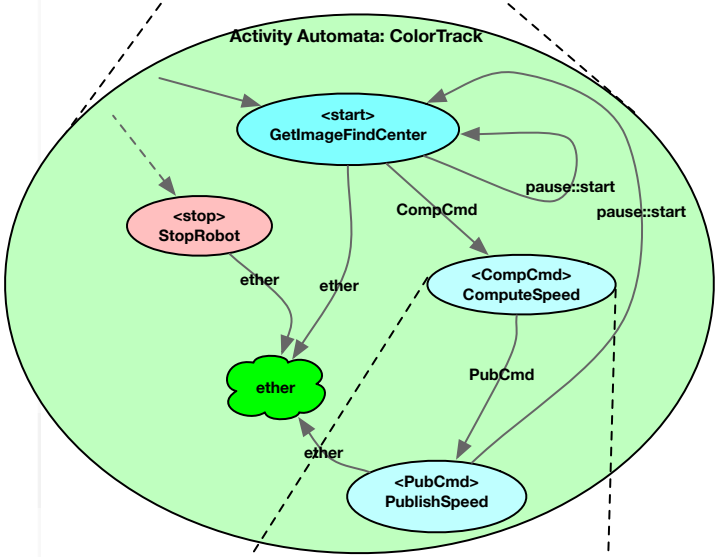
```
/* ---------- SERVICES DEFINITION: The activities ---------- */

activity ColorTrack () {
  doc           "Produce a twist so the robot follow the colored object.";

  task          track;      // The task in which ColorTrack will execute

  // Automata syntax
  // codel <state>    c_function({{ids|port|local}? {in|out|inout} arg_k,}*)
  //                  yield {pause::}?<state_i> {, {pause::}?<state_j>}*;
  // - ids/port/local is optional if arg_k name is not ambiguous,
  // - start, stop and ether are predefined states,
  // - yield pause::state means transition will wait the next task cycle to lead to state.

  codel <start>        GetImageFindCenter(port in ImagePort, ids in my_r, ids in my_g, ids in my_b,
                            ids in my_seuil, ids out x, ids out y,
                            ids out width, ids out height, ids in verbose)
                          yield pause::start, // no new image, wait next cycle of the exec task
                            CompCmd,          // found the image
                            ether;            // in case of error.
  codel <CompCmd> ComputeSpeed(ids in x, ids in y, ids in width, ids in height,
                            ids out cmd, ids in verbose)
                          yield PubCmd;
  codel <PubCmd>    PublishSpeed(ids in cmd, port out CmdPort)
                          yield pause::start, // Loop back at the start in the next cycle
                            ether;            // in case of error.
  codel <stop>         StopRobot(ids out cmd, port out CmdPort) // stop is a predefined state in GenoM
                          yield ether; // ColorTrack execution will jump to this state when the
                                       //service is interrupted

  throw         bad_cmd_port, bad_image_port, opencv_error; // Possible errors in the codels.
                                                            // Any will force execution to ether
  interrupts    ColorTrack; // Only one ColorTrack service running at a time
 };
};
```

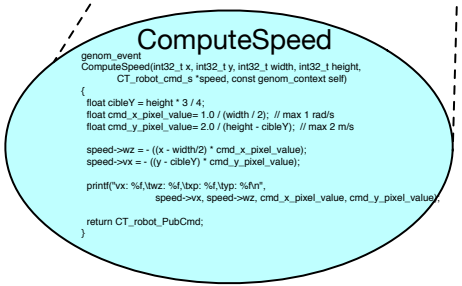


CT_robot .gen

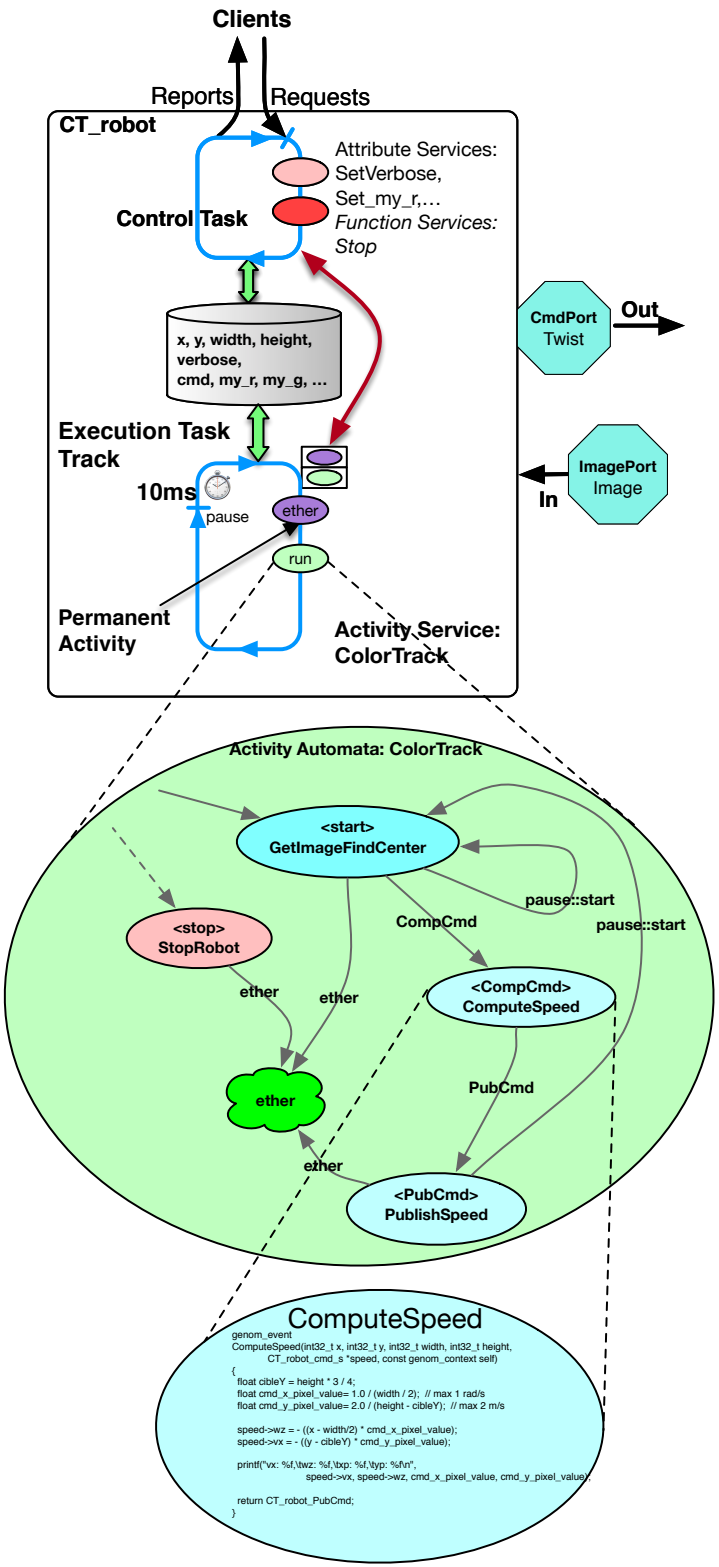
```
/* ---------- SERVICES DEFINITION: The activities ---------- */

activity ColorTrack () {
  doc           "Produce a twist so the robot follow the colored object.";

  task          track;      // The task in which ColorTrack will execute

  // Automata syntax
  // codel <state>    c_function({{ids|port|local}? {in|out|inout} arg_k,}*)
  //                  yield {pause::}?<state_i> {, {pause::}?<state_j>}*;
  // - ids/port/local is optional if arg_k name is not ambiguous,
  // - start, stop and ether are predefined states,
  // - yield pause::state means transition will wait the next task cycle to lead to state.

  codel <start>        GetImageFindCenter(port in ImagePort, ids in my_r, ids in my_g, ids in my_b,
                         ids in my_seuil, ids out x, ids out y,
                         ids out width, ids out height, ids in verbose)
                       yield pause::start, // no new image, wait next cycle of the exec task
                         CompCmd,          // found the image
                         ether;            // in case of error.
  codel <CompCmd> ComputeSpeed(ids in x, ids in y, ids in width, ids in height,
                         ids out cmd, ids in verbose)
                       yield PubCmd;
  codel <PubCmd>    PublishSpeed(ids in cmd, port out CmdPort)
                       yield pause::start, // Loop back at the start in the next cycle
                         ether;            // in case of error.
  codel <stop>         StopRobot(ids out cmd, port out CmdPort) // stop is a predefined state in GenoM
                         yield ether; // ColorTrack execution will jump to this state when the
                                      //service is interrupted

  throw         bad_cmd_port, bad_image_port, opencv_error; // Possible errors in the codels.
                                                            // Any will force execution to ether
  interrupts    ColorTrack; // Only one ColorTrack service running at a time
 };
};
```







CT_robot



CT_robot

IDS:

x, y, width, height,
verbose, cmd,
my_r, …

Task:

track 10ms

Services:

SetVerbose
Set_my_r, …
Stop
ColorTrack

CmdPort
Twist

ImagePort
Image

Attribute Service
Function Service
Activity Service

GenoMPort
Type

GenoM

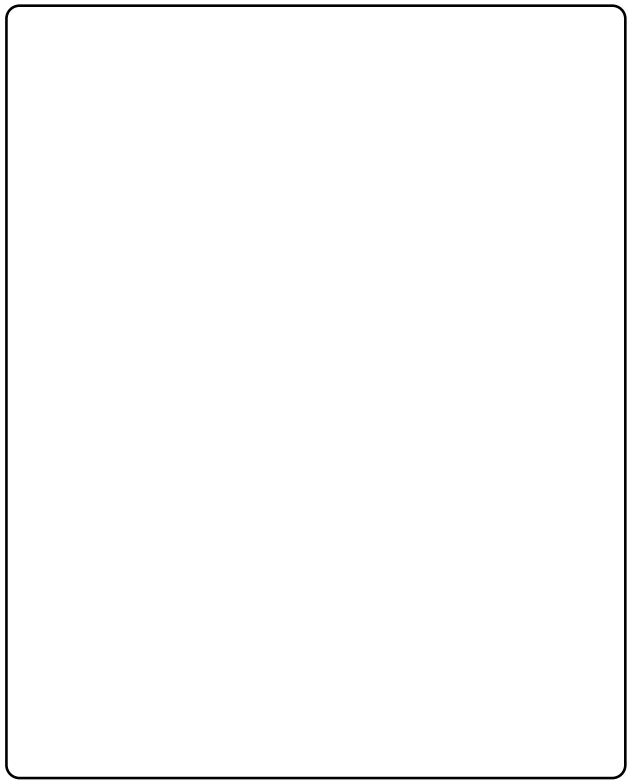
GenoM

To design a typical generic module which will be instantiated according to each specific module

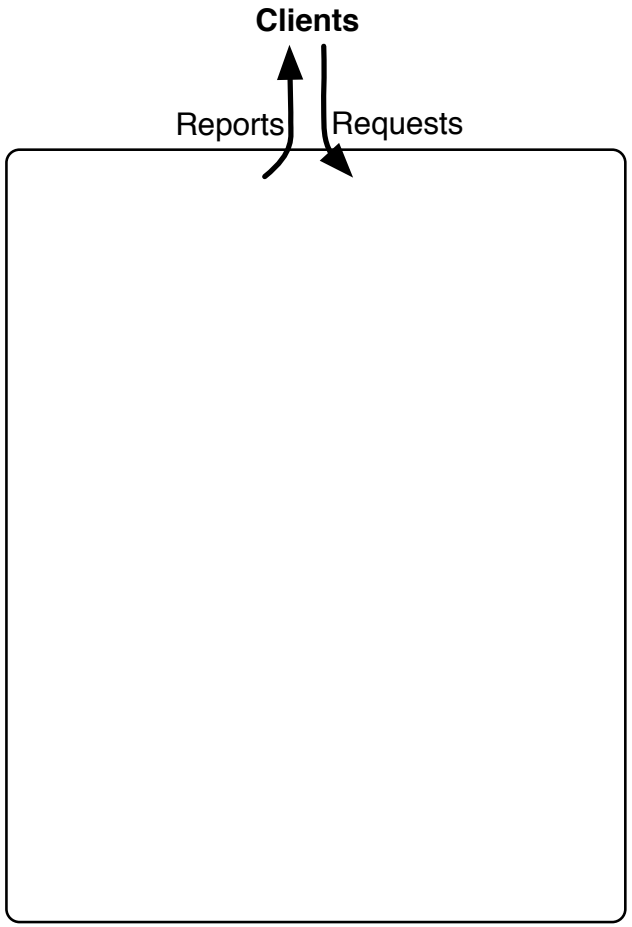
GenoM

To design a typical generic module which will be instantiated according to each specific module

- **a module is a program**



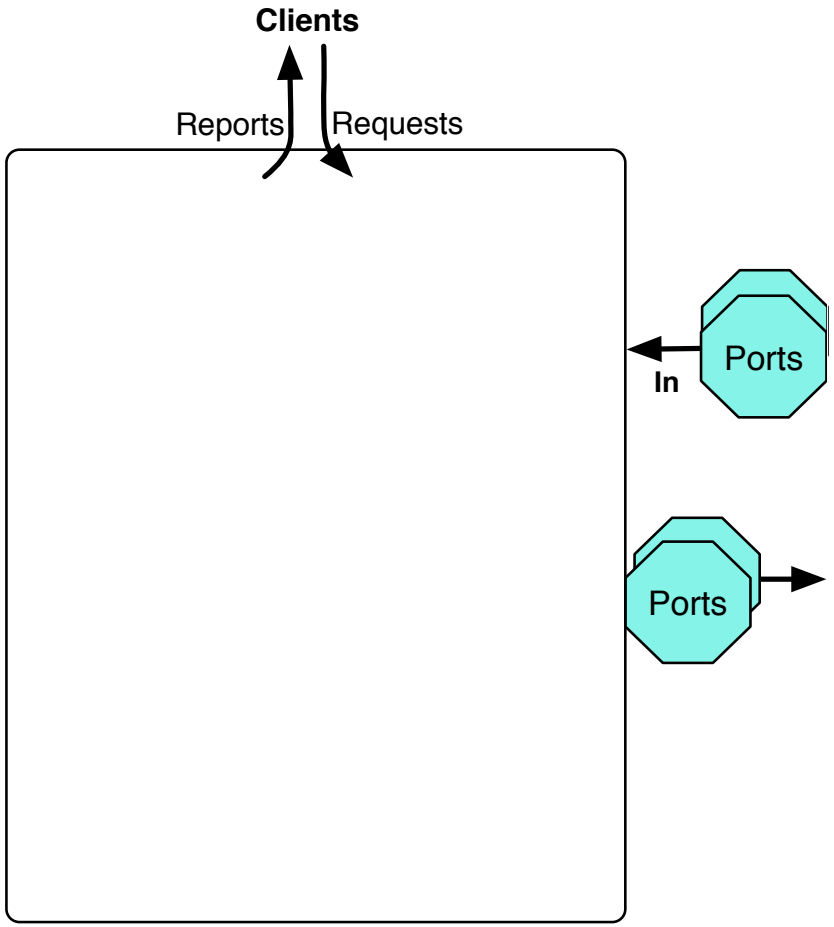
GenoM



To design a typical generic module which will be instantiated according to each specific module

- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**

GenoM



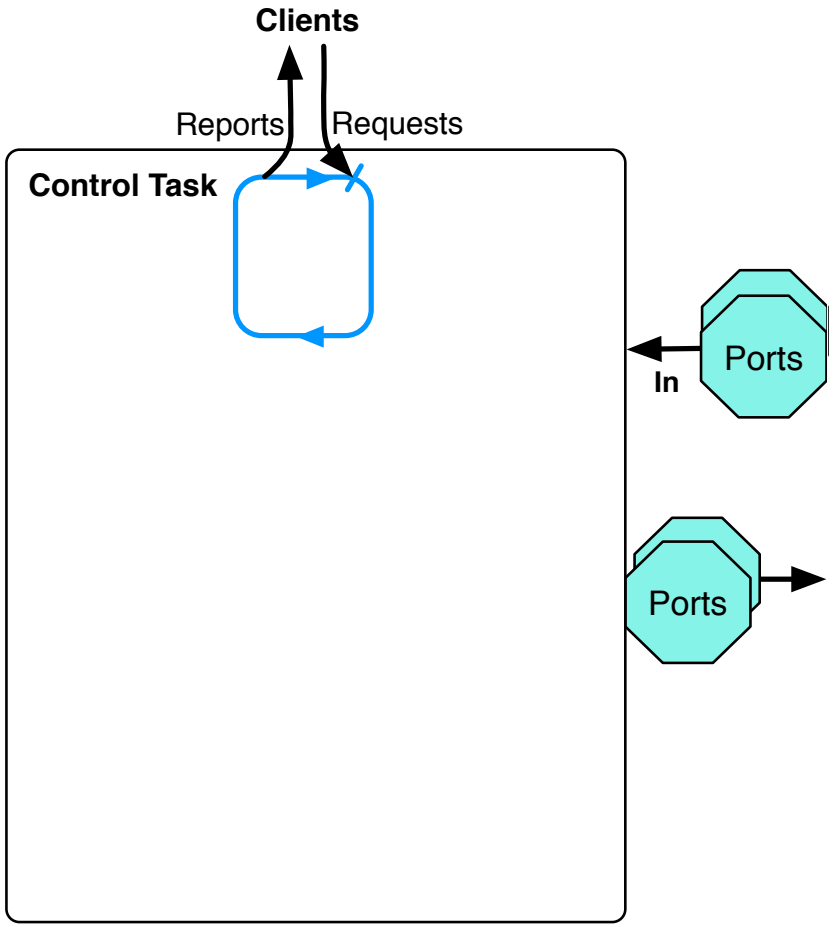
To design a typical generic module which will be instantiated according to each specific module

- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**

GenoM

To design a typical generic module which will be instantiated according to each specific module

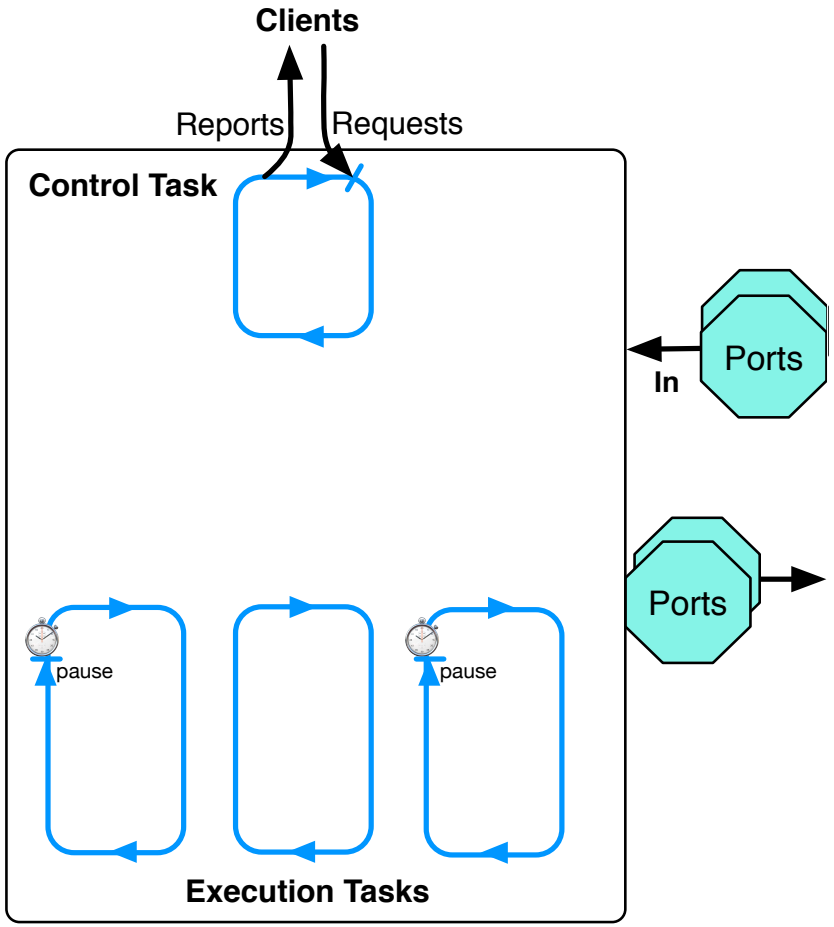
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**



GenoM

To design a typical generic module which will be instantiated according to each specific module

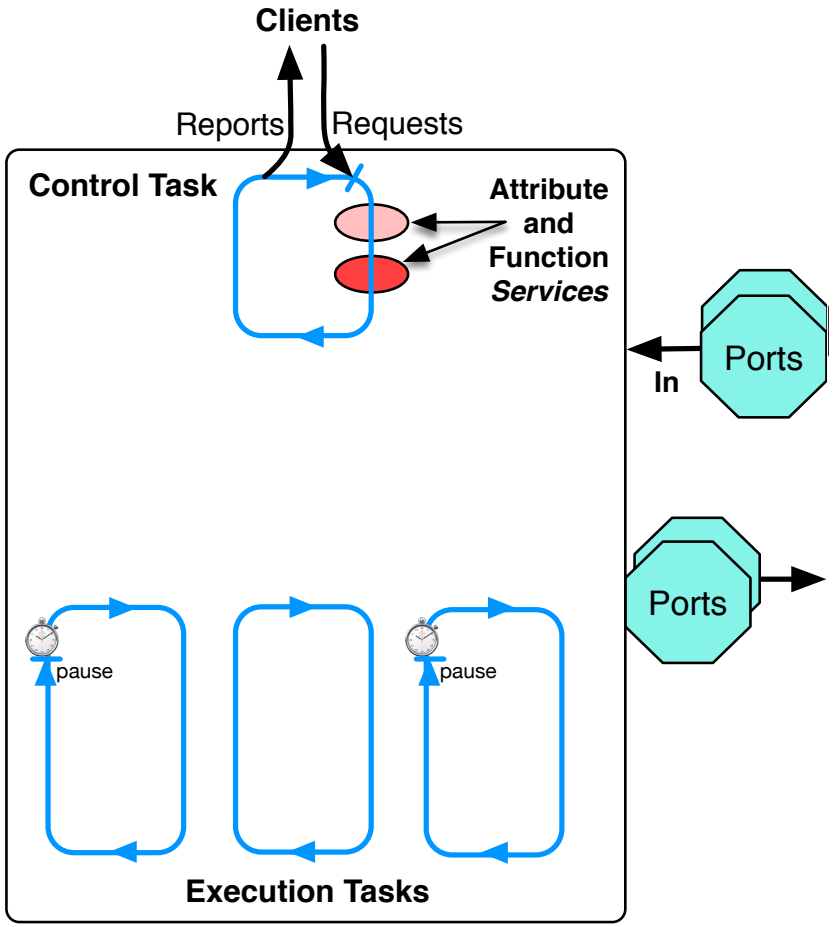
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**



GenoM

To design a typical generic module which will be instantiated according to each specific module

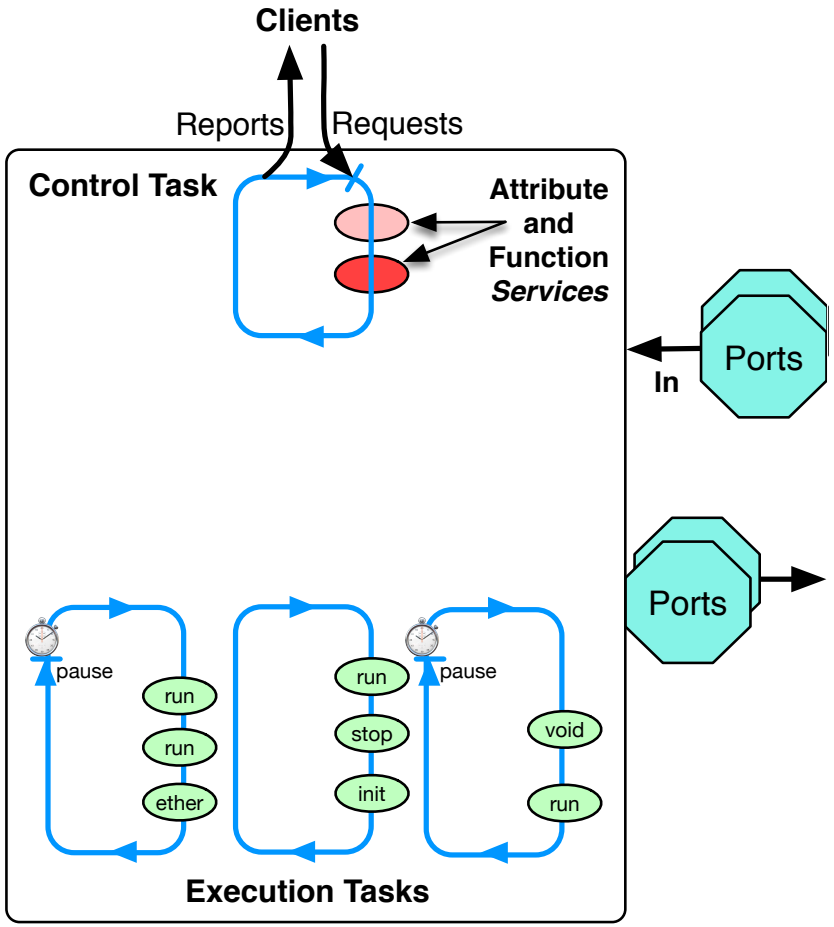
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**



GenoM

To design a typical generic module which will be instantiated according to each specific module

- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**
  - **and the executions task(s)**

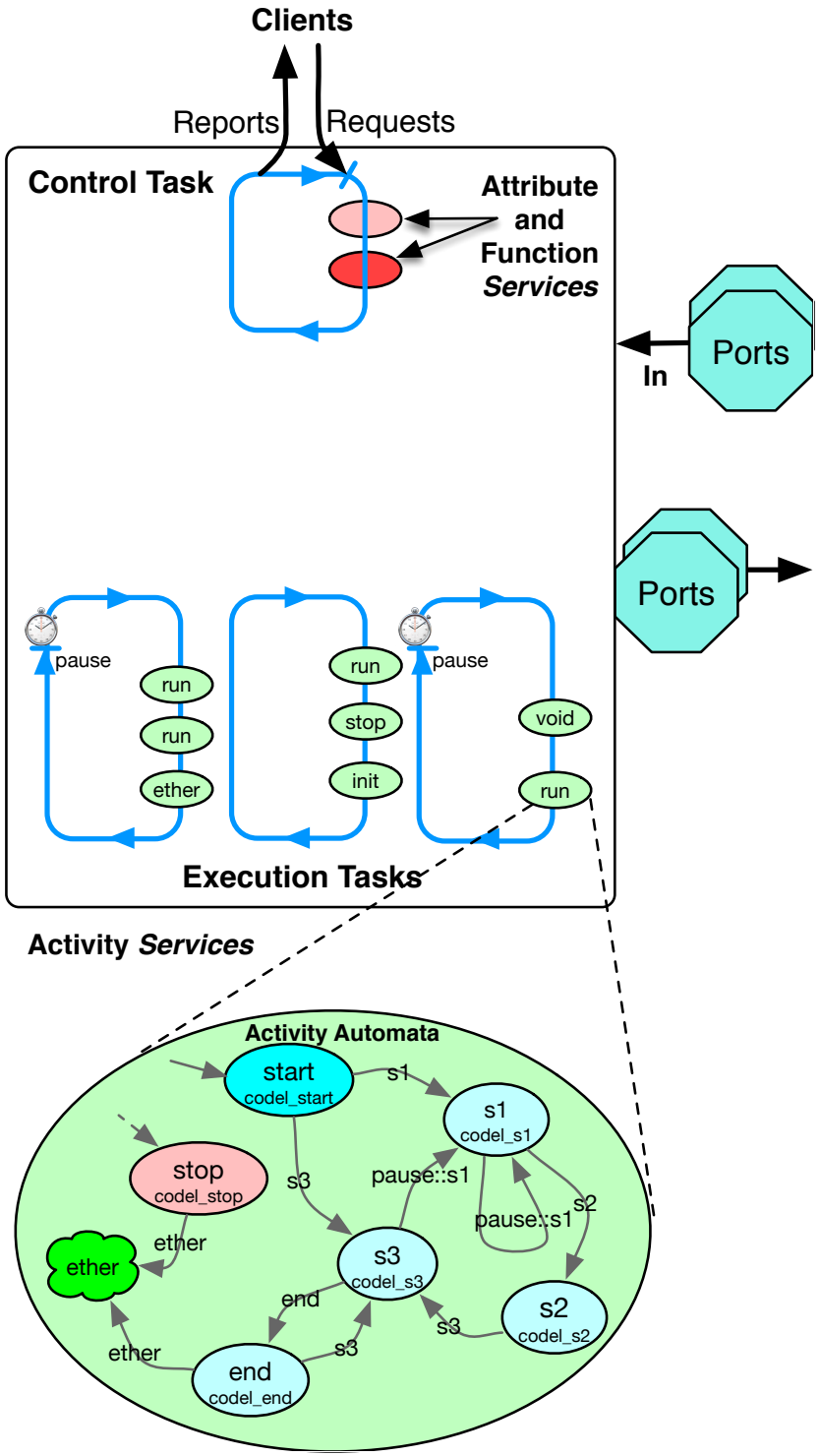


Activity *Services*

GenoM

To design a typical generic module which will be instantiated according to each specific module

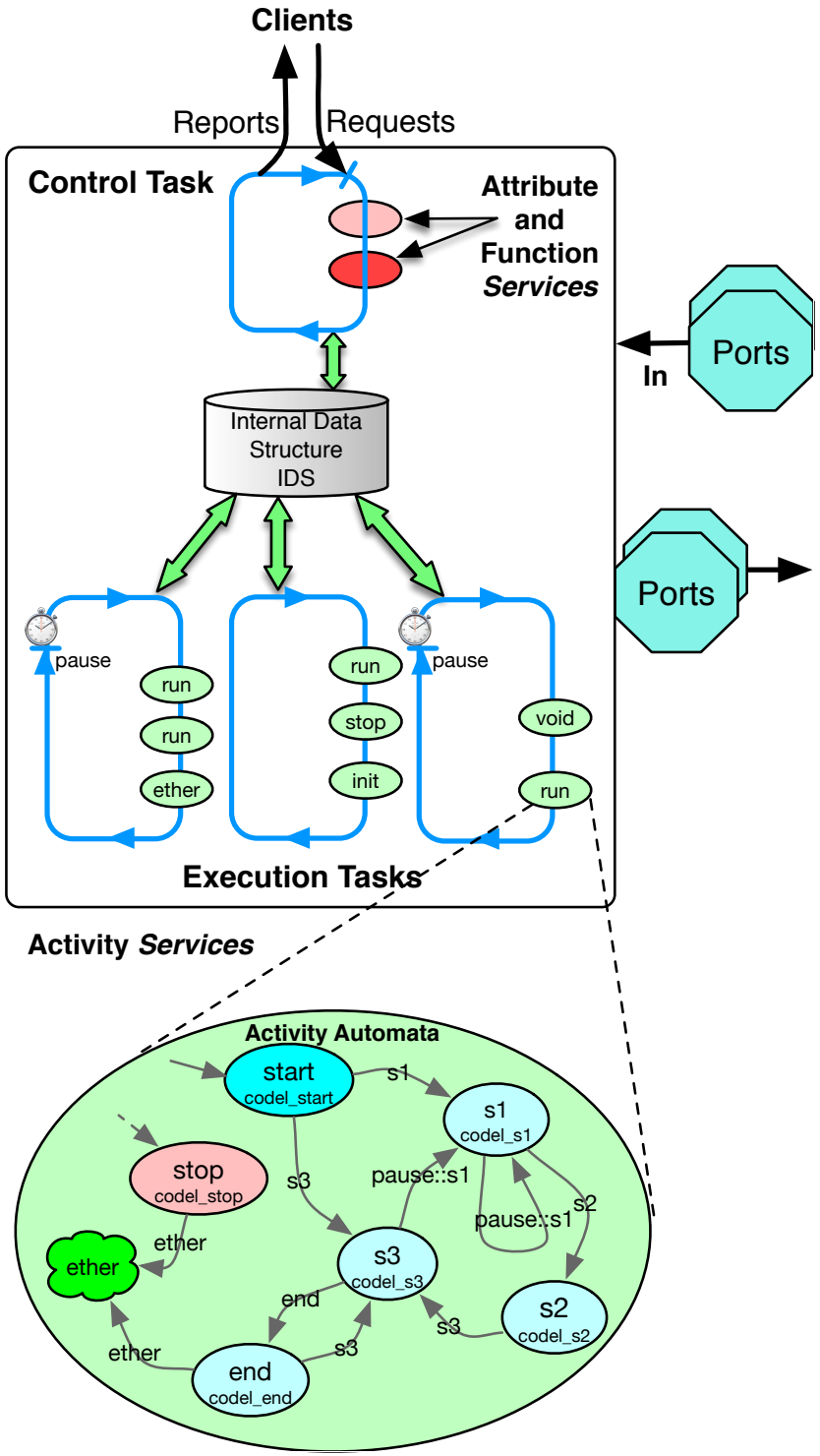
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**
  - **and the executions task(s)**
- **the slow one can have different steps (automata) to perform their processing**



GenoM

To design a typical generic module which will be instantiated according to each specific module

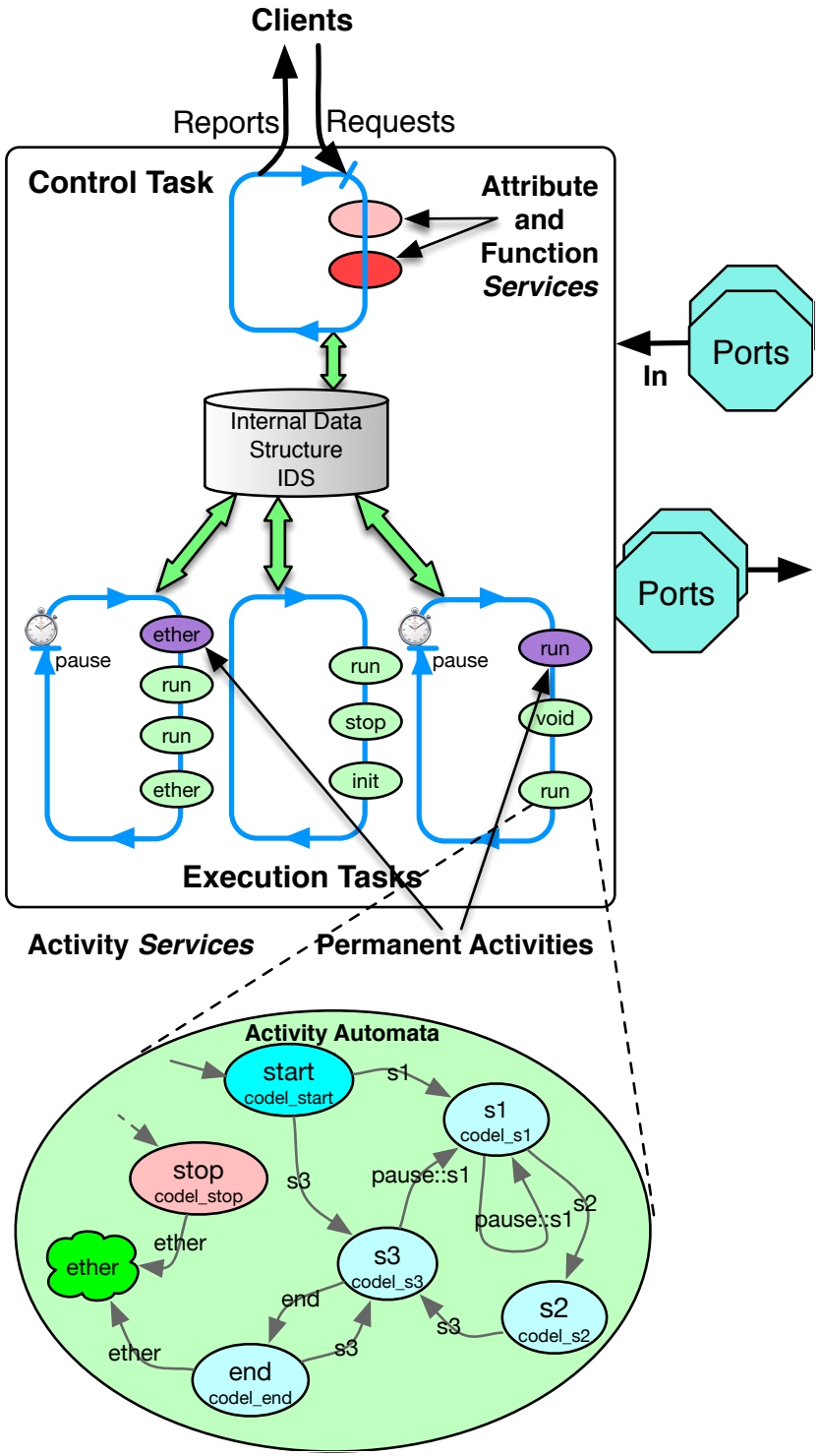
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**
  - **and the executions task(s)**
- **the slow one can have different steps (automata) to perform their processing**
- **services share a common data structure (IDS) for the need of their computation (parameters, computed values, internal state variables, etc)**



GenoM

To design a typical generic module which will be instantiated according to each specific module

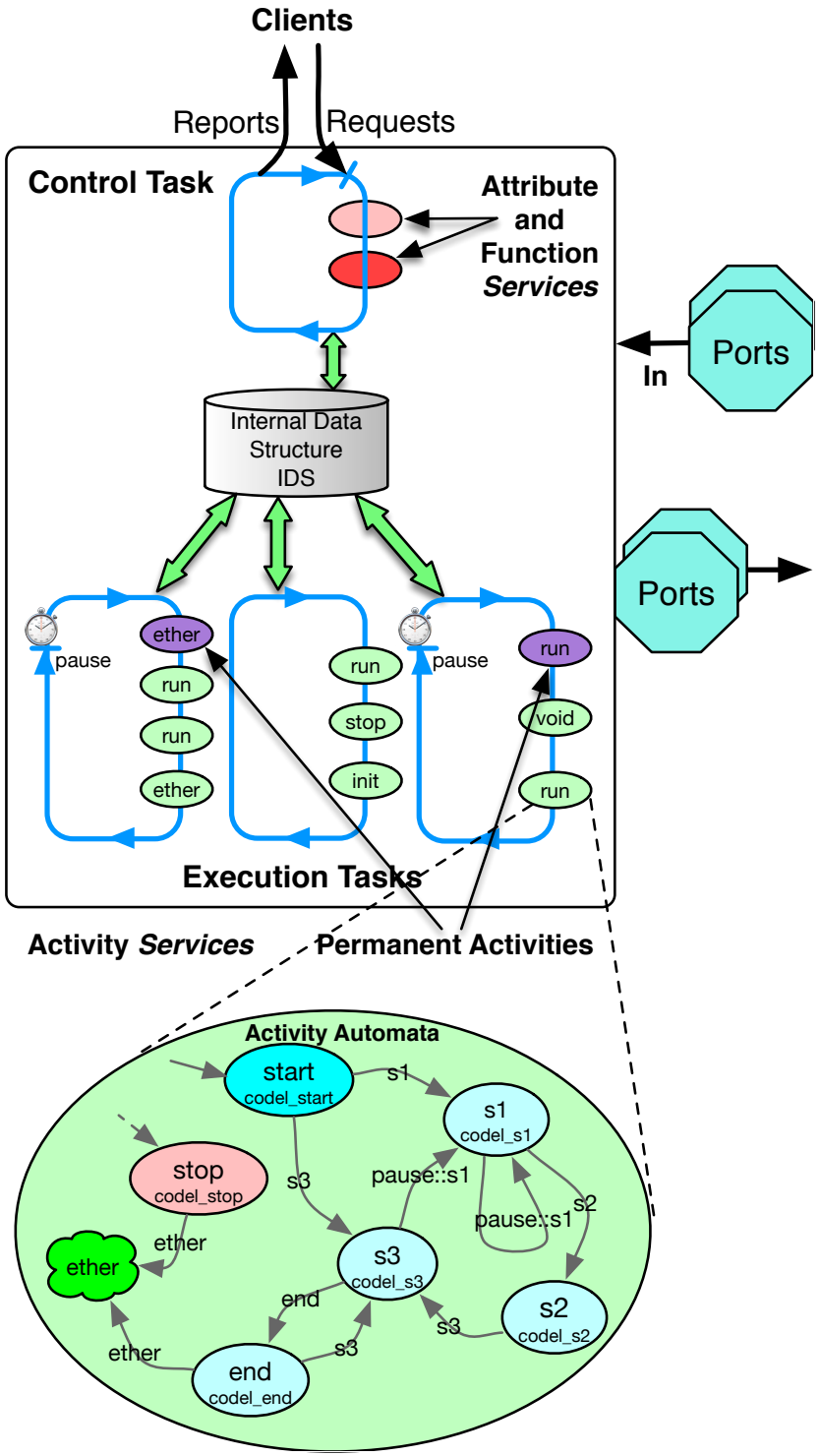
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**
  - **and the executions task(s)**
- **the slow one can have different steps (automata) to perform their processing**
- **services share a common data structure (IDS) for the need of their computation (parameters, computed values, internal state variables, etc)**
- **execution tasks may have a permanent activity**



GenoM

To design a typical generic module which will be instantiated according to each specific module

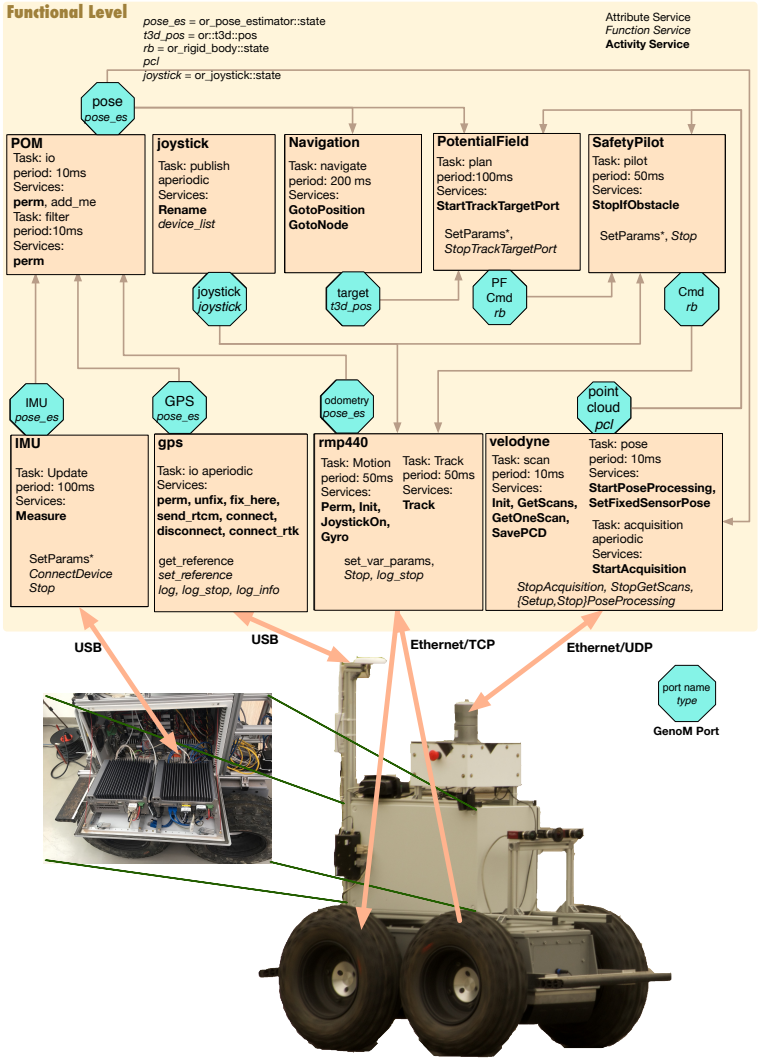
- **a module is a program**
- **a module has I/O**
  - **control: requests to start services/reports their results**
  - **data: ports in(to read external data) and out (to write external data)**
- **it supports a cyclic control task (aperiodic)**
- **and one or more cyclic execution tasks, periodic or aperiodic**
- **it provides services (fast and slow) to which we will associate C/C++ code**
  - **in the control task**
  - **and the executions task(s)**
- **the slow one can have different steps (automata) to perform their processing**
- **services share a common data structure (IDS) for the need of their computation (parameters, computed values, internal state variables, etc)**
- **execution tasks may have a permanent activity**
- **there may be exceptions, and incompatible services (they cannot run at the same time)**



Functional components specification

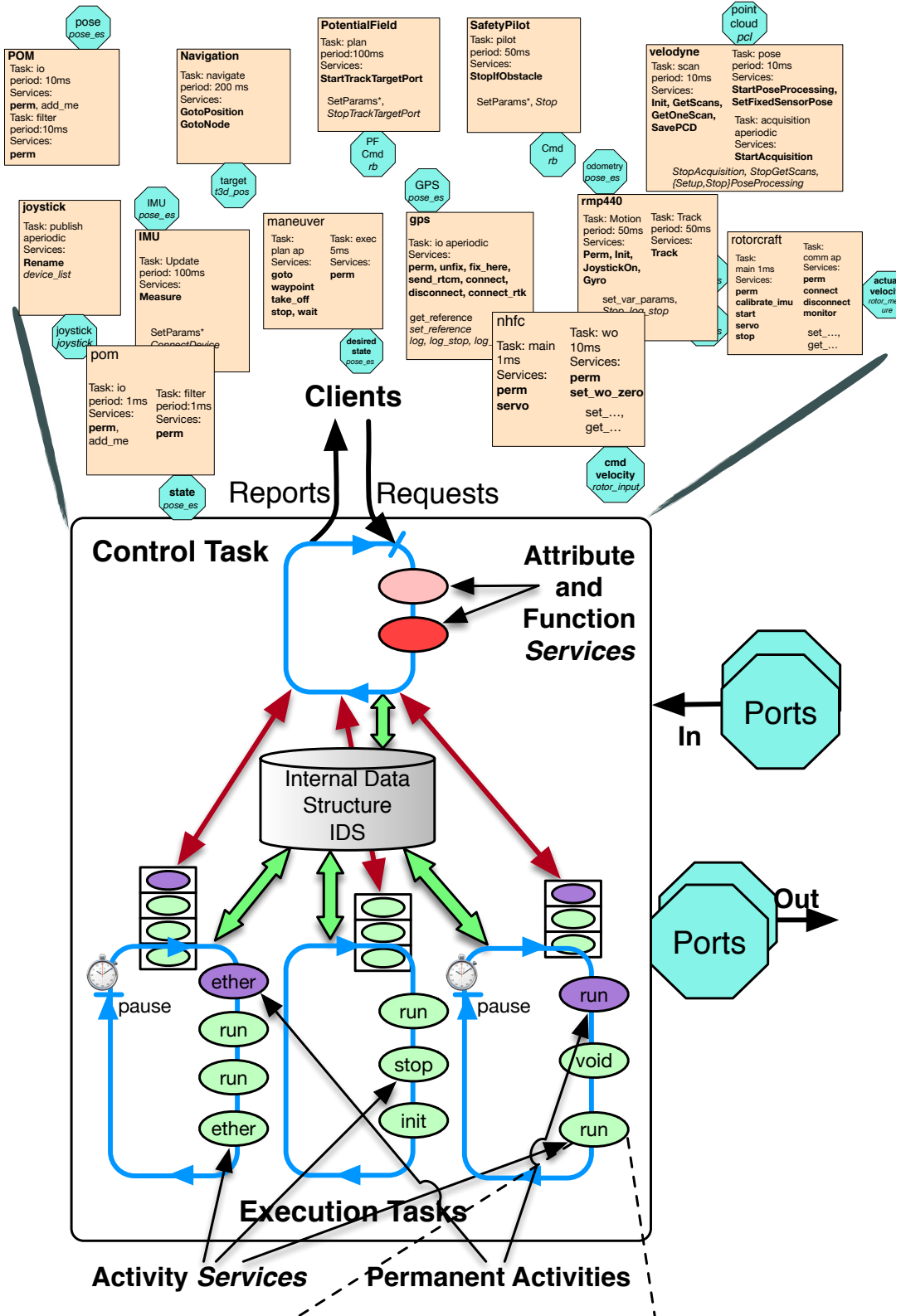
- Functional level : GenoM
- Modules
  - Services (control flow)
  - Ports (data flow)

Specification: Model-Driven Software Engineering



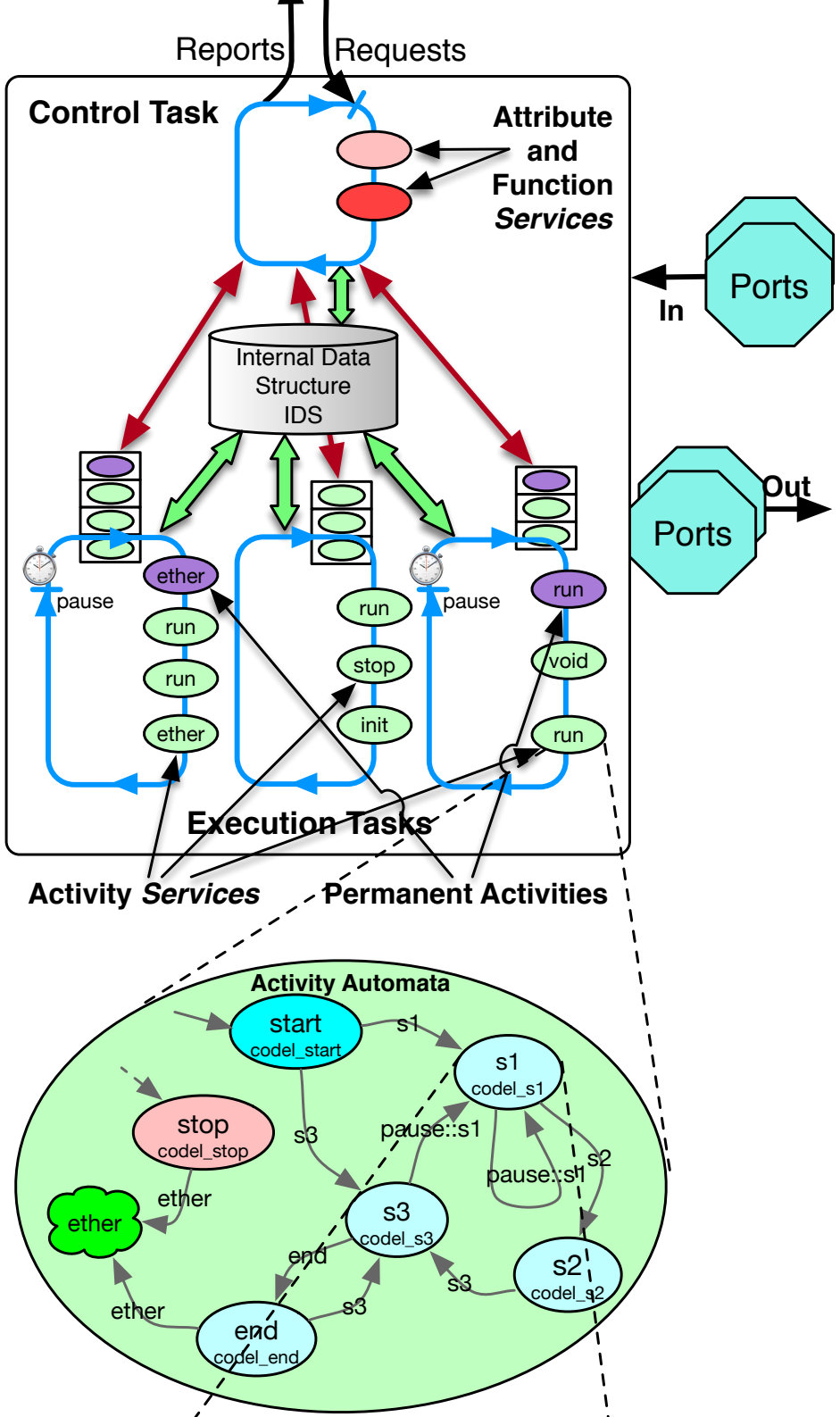
GenoM3 internal

- A program with I/O
  - control: requests to start services/reports their results
  - data: ports in (to import external data) and out (to export data)
- A cyclic event based control task (aperiodic)
- One or more cyclic execution tasks, periodic or aperiodic
- It provides services (short and long computation) to which we will associate C/C++ code
  - in the control task: attribute and function services (short)
  - and the executions task(s): activity services (long)
- services share a common Internal Data Structure for the needs of their computation (parameters, computed values, internal state variables, etc)



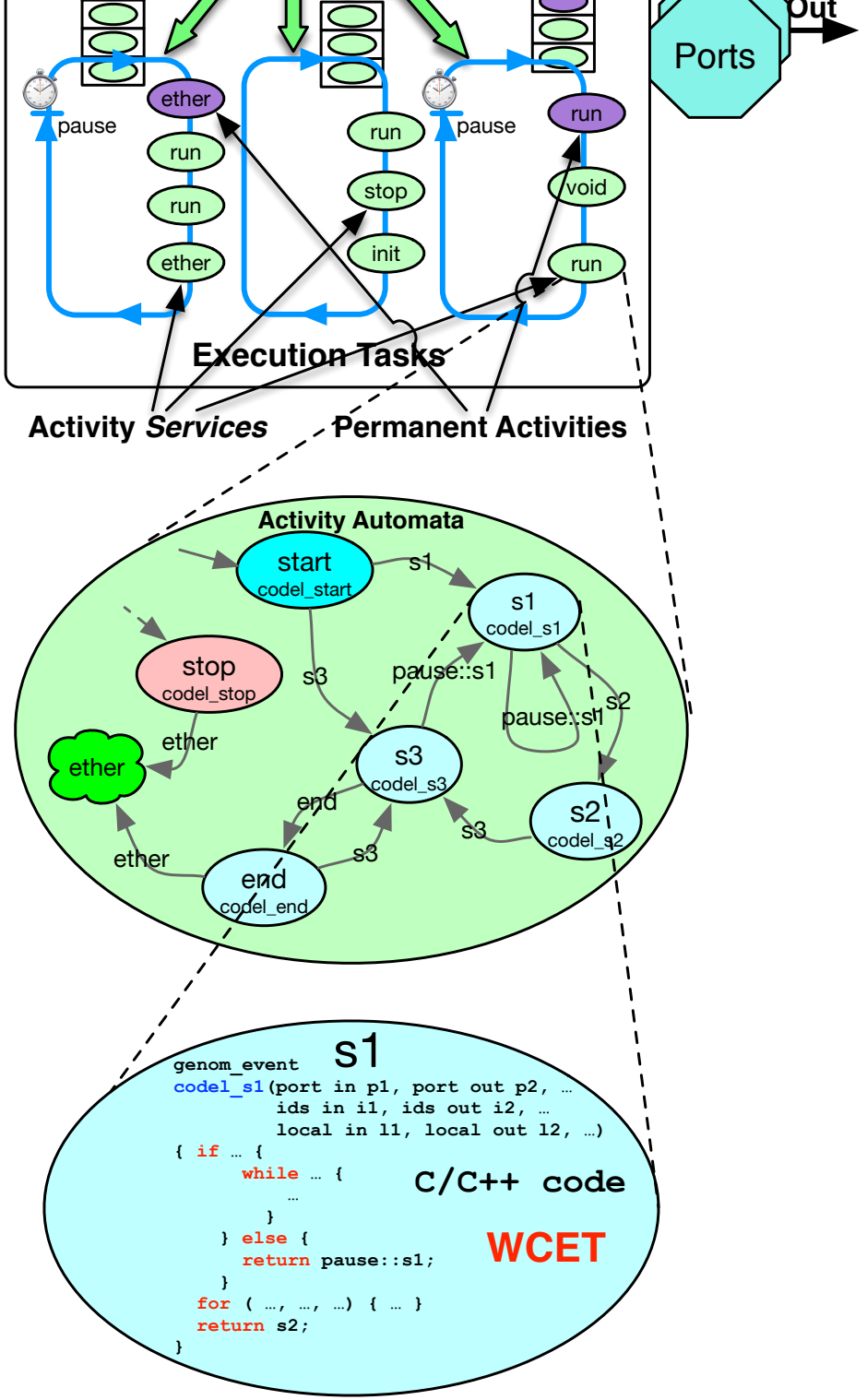
GenoM3 internal

- A program with I/O
  - control: requests to start services/reports their results
  - data: ports in (to import external data) and out (to export data)
- A cyclic event based control task (aperiodic)
- One or more cyclic execution tasks, periodic or aperiodic
- It provides services (short and long computation) to which we will associate C/C++ code
  - in the control task: attribute and function services (short)
  - and the executions task(s): activity services (long)
- services share a common Internal Data Structure for the needs of their computation (parameters, computed values, internal state variables, etc)
- activity services define automata to perform their processing



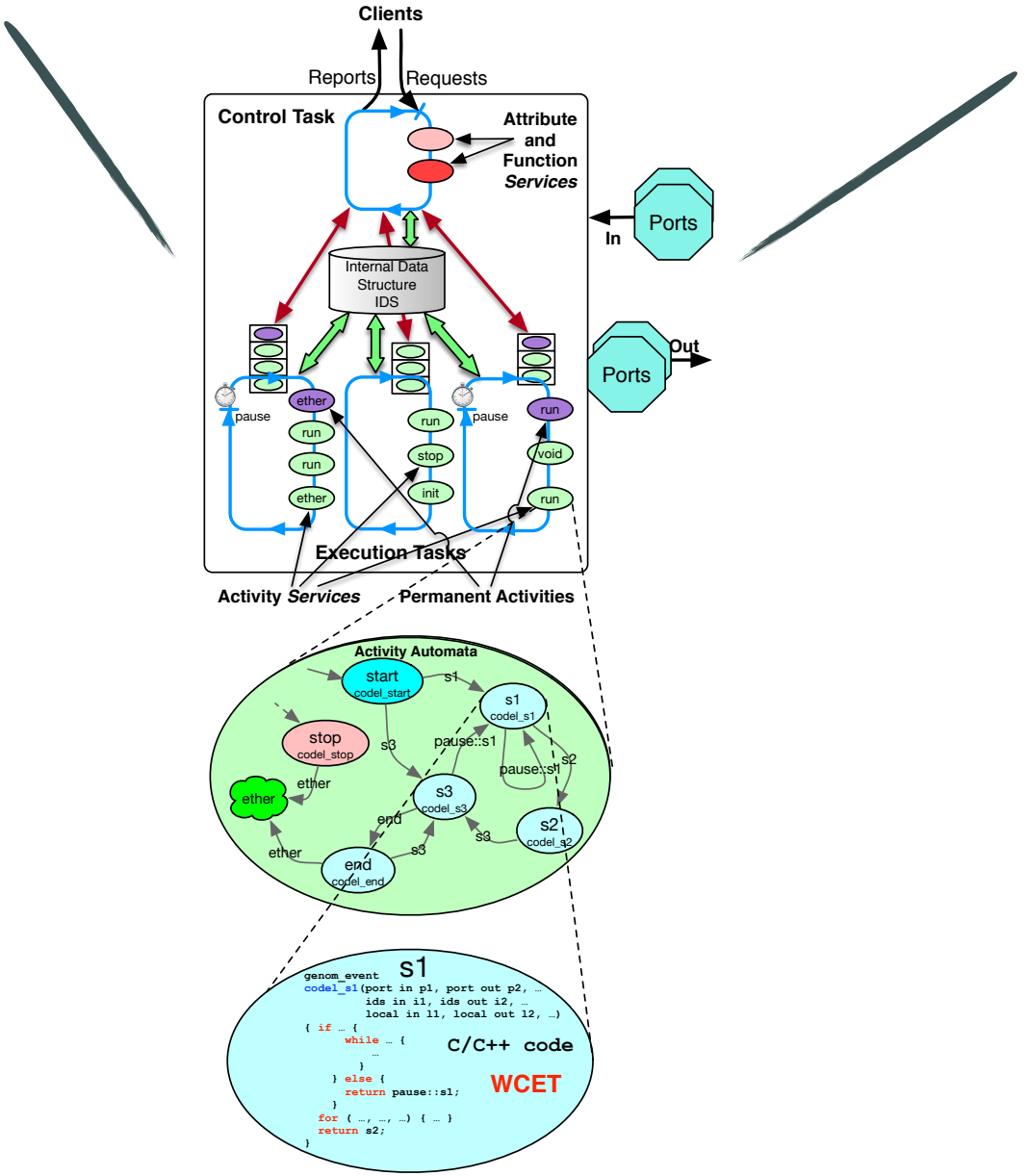
GenoM3 internal

- A program with I/O
  - control: requests to start services/reports their results
  - data: ports in (to import external data) and out (to export data)
- A cyclic event based control task (aperiodic)
- One or more cyclic execution tasks, periodic or aperiodic
- It provides services (short and long computation) to which we will associate C/C++ code
  - in the control task: attribute and function services (short)
  - and the executions task(s): activity services (long)
- services share a common Internal Data Structure for the needs of their computation (parameters, computed values, internal state variables, etc)
- activity services define automata to perform their processing
- each step is associated to a codel (C/C++ code)



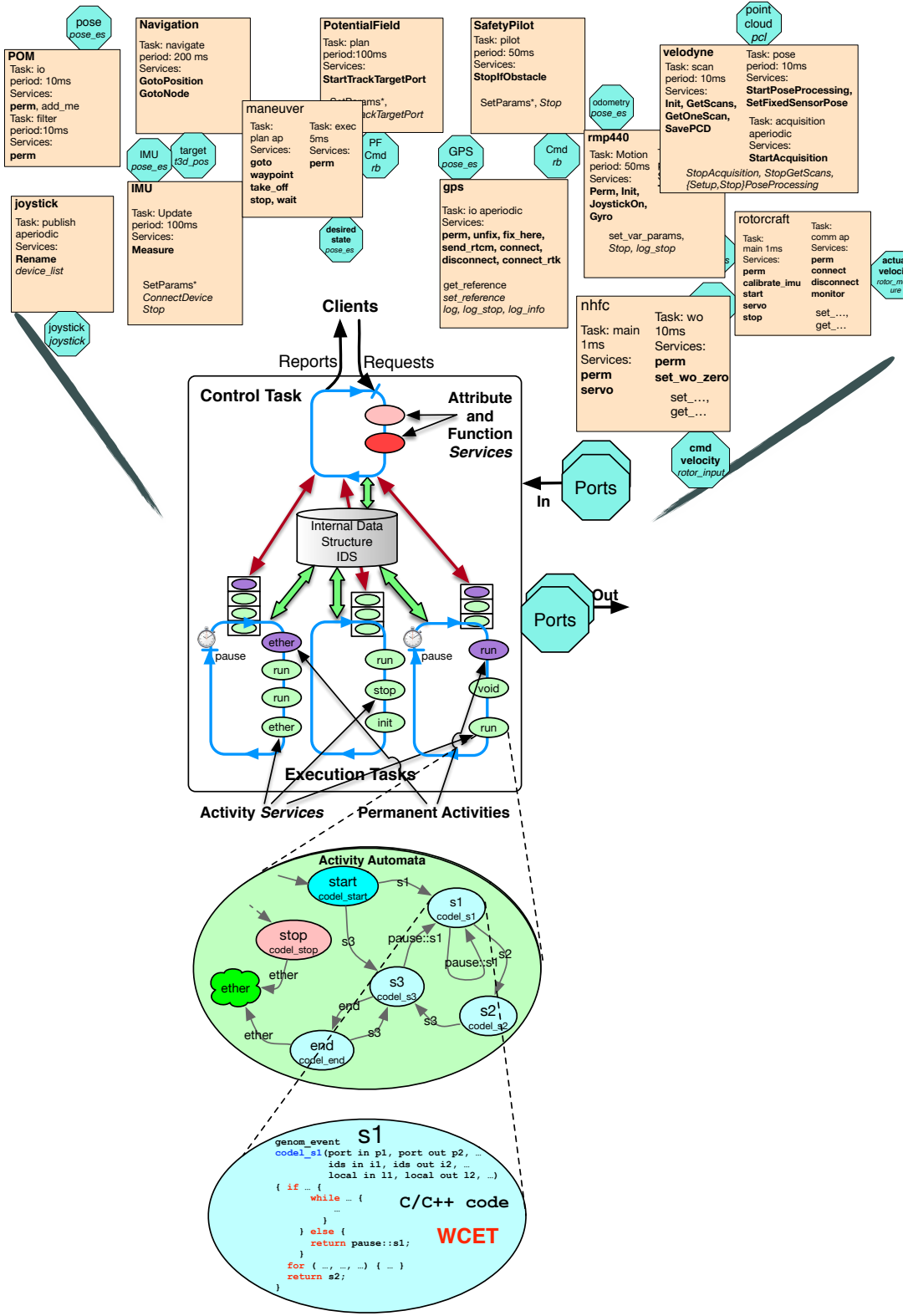
GenoM3 specifications

- IDS
- Ports (in & out)
- Execution Tasks
  - (periodic or aperiodic)
- Services
  - Attribute, function (control task)
  - Activity (execution task)
    - automata
    - and attached codels with WCET

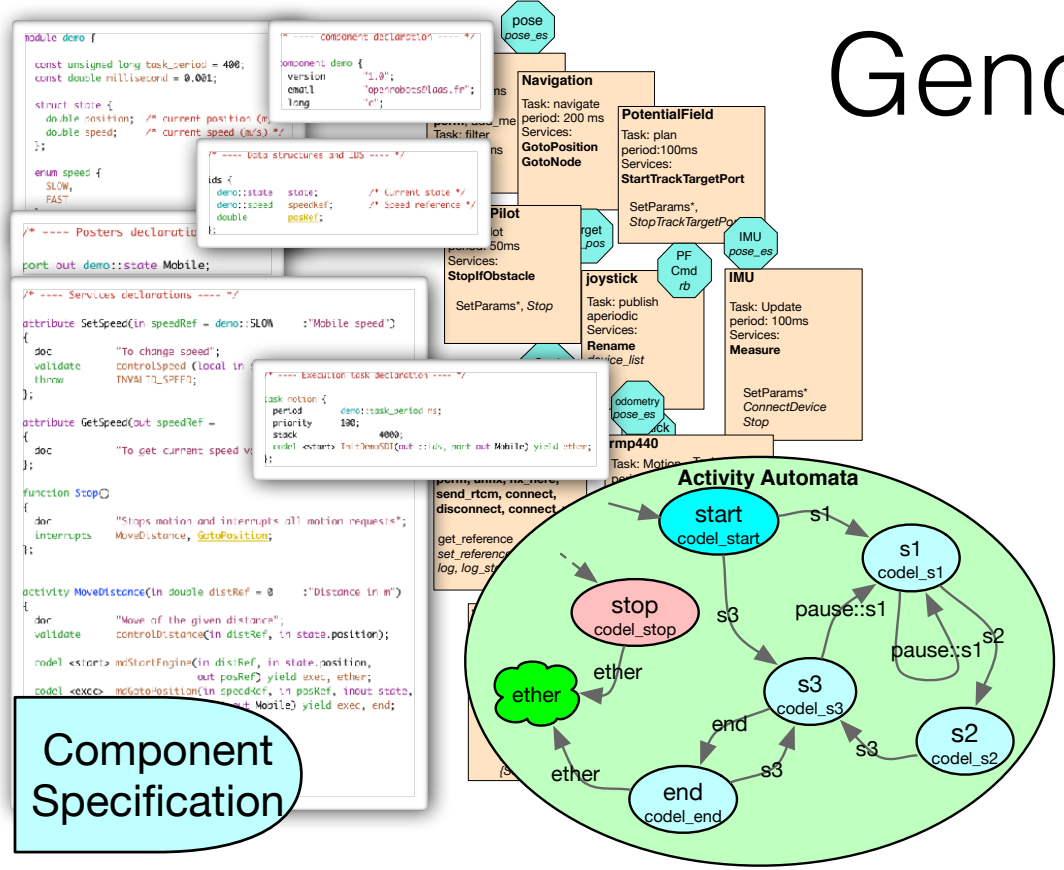


GenoM3 specifications

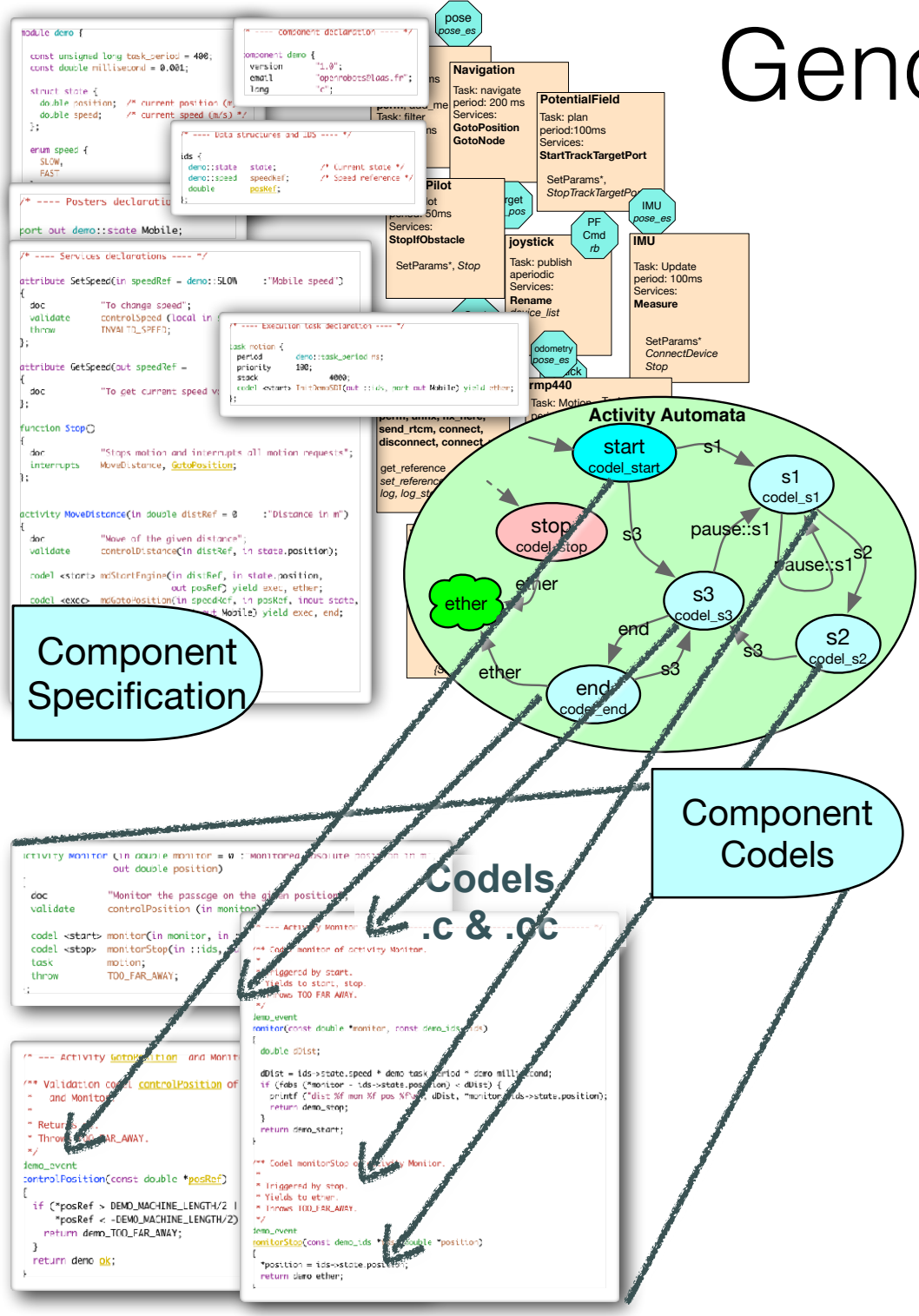
- IDS
- Ports (in & out)
- Execution Tasks
  - (periodic or aperiodic)
- Services
  - Attribute, function (control task)
  - Activity (execution task)
    - automata
    - and attached codels with WCET



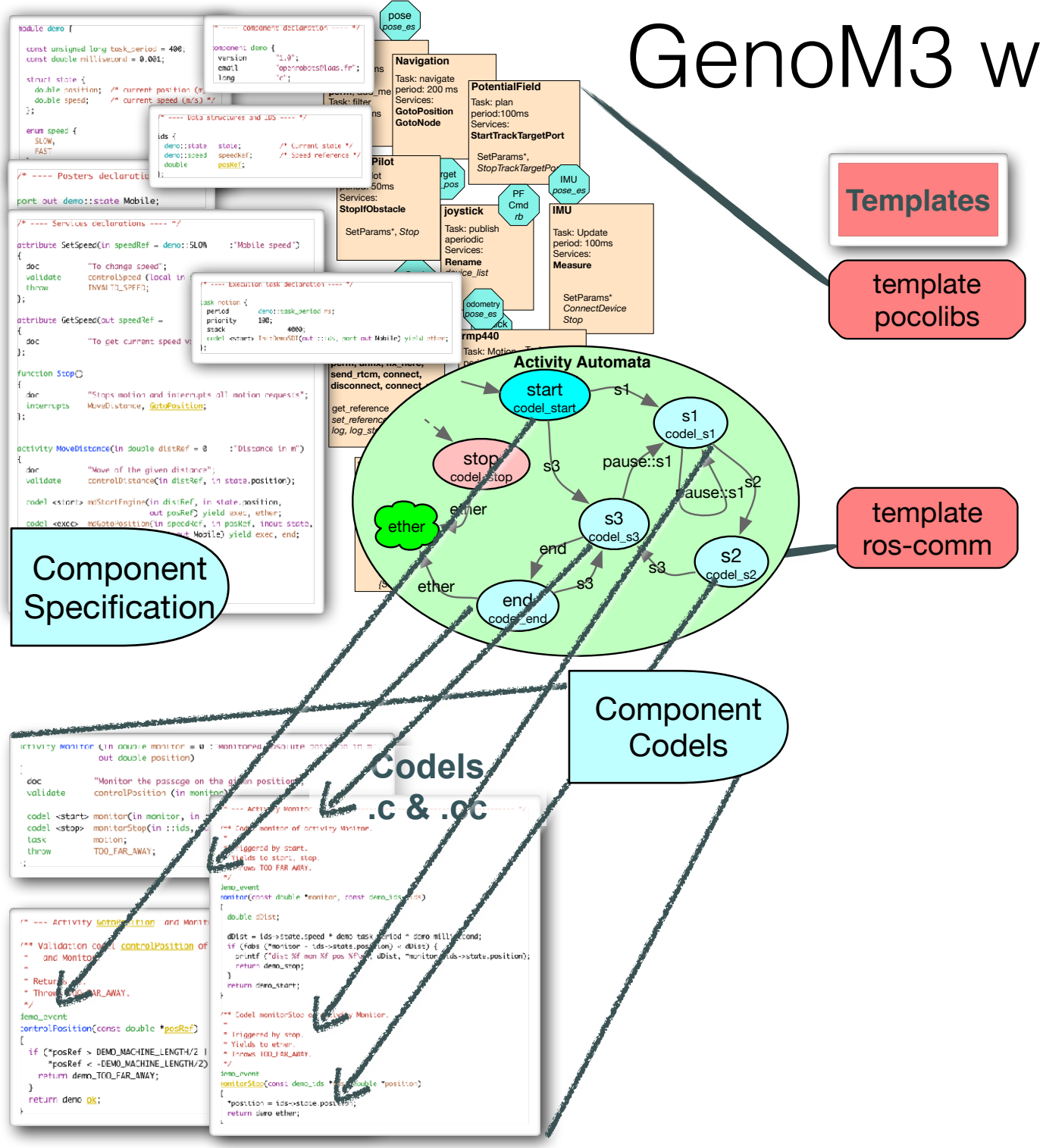
GenoM3 workflow



GenoM3 workflow



GenoM3 workflow



GenoM3 workflow

Component Specification

Activity Automata

Component Codels

Codels .c & .oc

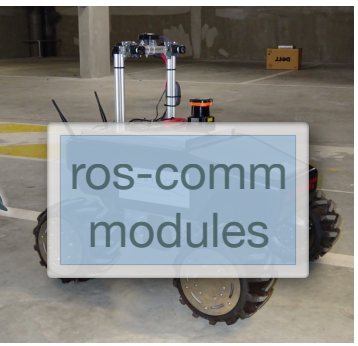
Templates

template pocolibs

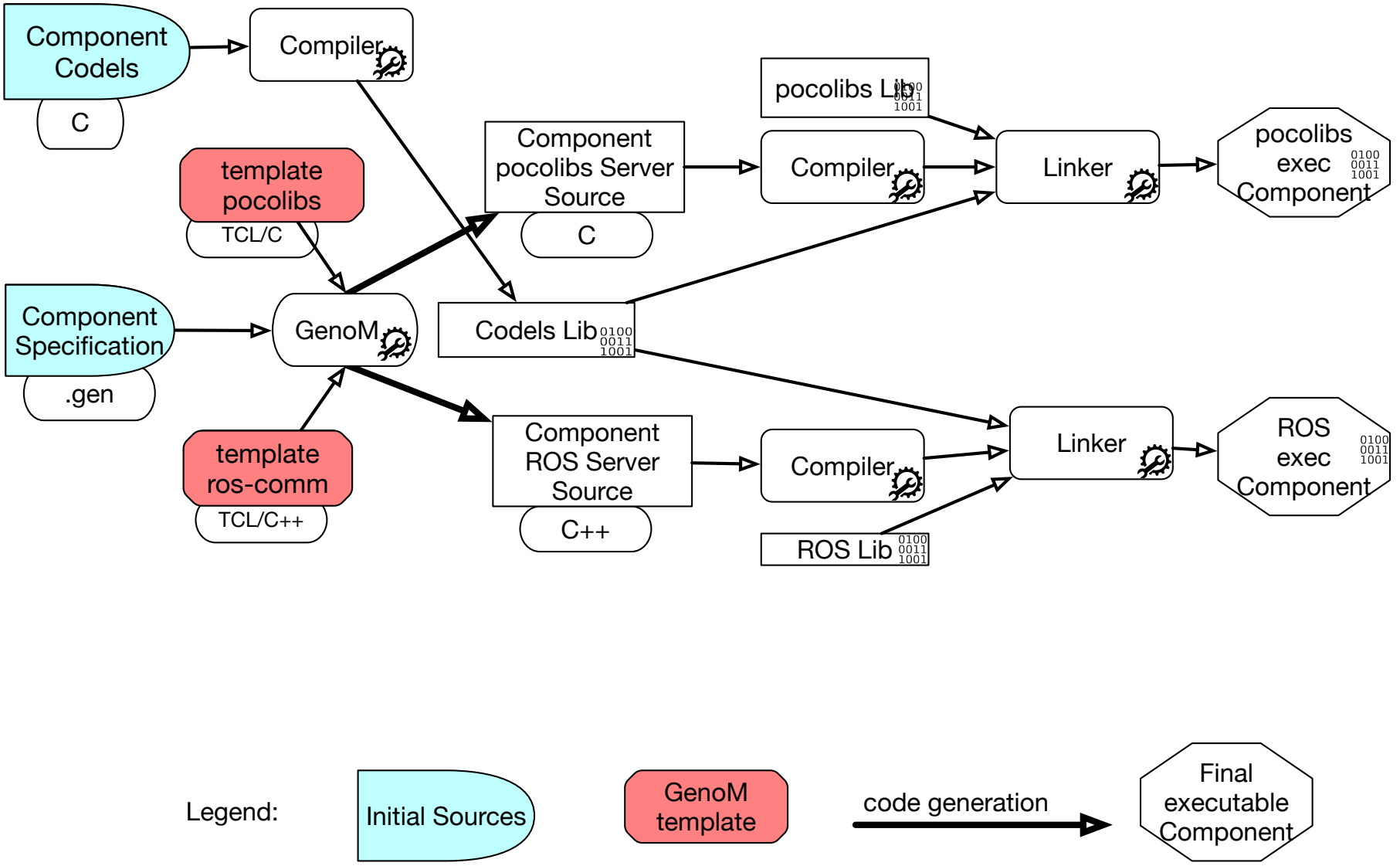
template ros-comm

pocolibs modules

ros-comm modules



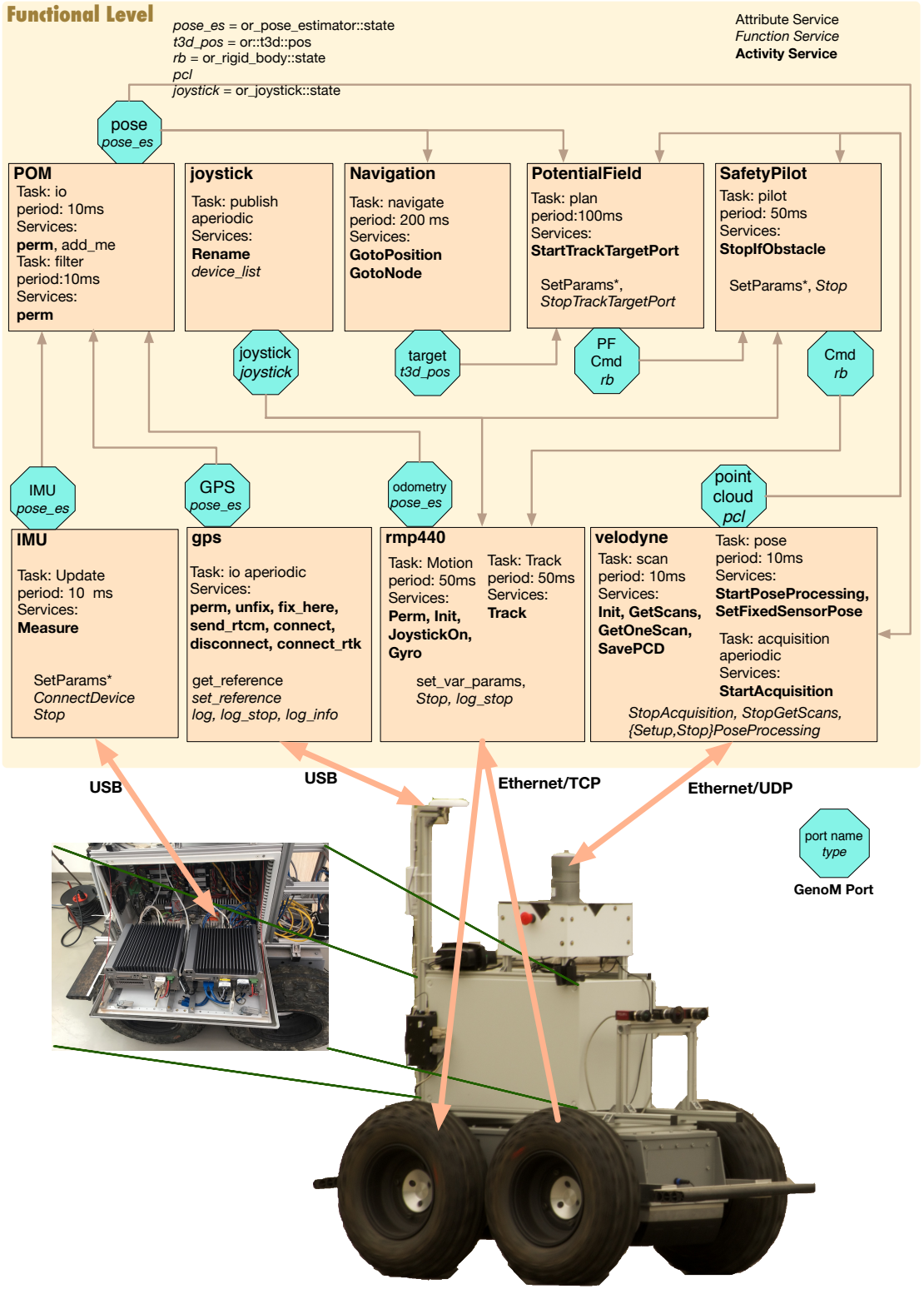
Template ROS and pocolibs

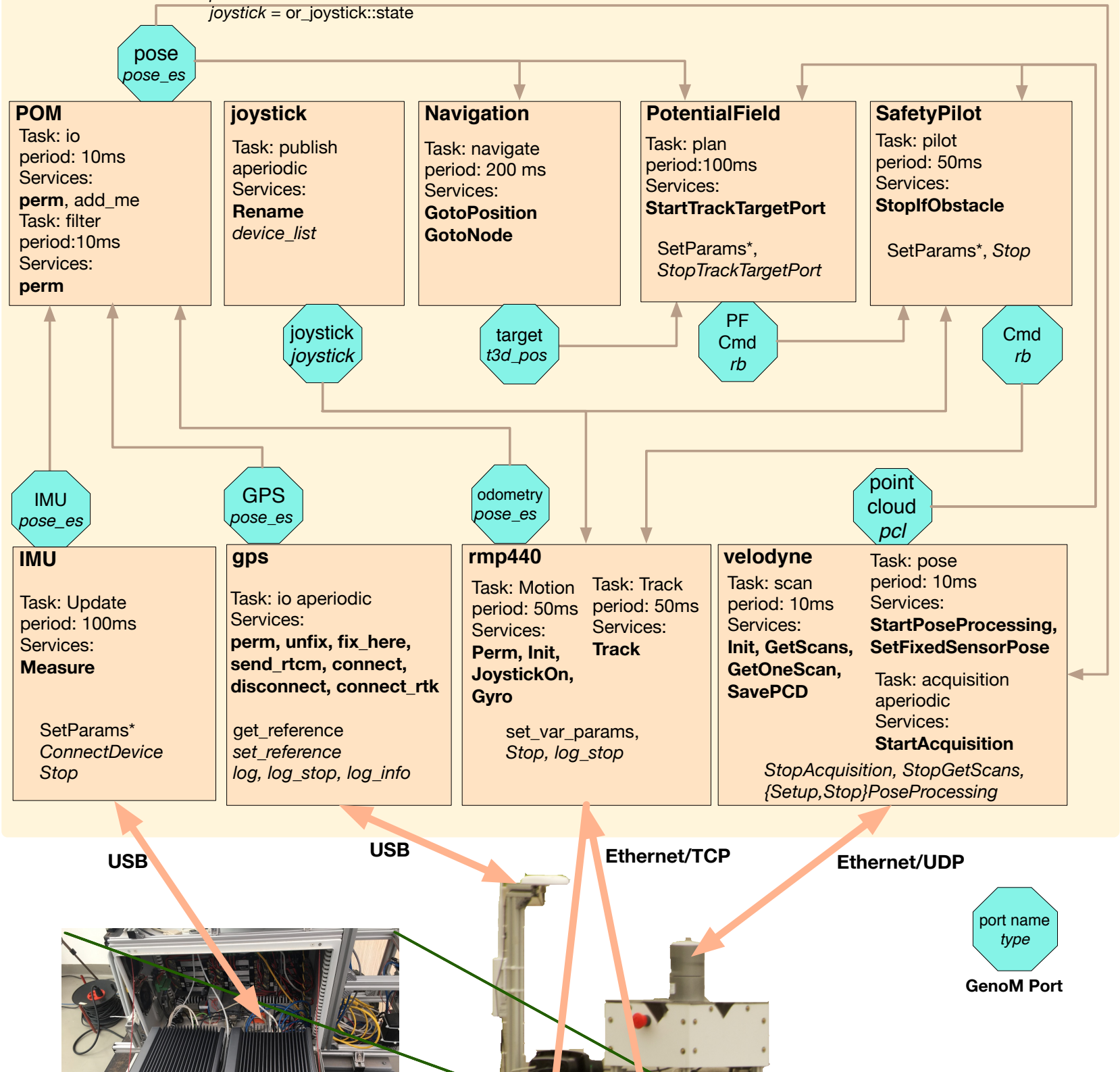


Minnie software architecture

- 9 components
- 9 ports
- (13 + 9) tasks (period)
- 38 activity services (with automata)
- 41 function services
- 43 attribute services
- 170 codels (14k loc) and their WCET
- 200k loc for all components + libraries

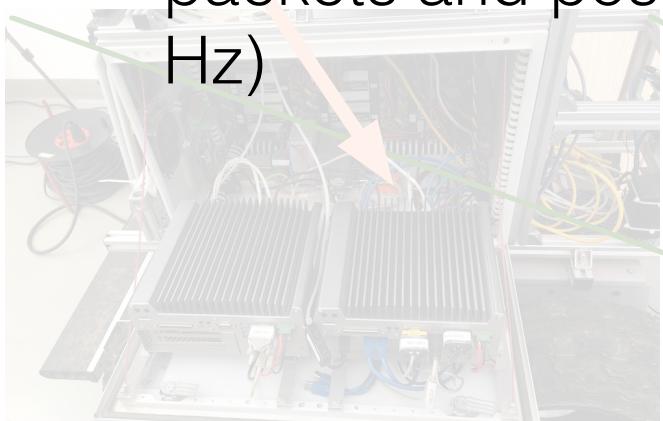
- Everything runs onboard…




joystick = or_joystick::state
pose
pose_es
POM
Task: io
period: 10ms
Services:
perm, add_me
Task: filter
period:10ms
Services:
perm
joystick
Task: publish
aperiodic
Services:
Rename
device_list
Navigation
Task: navigate
period: 200 ms
Services:
GotoPosition
GotoNode
PotentialField
Task: plan
period:100ms
Services:
StartTrackTargetPort
SetParams*,
StopTrackTargetPort
SafetyPilot
Task: pilot
period: 50ms
Services:
StopIfObstacle
SetParams*, Stop
joystick
joystick
target
t3d_pos
PF
Cmd
rb
Cmd
rb
IMU
pose_es
GPS
pose_es
odometry
pose_es
point
cloud
pcl
IMU
Task: Update
period: 100ms
Services:
Measure
SetParams*
ConnectDevice
Stop
gps
Task: io aperiodic
Services:
perm, unfix, fix_here,
send_rtcm, connect,
disconnect, connect_rtk
get_reference
set_reference
log, log_stop, log_info
rmp440
Task: Motion
period: 50ms
Services:
Perm, Init,
JoystickOn,
Gyro
Task: Track
period: 50ms
Services:
Track
set_var_params,
Stop, log_stop
velodyne
Task: scan
period: 10ms
Services:
Init, GetScans,
GetOneScan,
SavePCD
Task: pose
period: 10ms
Services:
StartPoseProcessing,
SetFixedSensorPose
Task: acquisition
aperiodic
Services:
StartAcquisition
StopAcquisition, StopGetScans,
{Setup,Stop}PoseProcessing
USB
USB
Ethernet/TCP
Ethernet/UDP
port name
type
GenoM Port

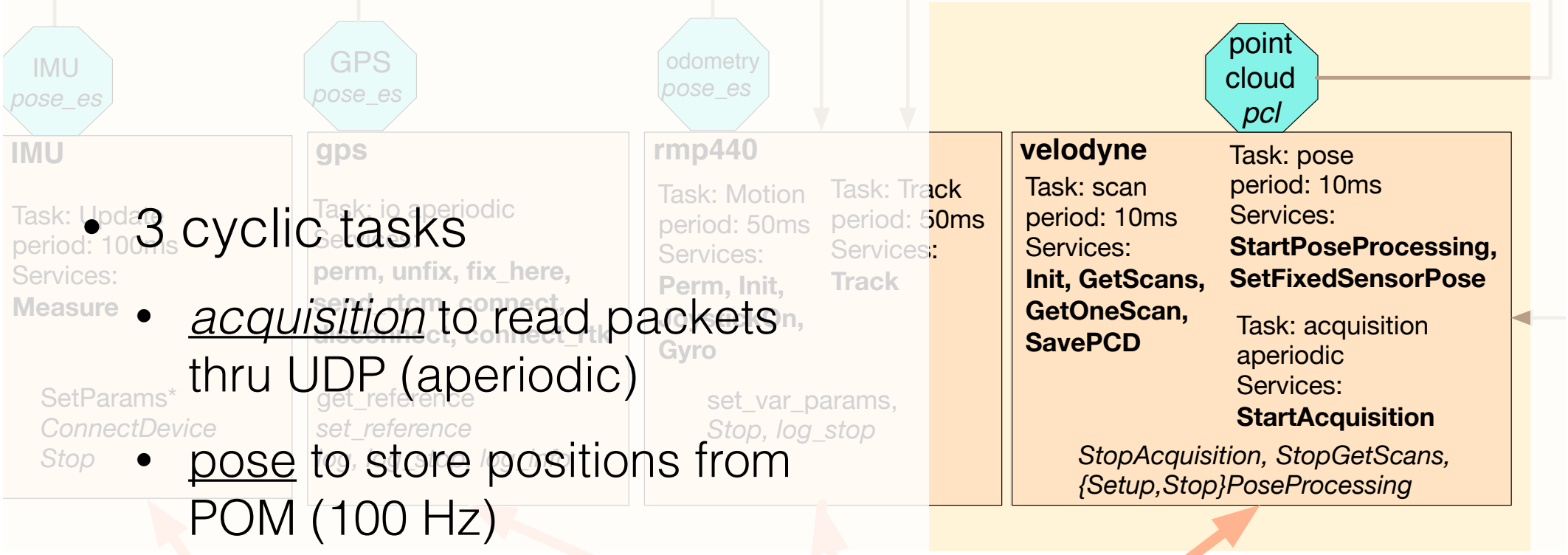
Minnie's Velodyne

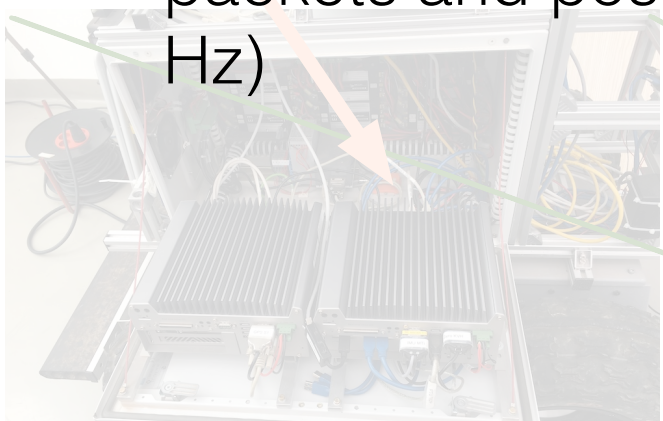
- 3 cyclic tasks
  - *acquisition* to read packets thru UDP (aperiodic)
  - pose to store positions from POM (100 Hz)
  - scan to build PCL scans from packets and positions (100 Hz)

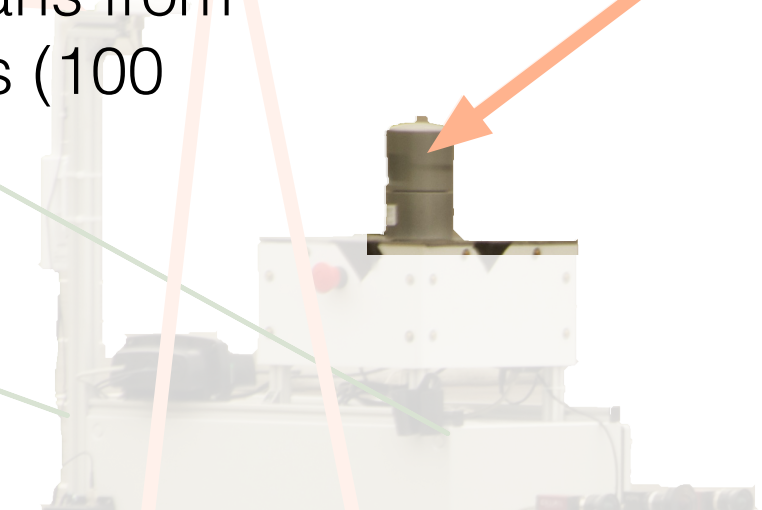


Minnie's Velodyne

- 3 cyclic tasks
  - *acquisition* to read packets thru UDP (aperiodic)
  - pose to store positions from POM (100 Hz)
  - scan to build PCL scans from packets and positions (100 Hz)

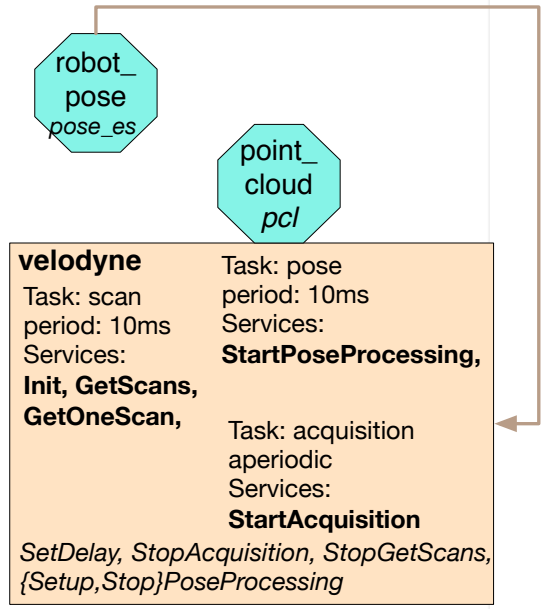




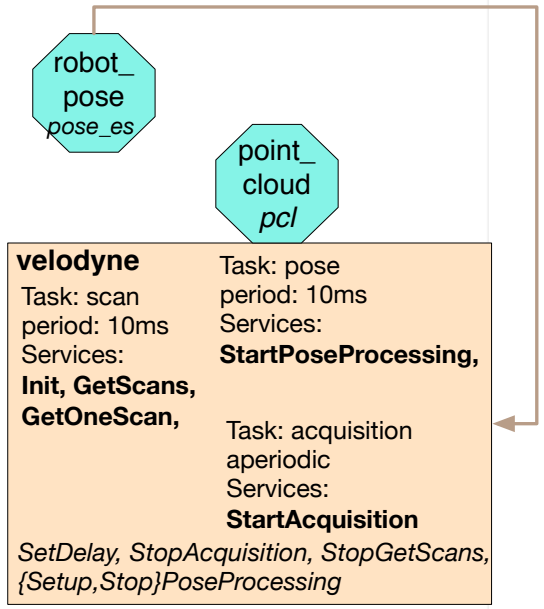


velodyne (simplified)

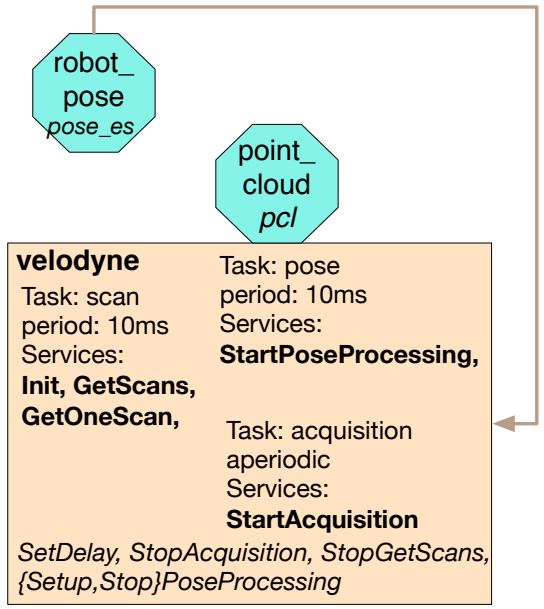
```
/*
 * Copyright (c) 2018-2019 LAAS/CNRS
 * All rights reserved.
 *
 * Redistribution  and  use  in  source  and binary  forms,  with  or  without
 * modification, are permitted provided that the following conditions are met:
 *
 *   1. Redistributions of  source  code must retain the  above copyright
 *      notice and this list of conditions.
 *   2. Redistributions in binary form must reproduce the above copyright
 *      notice and  this list of  conditions in the  documentation and/or
 *      other materials provided with the distribution.
 *
 * THE SOFTWARE  IS PROVIDED "AS IS"  AND THE AUTHOR  DISCLAIMS ALL WARRANTIES
 * WITH  REGARD   TO  THIS  SOFTWARE  INCLUDING  ALL   IMPLIED  WARRANTIES  OF
 * MERCHANTABILITY AND  FITNESS.  IN NO EVENT  SHALL THE AUTHOR  BE LIABLE FOR
 * ANY  SPECIAL, DIRECT,  INDIRECT, OR  CONSEQUENTIAL DAMAGES  OR  ANY DAMAGES
 * WHATSOEVER  RESULTING FROM  LOSS OF  USE, DATA  OR PROFITS,  WHETHER  IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR  OTHER TORTIOUS ACTION, ARISING OUT OF OR
 * IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#pragma require "openrobots2-idl >= 2.0"
#pragma require "minnie-idl"

#include "or/pose/pose_estimator.gen"
#include "mi/sensor/pcl.idl"

component velodyne {
  doc               "Provides corrected scans from Velodyne sensors.";
  version           "1.0";
  lang              "c";
  email             "felix@laas.fr";
  require           "genom3 >= 2.99.26";
  codels-require    "velodyne-libs >= 0.7", "eigen3", "pcl_common-1.7", "pcl_io-1.7";

  port in  or_pose_estimator::state robot_pose;
  port out or::pcl point_cloud;

  exception e_sys { string<256> what; };
//...
```



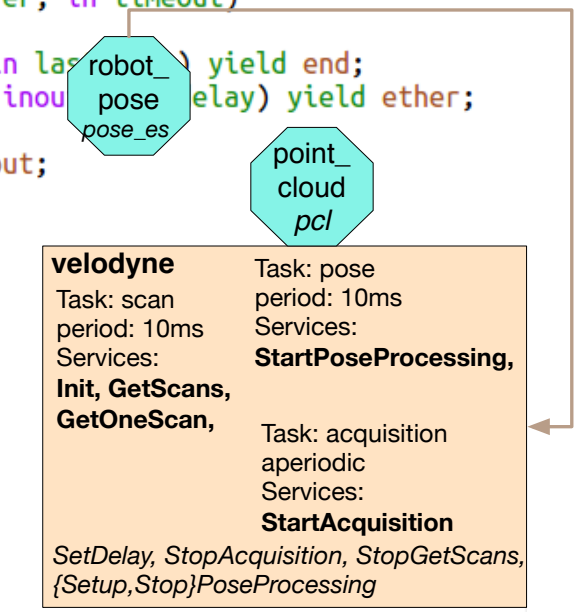
```
  exception e_sys { string<256> what; };
//...
  exception e_port { string<256> what; };

  /* --- ids ------------------------------------------------------------- */
  ids {
    AcquisitionParams acquisition_params; // Acquisition parameters
    PacketBuffer packet_buffer; // Buffer to store time stamped raw packets
    PoseBuffer pose_buffer;     // Buffer to store time stamped pose.
    long fd;                    // file descriptor to get the raw packets (UDP)

    long usec_delay;            // for fault injection purpose to delay port scan writting
  };

  attribute SetDelay(in usec_delay =  1000000)
  {
    doc         "Set the delay in usec we will delay port update (to test the BIP monitor).";
  };

  /* --- acquisition task ------------------------------------------------ */
  task acquisition {
    codel<start>        velodyneAcquisitionTaskStart() yield ether;
    codel<stop>         velodyneAcquisitionTaskStop()  yield ether;

    throws e_mem, e_grabber;
  };


  activity StartAcquisition() {
    doc                 "Starts the data acquisition";
    task                acquisition;

    codel<start>        velodyneStartAcquisitionStart(out fd, in acquisition_params) yield poll, ether;
    async codel<poll>   velodyneStartAcquisitionPoll(in fd) yield poll, main, ether;
    codel<main>         velodyneStartAcquisitionMain(out packet_buffer) yield poll, end, ether;
    codel<stop,end>     velodyneStartAcquisitionEnd() yield ether;

    after               Init;
    throws              e_sys, e_interface, e_runtime, e_not_implemented;
  };

  function StopAcquisition() {
    doc                 "Stop the data acquisition";
    codel               velodyneStopAcquisition();
    interrupts          StartAcquisition, GetOneScan, GetScans;
  };
```



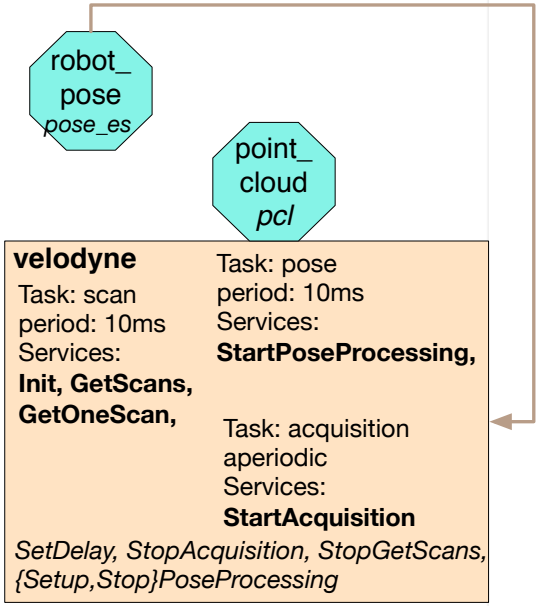
```
  after               Init;
  throws              e_sys, e_interface, e_runtime, e_not_implemented;
};

function StopAcquisition() {
  doc                 "Stop the data acquisition";
  codel               velodyneStopAcquisition();
  interrupts          StartAcquisition, GetOneScan, GetScans;
};


/* --- scan task ------------------------------------------------ */
task scan {
  period           10ms;

  codel<start>        velodyneScanTaskStart(out ::ids, port out point_cloud) yield ether;
  codel<stop>         velodyneScanTaskStop(out ::ids)  yield ether;

  throws e_mem, e_grabber;
};


/* --- scan services -------------------------------------------- */
activity Init ( in Model              model              =           :"Which model?")
{
  doc                 "Opens the interface .";
  task                scan;

  codel<start>        velodyneInit(out acquisition_params, in model) yield ether;

  throws              e_params, e_interface;
};

activity GetOneScan ( in double firstAngle = :"First angle of the scan (in degrees)",
                      in double lastAngle  = :"Last angle of the scan (in degrees)",
                      in double timeout    = :"Timeout used when stamping packets")
{
  doc                 "Acquire a full scan from the velodyne sensor";
  task                scan;

  validate             velodyneGetOneScanValidate(in firstAngle, in lastAngle);

  codel<start>         velodyneGetOneScanStart(in acquisition_params) yield copy_packets;
  codel<copy_packets>  velodyneGetOneScanCopyPackets(in acquisition_params, out packet_buffer) yield stamp_packets;
  codel<stamp_packets> velodyneGetOneScanStampPackets(in acquisition_params, out pose_buffer, in timeout)
                         yield pause::stamp_packets, build_scan;
  codel<build_scan>    velodyneGetOneScanBuildScan(in acquisition_params, in firstAngle, in lastAngle) yield end;
```



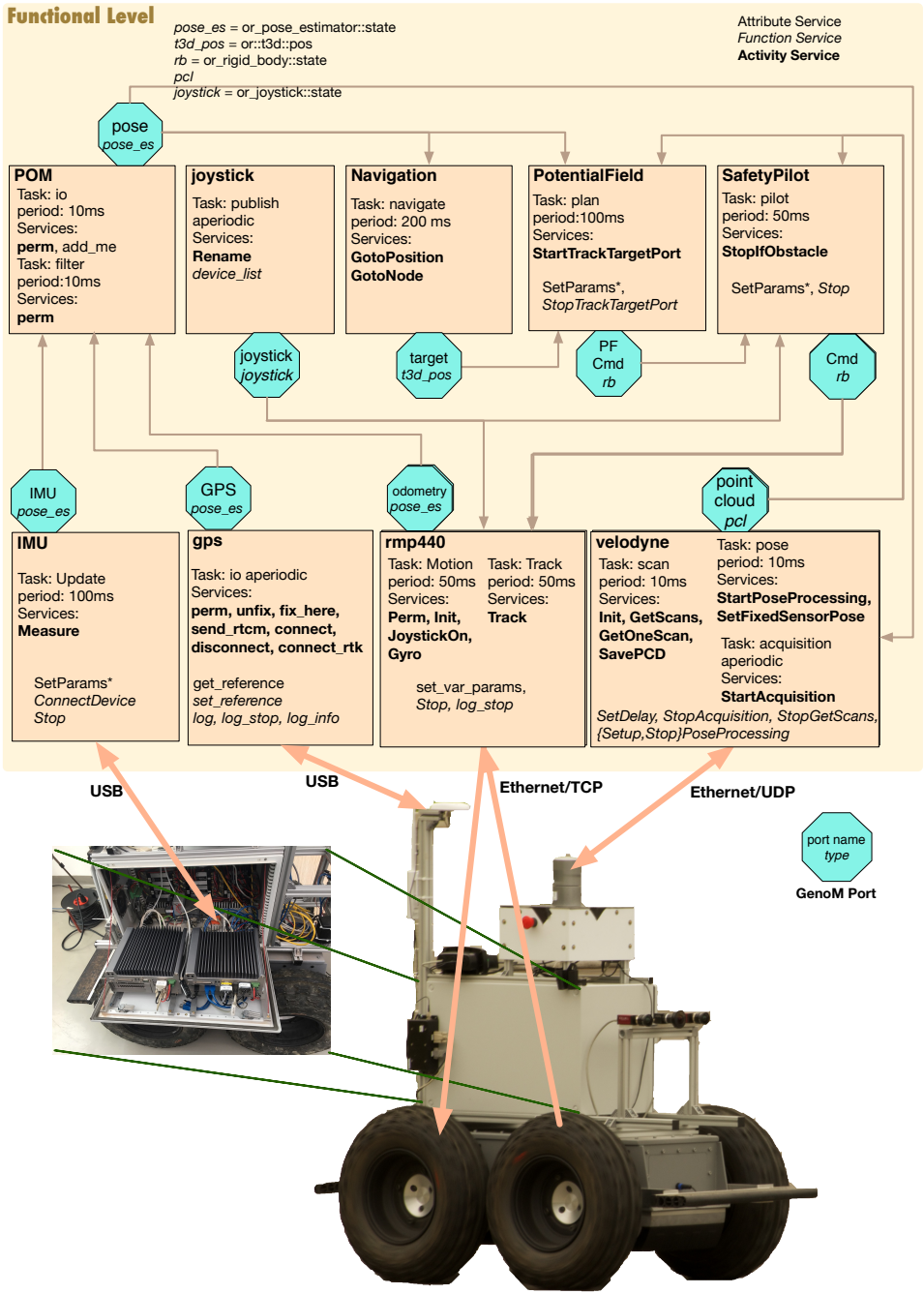
```
{
  doc                 "Acquire a full scan from the velodyne sensor";
  task                scan;

  validate             velodyneGetOneScanValidate(in firstAngle, in lastAngle);

  codel<start>         velodyneGetOneScanStart(in acquisition_params) yield copy_packets;
  codel<copy_packets>  velodyneGetOneScanCopyPackets(in acquisition_params, out packet_buffer) yield stamp_packets;
  codel<stamp_packets> velodyneGetOneScanStampPackets(in acquisition_params, out pose_buffer, in timeout)
                         yield pause::stamp_packets, build_scan;
  codel<build_scan>    velodyneGetOneScanBuildScan(in acquisition_params, in firstAngle, in las ) yield end;
  codel<end>           velodyneGetOneScanEnd(in acquisition_params, port out point_cloud, inou elay) yield ether;

  throws              e_params, e_runtime, e_interface, e_not_implemented, e_port, e_timeout;
  after               Init;
  interrupts          GetOneScan, GetScans;
};

activity GetScans ( in double firstAngle = :"First angle of the scan (in degrees)",
                    in double lastAngle  = :"Last angle of the scan (in degrees)",
                    in double period     = :"Time in between scan acquisitions",
                    in double timeout    = :"Timeout used when stamping packets")
{
  doc  "Acquire full scans from the velodyne sensor periodically";
  task scan;

  validate velodyneGetScansValidate(in firstAngle, in lastAngle, in period);

  codel<start>          velodyneGetScansStart(in acquisition_params) yield copy_packets;
  codel<copy_packets>   velodyneGetOneScanCopyPackets(in acquisition_params, out packet_buffer) yield stamp_packets;
  codel<stamp_packets>  velodyneGetOneScanStampPackets(in acquisition_params, out pose_buffer, in timeout)
                          yield pause::stamp_packets, build_scan;
  codel<build_scan>     velodyneGetOneScanBuildScan(in acquisition_params, in firstAngle, in lastAngle) yield end;
  codel<end>            velodyneGetOneScanEnd(in acquisition_params,  port out point_cloud, inout usec_delay) yield wait;
  codel<wait>           velodyneGetScansWait(in period) yield pause::wait, copy_packets;

  interrupts            GetOneScan, GetScans;
  after               Init;
  throws              e_params, e_runtime, e_interface, e_not_implemented, e_port, e_timeout;
};

function StopGetScans() {
  doc  "Stop the periodic acquisition of full scans";
  interrupts GetOneScan, GetScans;
};

/* --- pose task ------------------------------------------------------- */
task pose {
```



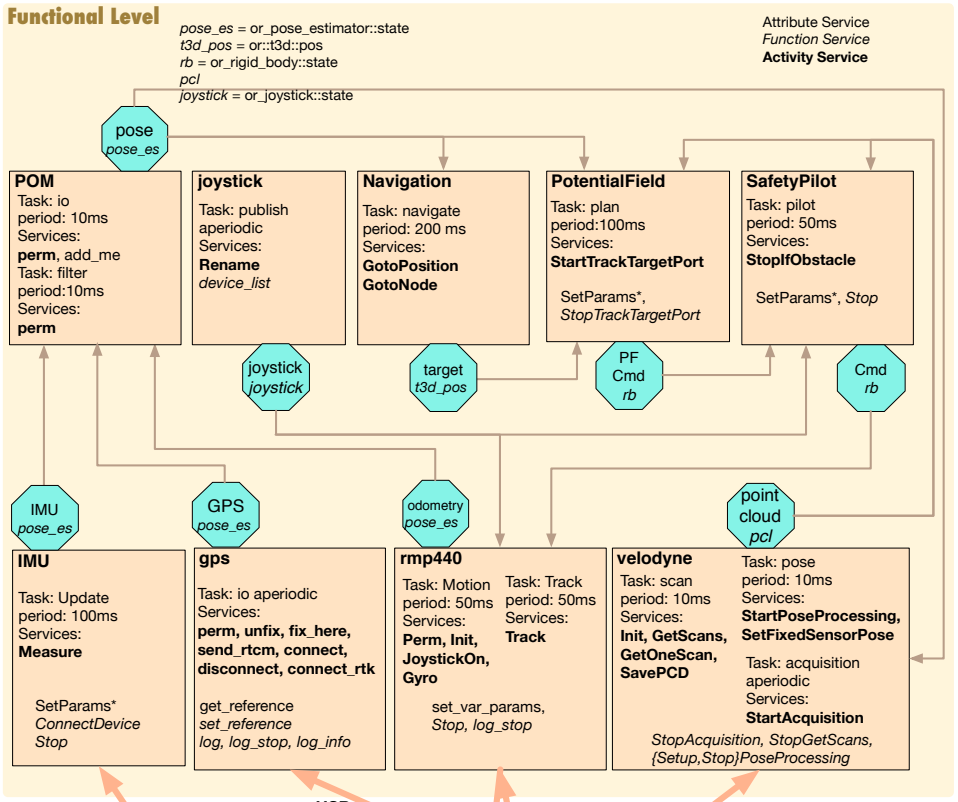
```
  throws              e_params, e_runtime, e_interface, e_not_implemented, e_port, e_timeout;
};

function StopGetScans() {
  doc  "Stop the periodic acquisition of full scans";
  interrupts GetOneScan, GetScans;
};

/* --- pose task ------------------------------------------------------- */
task pose {
  period           10ms;
  codel<start> velodynePoseTaskStart(out pose_buffer) yield ether;
};

/* --- pose services --------------------------------------------------- */
function SetupPoseProcessing(in unsigned long pose_buffer_size) {
  doc  "Setup the processing of poses";
  codel velodyneSetupPoseProcessing(in pose_buffer_size);
  interrupts StartPoseProcessing;
};

activity StartPoseProcessing() {
  doc         "Starts the processing of poses coming from input ports";
  task        pose;

  codel<start> velodyneStartPoseProcessingStart(in pose_buffer) yield main, ether;
  codel<main>  velodyneStartPoseProcessingMain(out pose_buffer, port in robot_pose) yield pause::main;

  throws      e_port, e_runtime;
  after       SetupPoseProcessing;
  interrupts  StartPoseProcessing;
};

function StopPoseProcessing() {
  doc         "Stop the processing of poses coming from input ports";
  interrupts StartPoseProcessing;
};
};
```

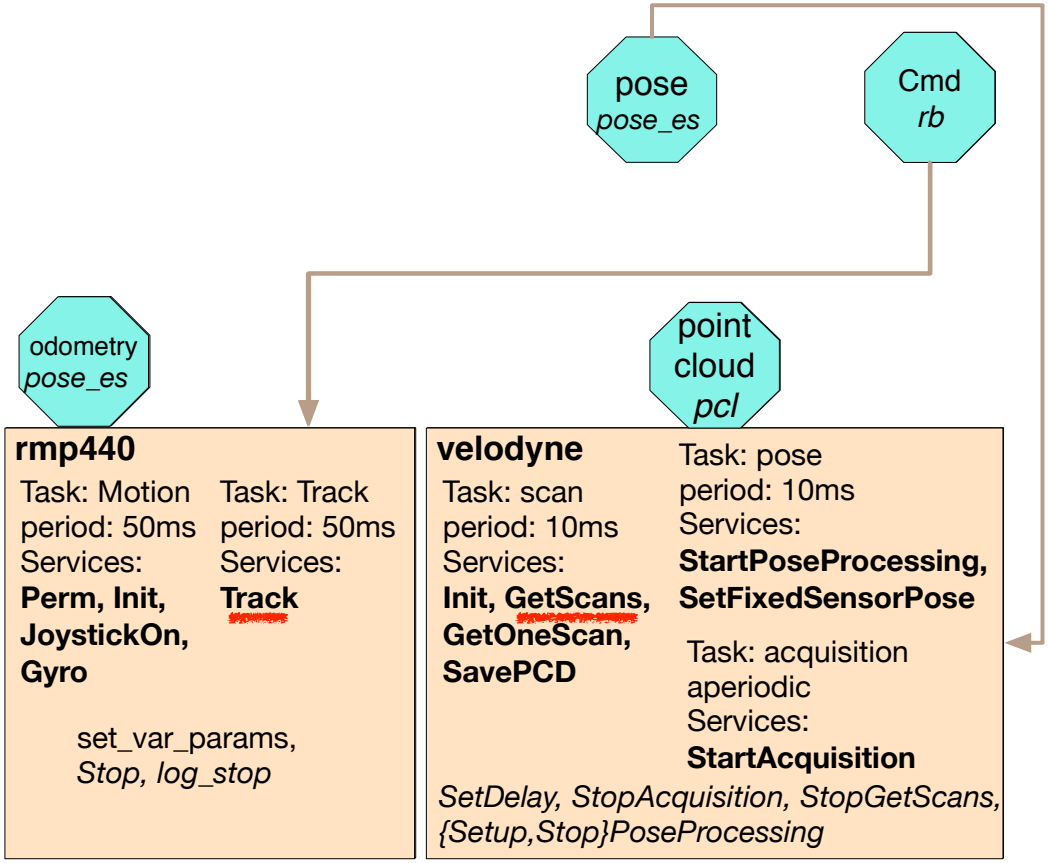


Services Examples



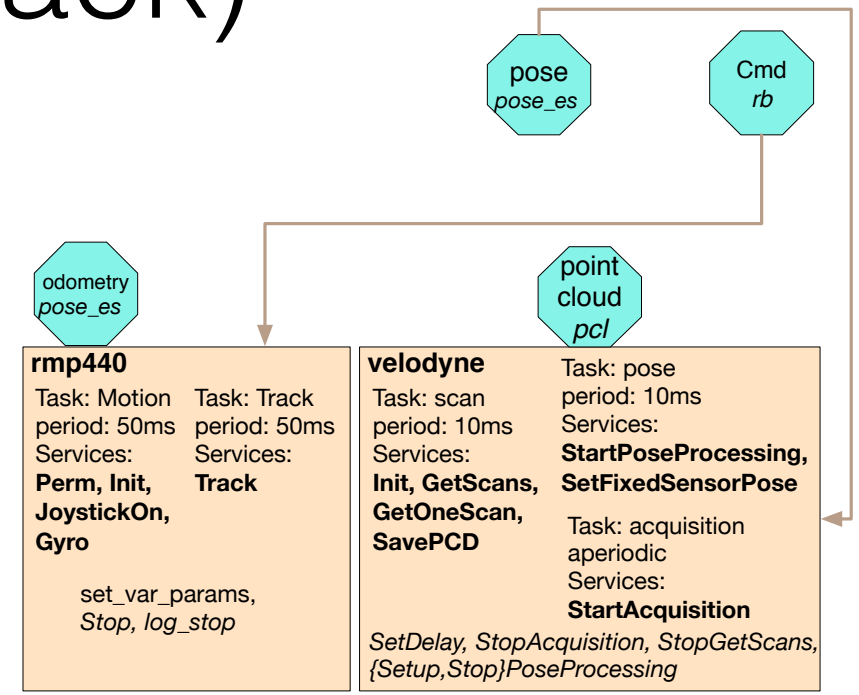
Services Examples

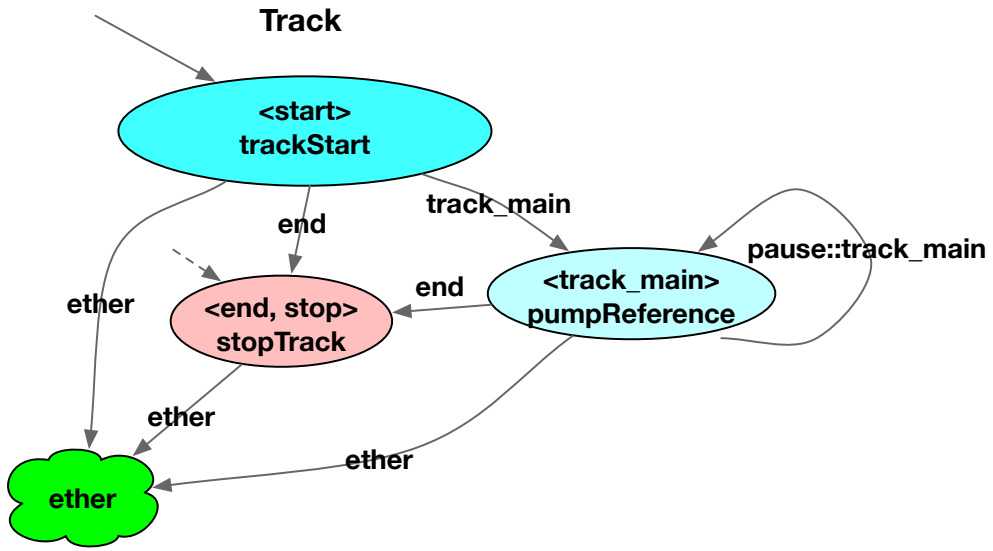




GenoM: Service Internal Automata (rmp440 Track)

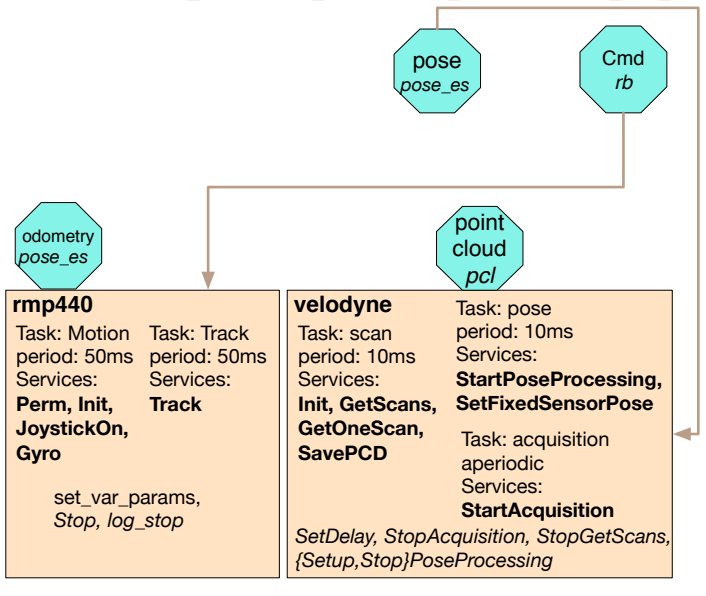
```
activity Track() {
  doc  "Start tracking a reference port";
  validate trackControl(in rmp);
  codel <start>trackStart(inout rs_mode,
                          out max_accel,
                          port in cmd_vel) yield track_main, end;
  codel <track_main>pumpReference(in rs_mode,
                            port in cmd_vel,
                            out ref) yield pause::track_main, end;
  codel <end,stop>stopTrack(inout rs_mode,
                            out ref) yield ether;
  task TrackTask;
  throw not_connected, port_not_found, bad_ref, cmd_stop_track,
    motors_off, emergency_stop, power_cord_connected;
  interrupts JoystickOn, Track;
};
```

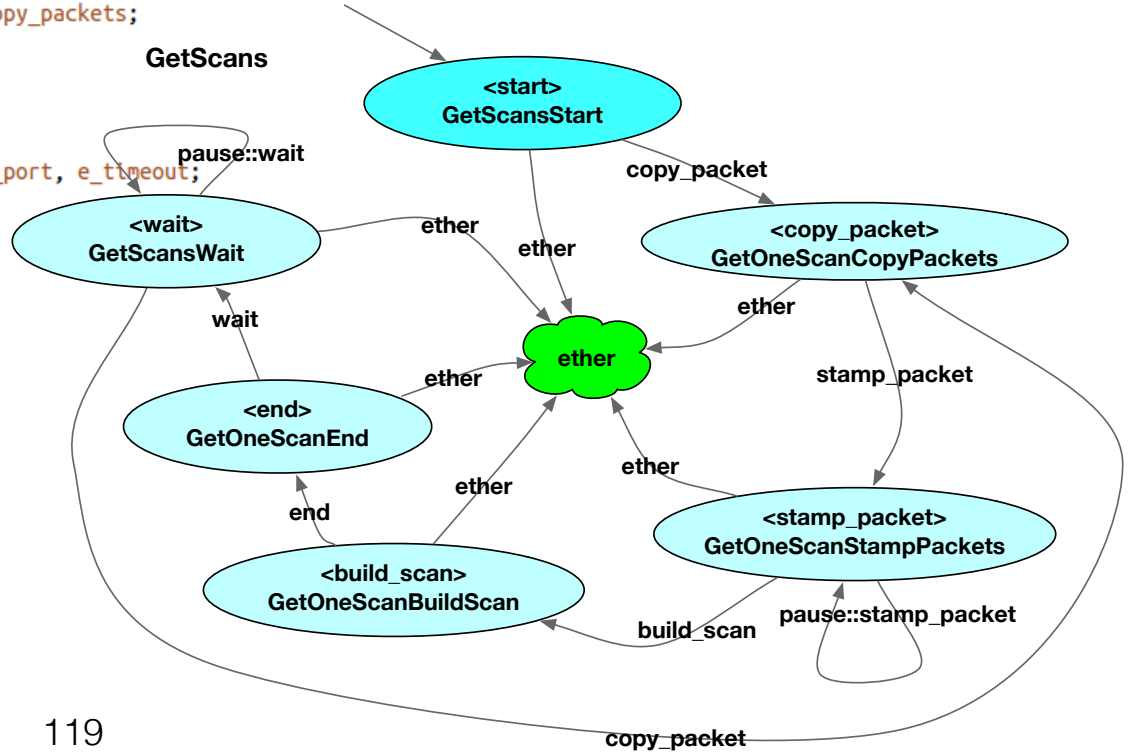




GenoM: Service Internal Automata (velodyne GetScans)

```
activity GetScans ( in double firstAngle = :"First angle of the scan (in degrees)",
                    in double lastAngle  = :"Last angle of the scan (in degrees)",
                    in double period     = :"Time in between scan acquisitions",
                    in double timeout    = :"Timeout used when stamping packets")
{
  doc  "Acquire full scans from the velodyne sensor periodically";
  task scan;

  validate velodyneGetScansValidate(in firstAngle, in lastAngle, in period);

  codel<start>          velodyneGetScansStart(in acquisition_params) yield copy_packets;
  codel<copy_packets>   velodyneGetOneScanCopyPackets(in acquisition_params, out mutex_buffer) yield stamp_packets;
  codel<stamp_packets>  velodyneGetOneScanStampPackets(in acquisition_params,
                                                       out mutex_pose_data, in timeout) yield pause::stamp_packets, build_scan;
  codel<build_scan>     velodyneGetOneScanBuildScan(in acquisition_params,
                                                    in firstAngle, in lastAngle) yield end;
  codel<end>            velodyneGetOneScanEnd(in acquisition_params,  in auto_save_pcd, out auto_save_pcd_count,
                                              in auto_save_pcd_prefix, port out point_cloud, port out point_cloud2, inout usec_delay)
                                              yield wait;
  codel<wait>           velodyneGetScansWait(in period) yield pause::wait, copy_packets;

  interrupts            GetOneScan, SavePCD, GetScans;

  after               Init;

  throws              e_params, e_runtime, e_interface, e_not_implemented, e_port, e_timeout;

};
```





GenoM3 : “Formal” Internal Component Model?

GenoM3 framework is middleware independent and enforces a clear internal organization of components

- Pros
  - Codel granularity (better parallelism specification)
  - Data access (and sharing/locking) is fully specified (ports, IDS, and nothing else)
  - Tasks are clearly specified (how many, periodic, sporadic)
  - Automata specification provides execution sequence and time/period management
  - Robustness (non user code is shared and used more often)
  - Decoupling architecture / algorithmic core
  - Do not rely on “magic” mechanism (e.g. ros spin), often badly understood by developers
  - Allow the use of Validation and Verification tools (later on this)
  - Template mechanism…
- Cons
  - Developers have to follow strict guidelines