Vérification formelle d'un robot mobile

Félix Ingrand
LAAS/CNRS
felix@laas.fr

et aussi
Anthony Mallet, Silvano Dal Zilio, Pierre Emmanuel Hladik

Le bilan carbone des déplacements nécessaires pour cette présentation est de:
eCO$_2$: 0,000 kg

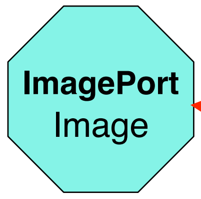
Why formal V&V for Robotic Systems?

- We are dealing with critical systems whose failure can be catastrophic!
- Formal V&V is one approach, among many available, to increase the trust we have in robotic systems
- Already use in many critical domains (aeronautic, nuclear plant, etc)
- It does not solve “all the problems”, yet it is a step in the right direction
- It can be integrated in existing frameworks

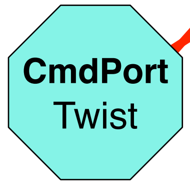
An illustrating example

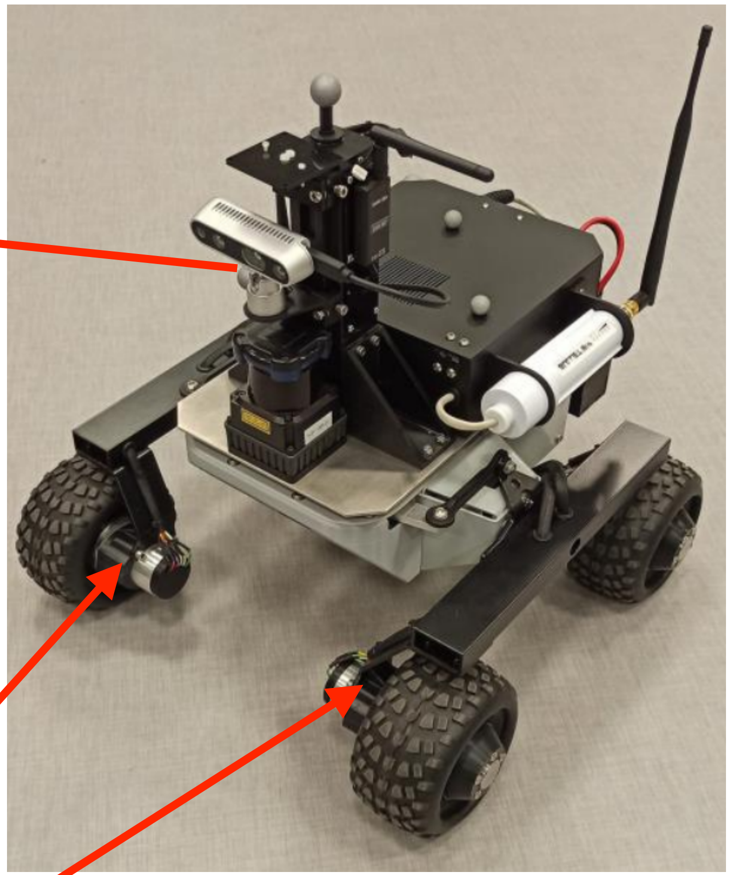
GenoM3, simple example, ColorTrack Robot: CT_robot

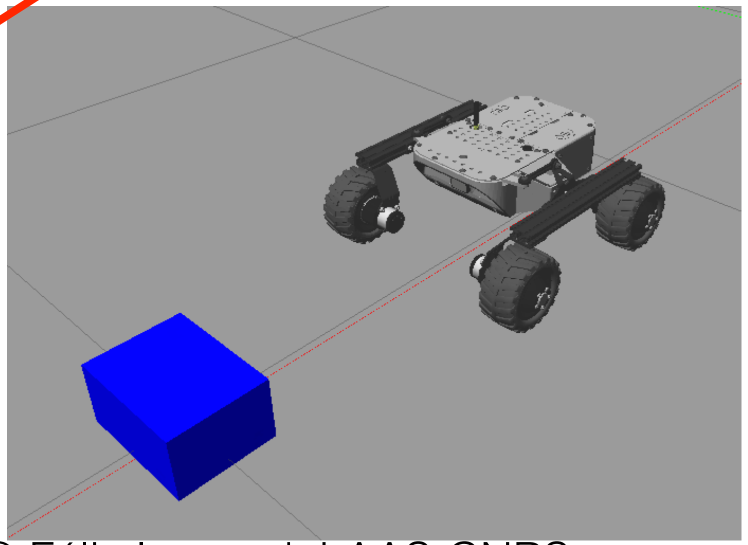
- CT_robot component (node):
  - has access to an **image** in **ImagePort**



  - 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**

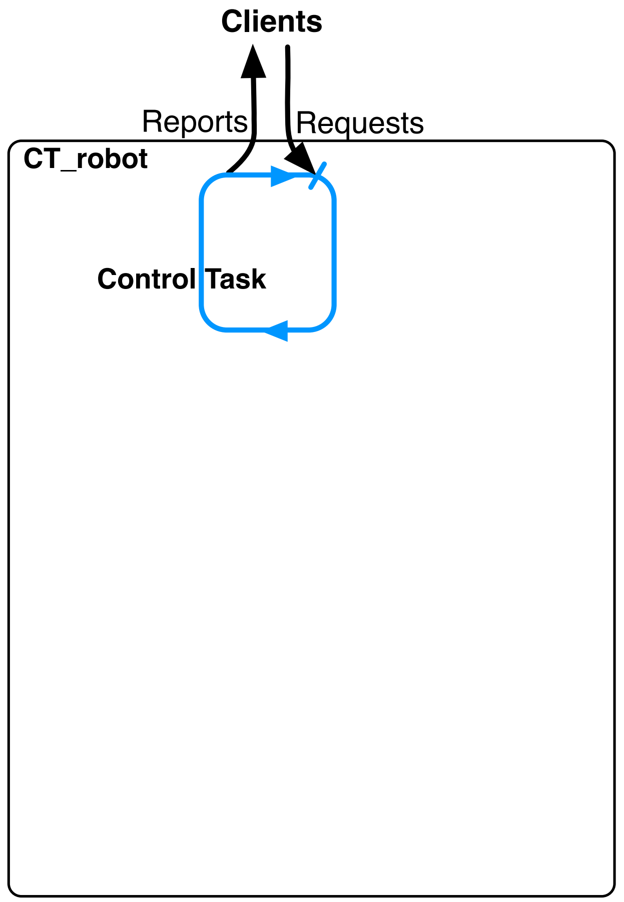




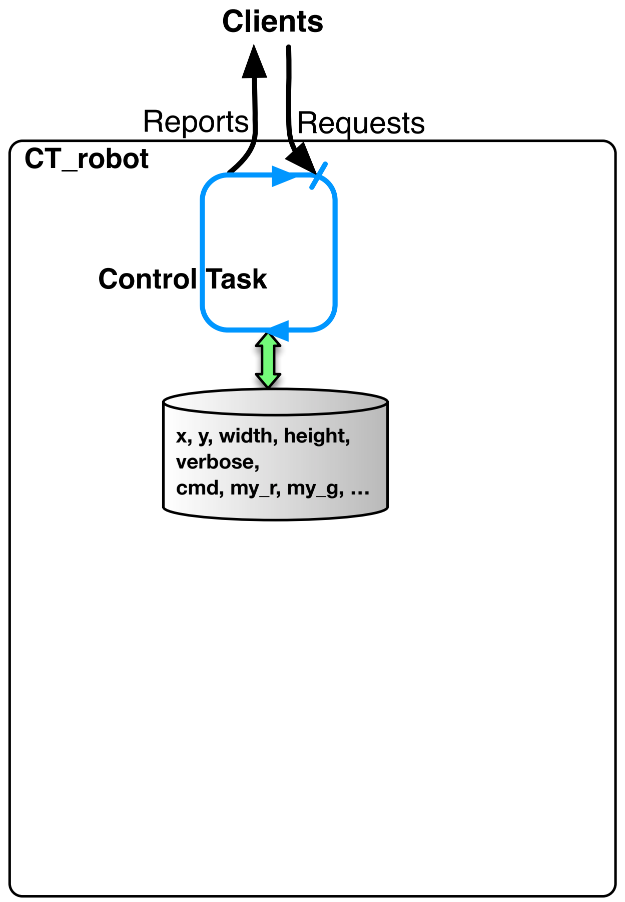


CT_robot .gen

```
/*
 * 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;
```

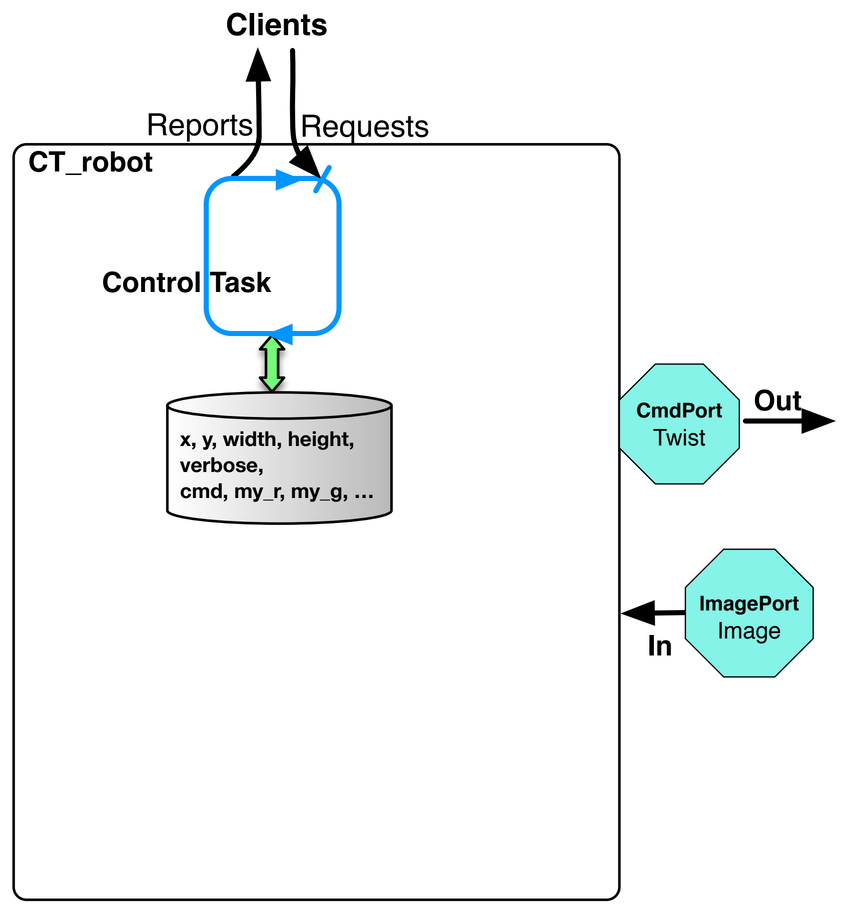


CT_robot .gen



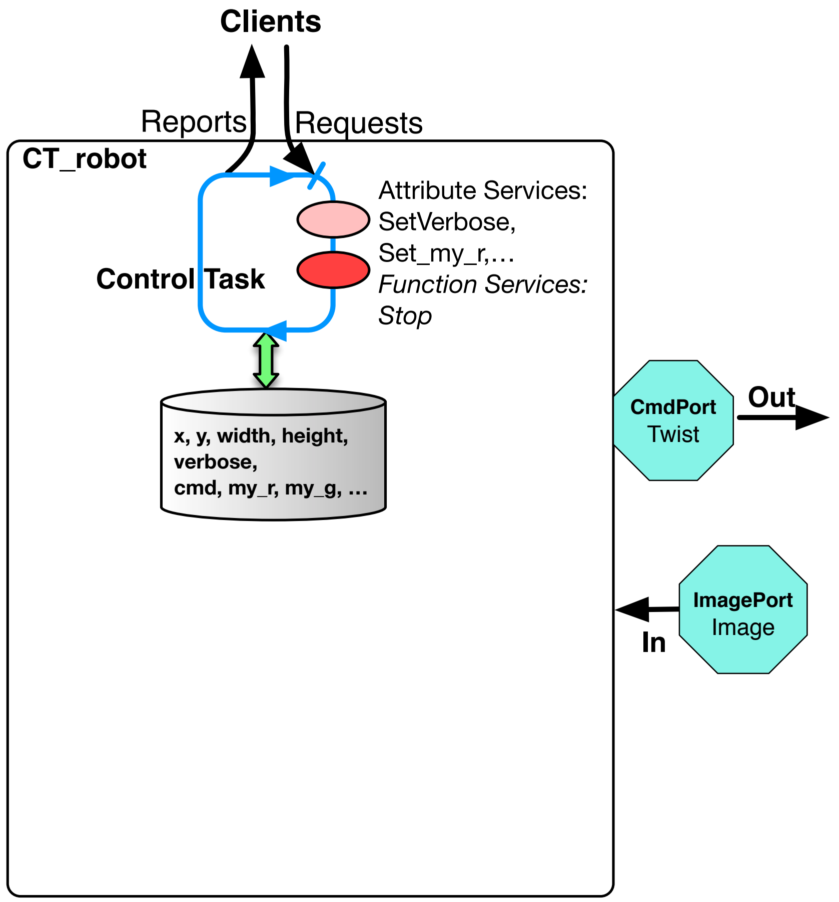
```
struct cmd_s{
    double vx;       // The internal speed struct declaration
    double wz;
};
ids {
  long x,y;              // Position of the center of orange object in the image
  long width,height; // Size of the image
  long verbose;        // For logging verbosity
  cmd_s cmd;           //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



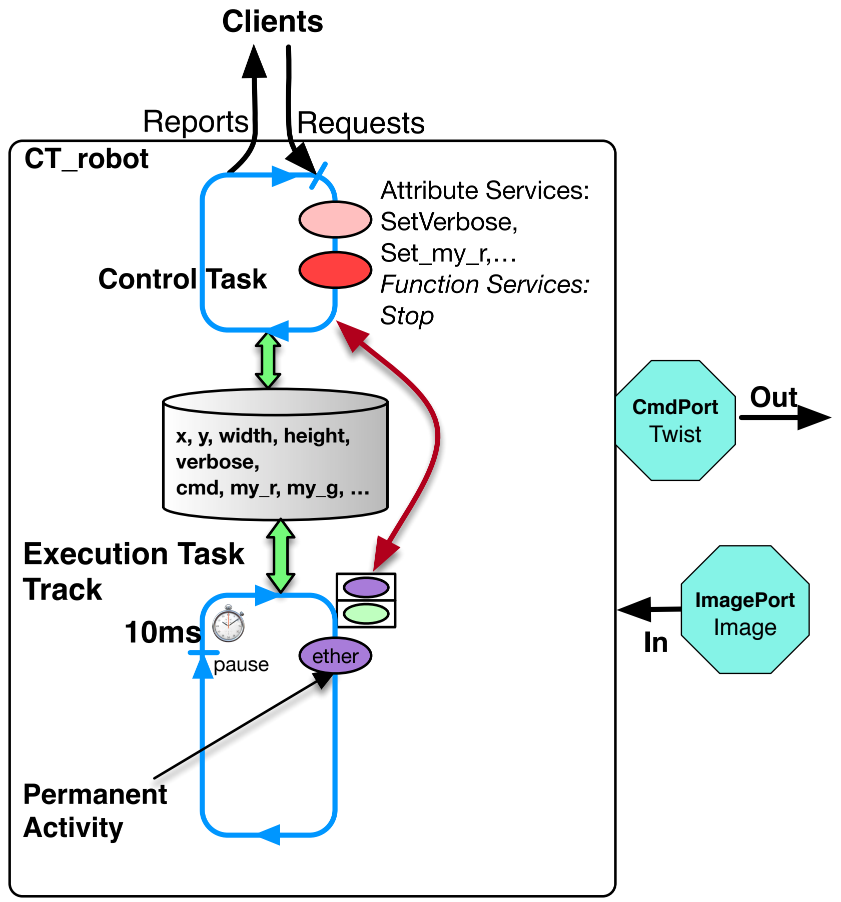
```
/* ------------- DEFINITION OF PORTS -------------- */
port in sensor::Image ImagePort {
  doc "The port ImagePort containing the image from the camera.";
};

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



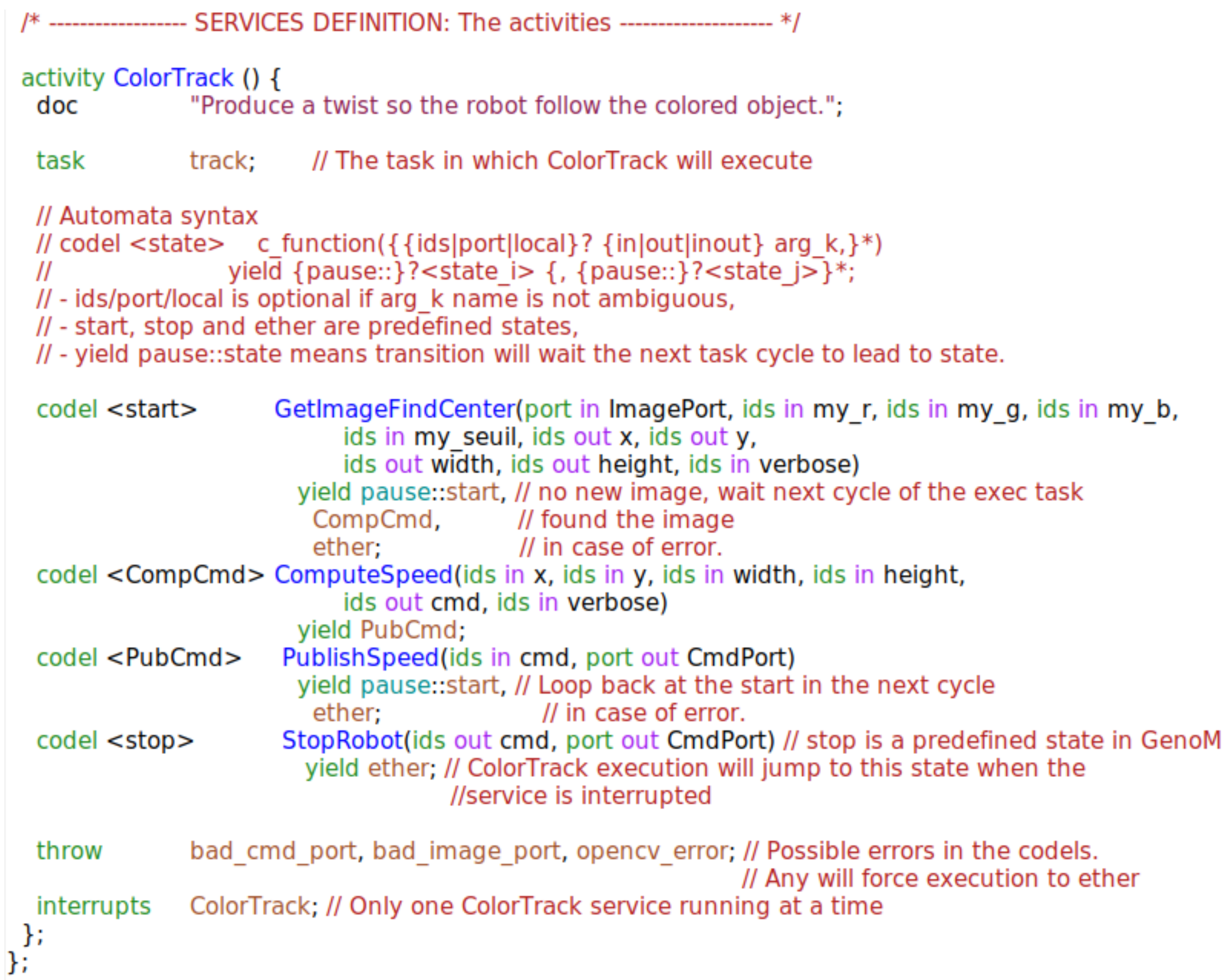
```
/* ----------------- SERVICES DEFINITION: The attributes ------------------- */
attribute SetVerbose(in verbose = 0 : "Verbose level")
{
 doc            "Set the verbose level.";
};

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

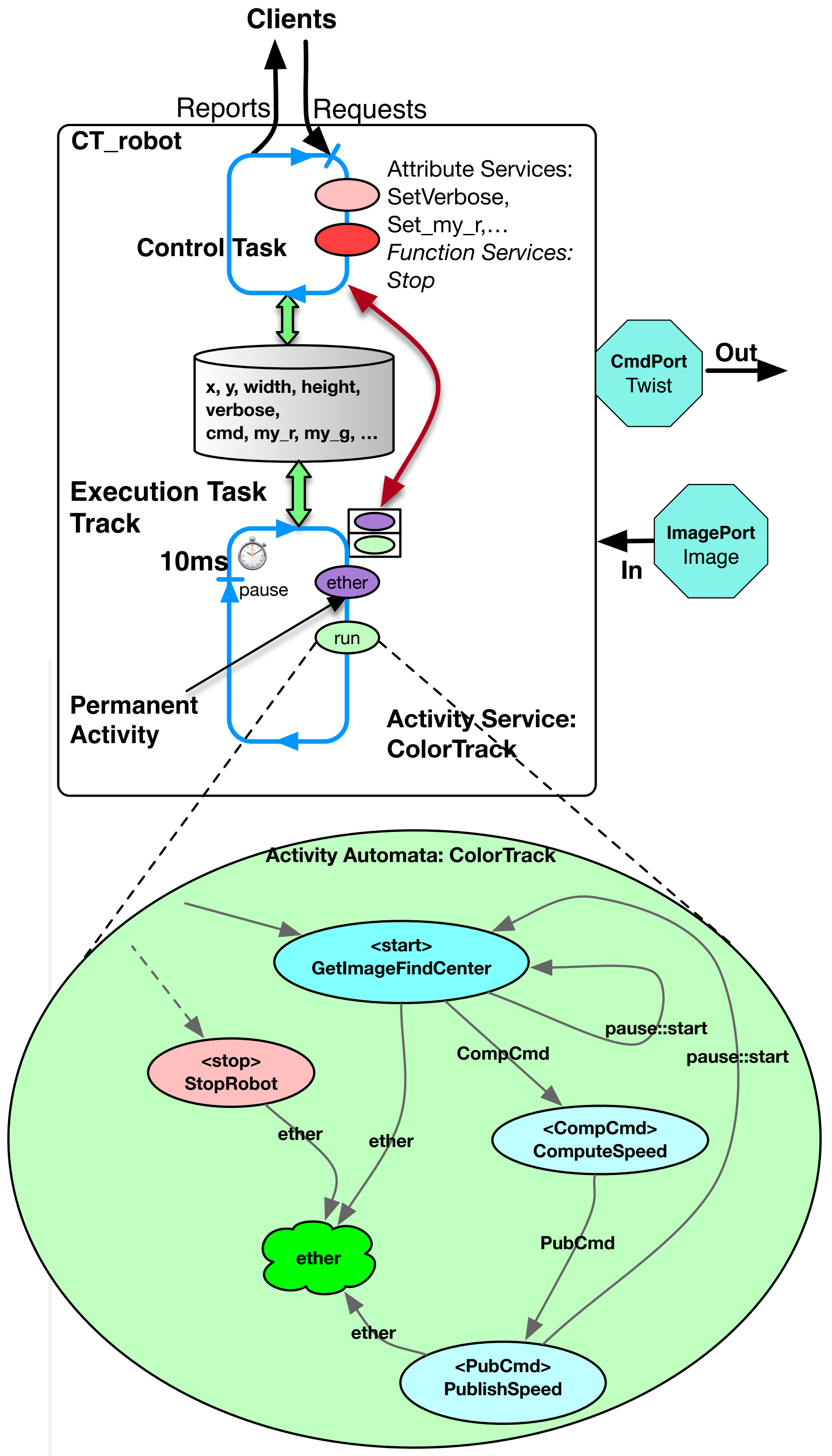


```
/* ------------------ TASK DEFINITION -------------------- */
task track {
  period           10 ms;     // fast, but we only process the image when it is new.
  codel <start>    InitIDS(port out CmdPort, ids out cmd, ids out x, ids out y) yield ether;
  codel <stop>     CleanIDS(port out CmdPort)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

CT_robot .gen

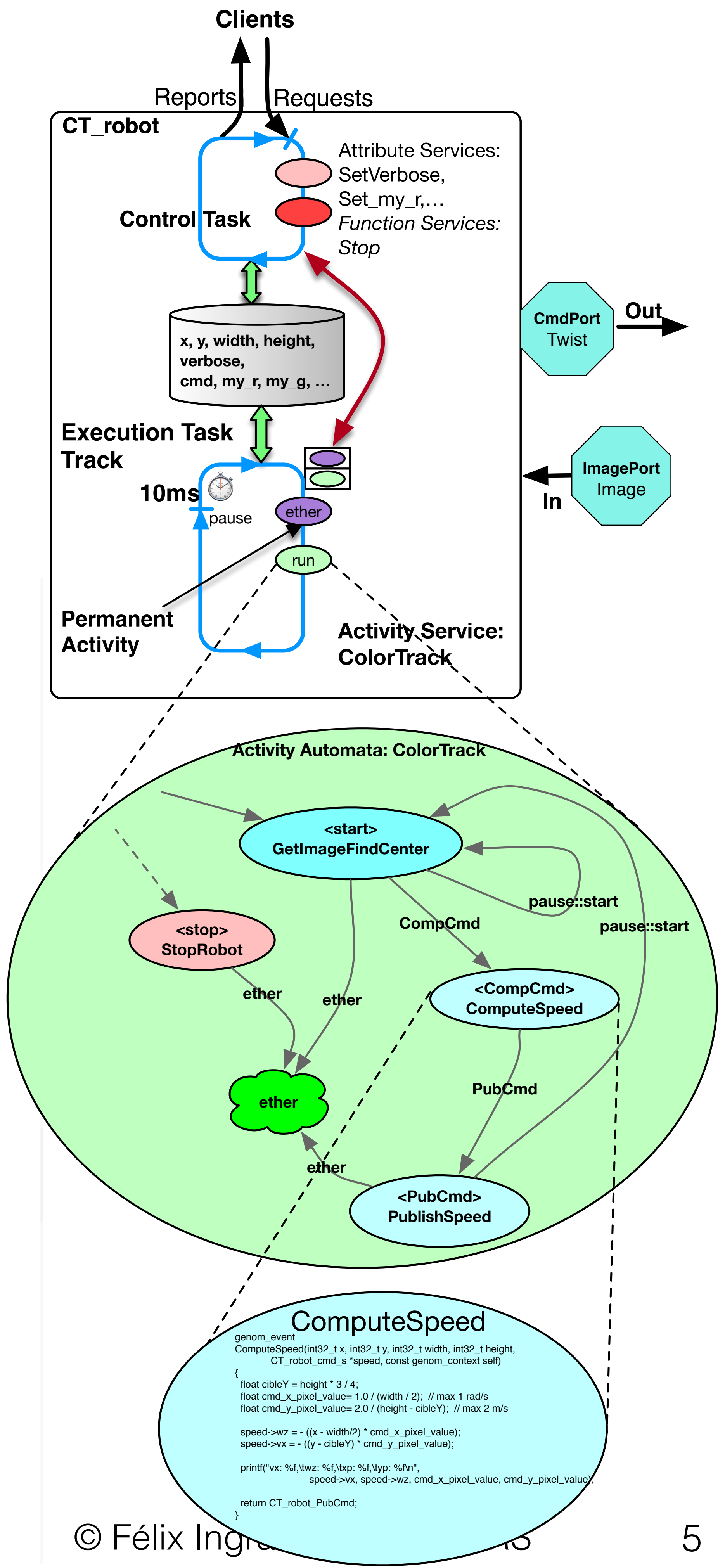


```
/* ----------------- SERVICES DEFINITION: The activities ------------------ */

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
 };
};
```



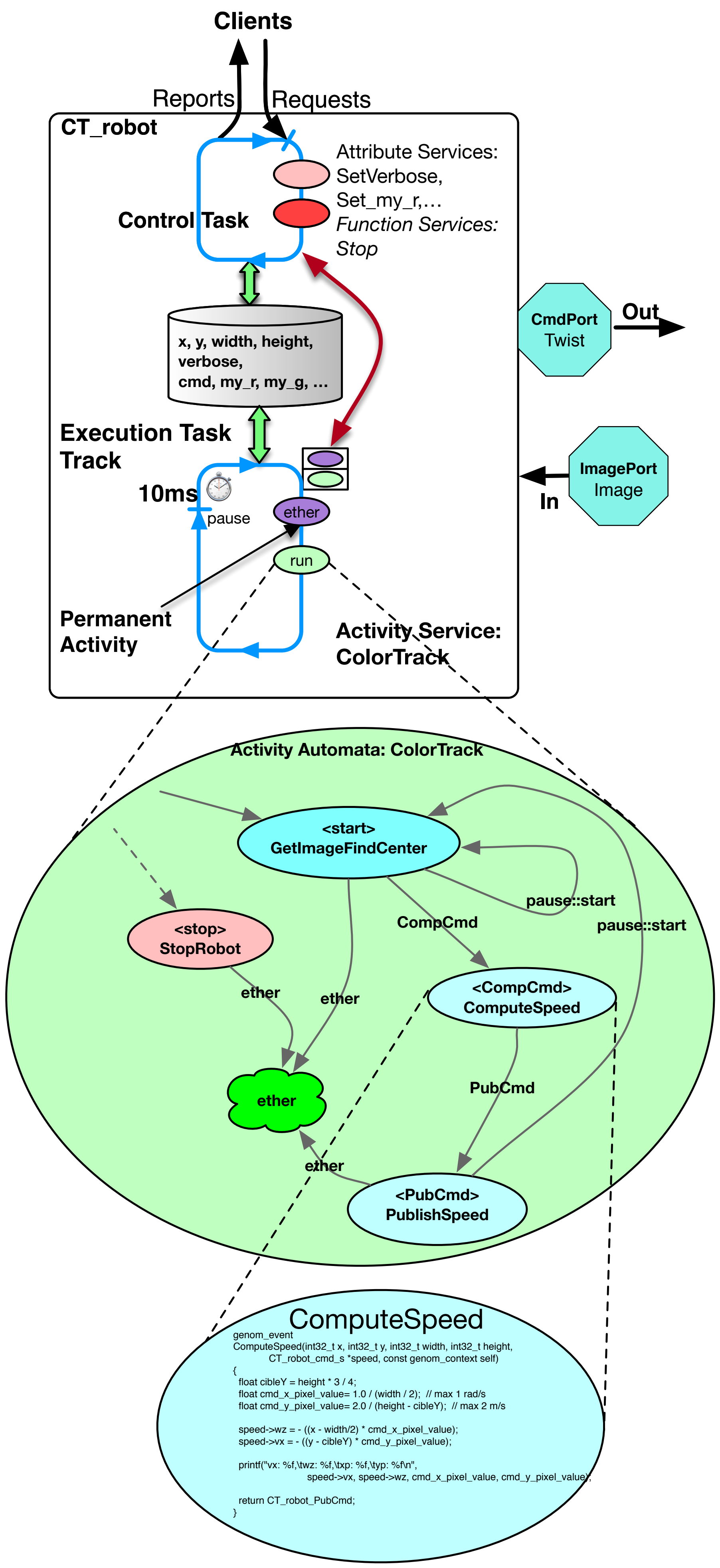
CT_robot .gen

```
/* ------------------ 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
};
};
```

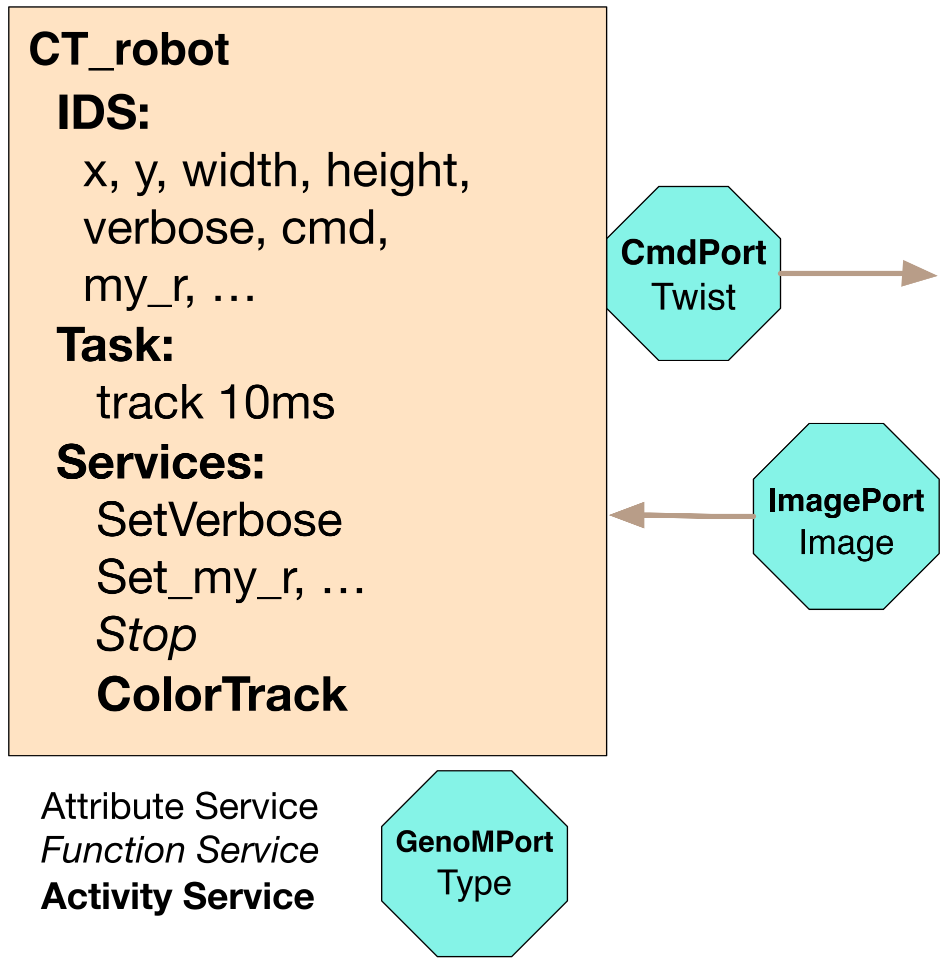


```
genom_event
ComputeSpeed(int32_t x, int32_t y, int32_t width, int32_t height,
        CT_robot_cmd_s *speed, const genom_context self)
{
  float cibleY = height * 3 / 4;
  float cmd_x_pixel_value= 1.0 / (width / 2);  // max 1 rad/s
  float cmd_y_pixel_value= 2.0 / (height - cibleY);  // max 2 m/s

  speed->wz = - ((x - width/2) * cmd_x_pixel_value);
  speed->vx = - ((y - cibleY) * cmd_y_pixel_value);

  printf("vx: %f,\twz: %f,\txp: %f,\typ: %f\n",
                 speed->vx, speed->wz, cmd_x_pixel_value, cmd_y_pixel_value);

  return CT_robot_PubCmd;
}
```

CT_robot

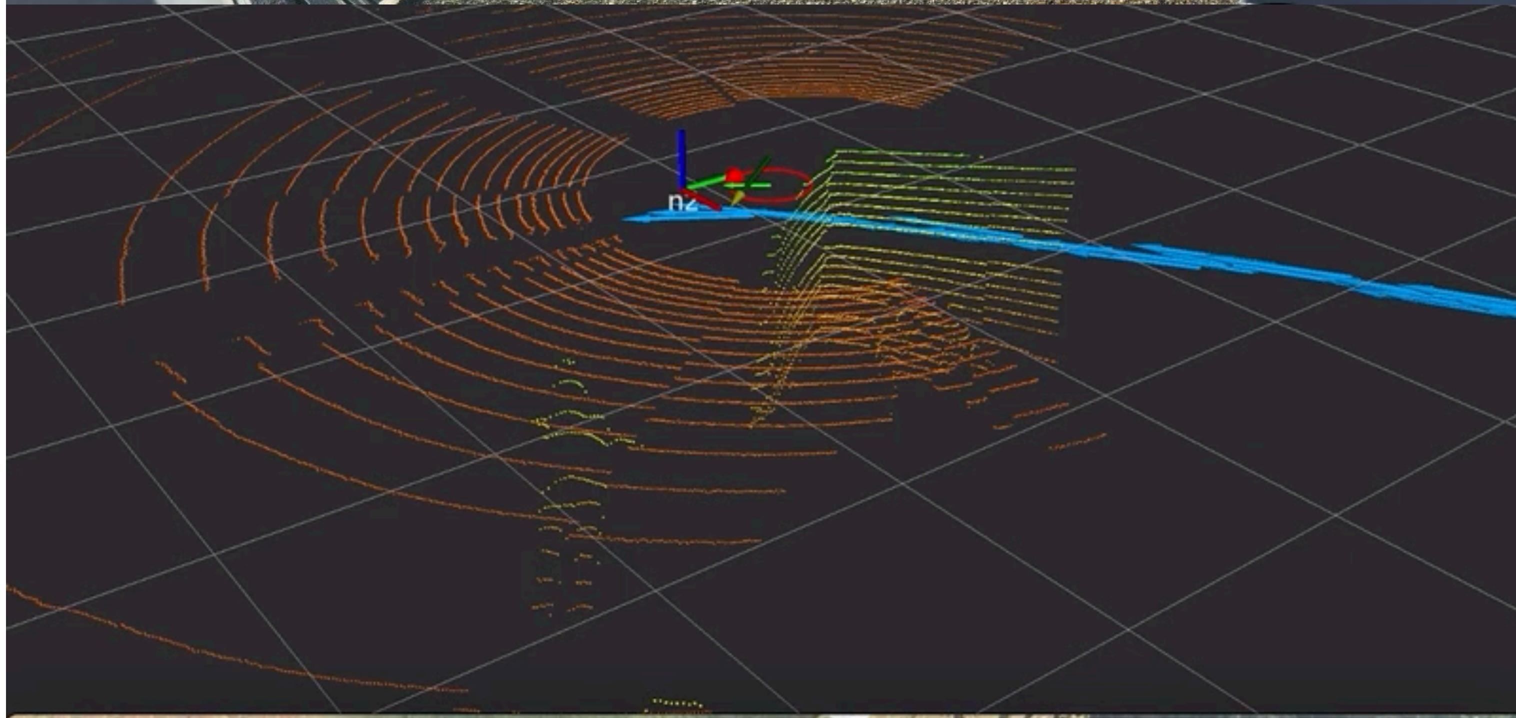
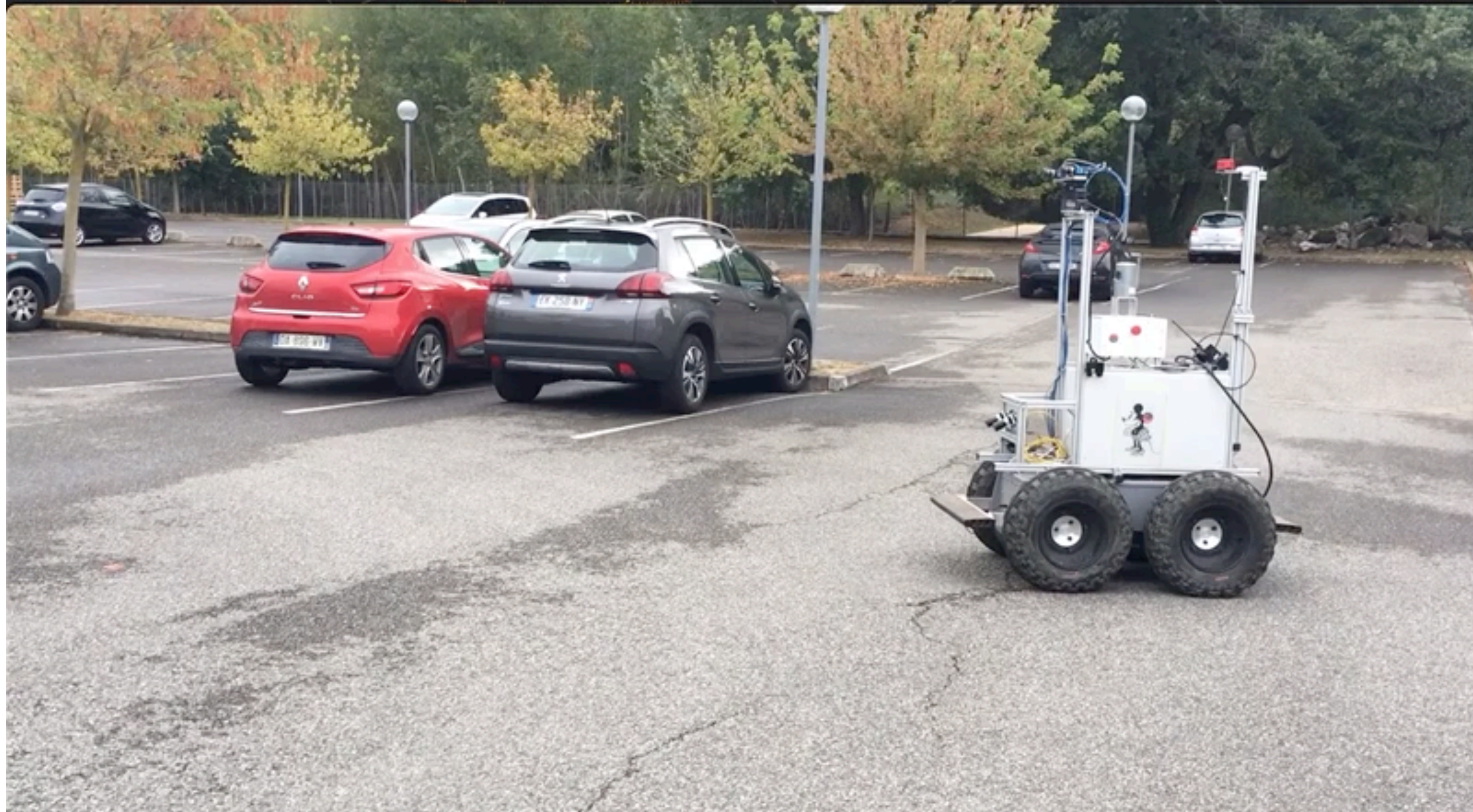


```
genom_event
ComputeSpeed(int32_t x, int32_t y, int32_t width, int32_t height,
         CT_robot_cmd_s *speed, const genom_context self)
{
  float cibleY = height * 3 / 4;
  float cmd_x_pixel_value= 1.0 / (width / 2);  // max 1 rad/s
  float cmd_y_pixel_value= 2.0 / (height - cibleY);  // max 2 m/s

  speed->wz = - ((x - width/2) * cmd_x_pixel_value);
  speed->vx = - ((y - cibleY) * cmd_y_pixel_value);

  printf("vx: %f,\twz: %f,\txp: %f,\typ: %f\n",
                   speed->vx, speed->wz, cmd_x_pixel_value, cmd_y_pixel_value);

  return CT_robot_PubCmd;
}
```



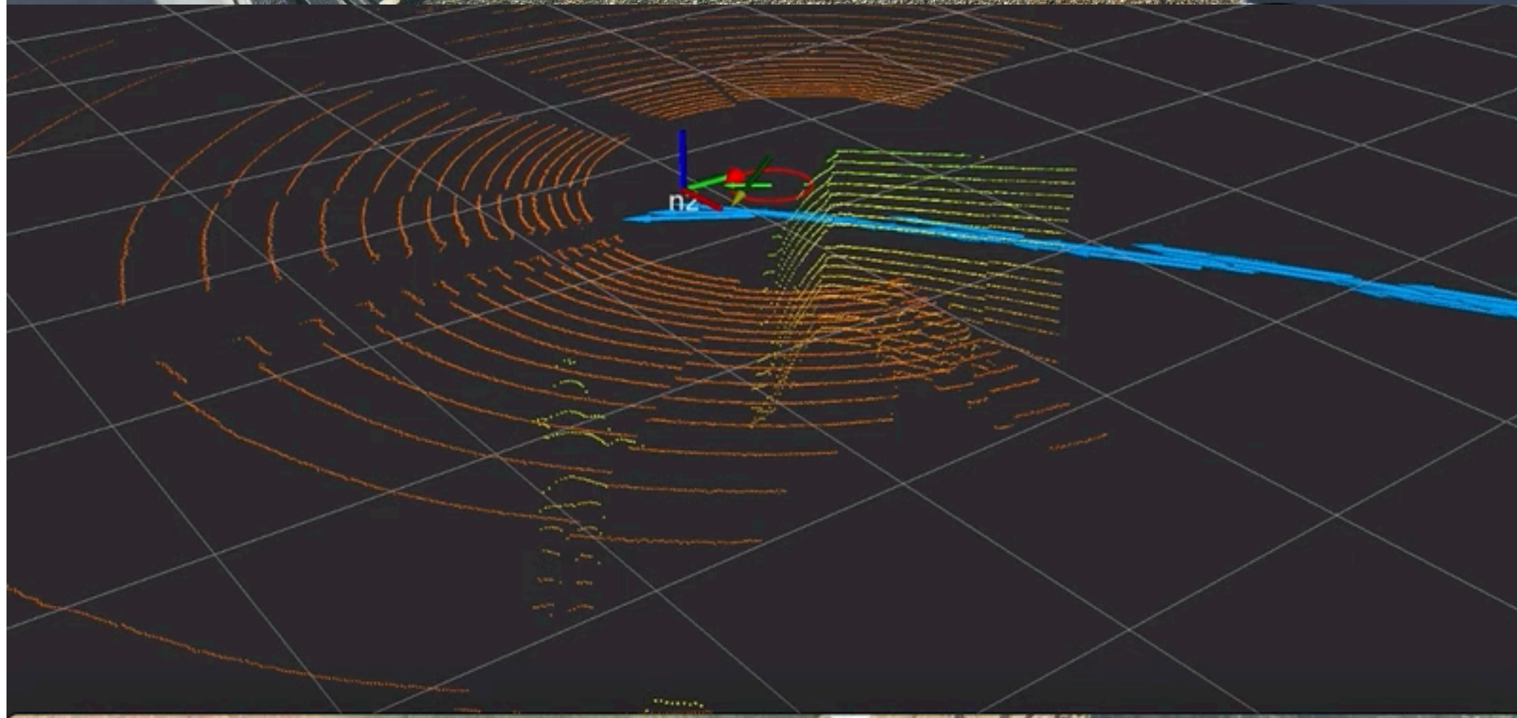
Segway RMP 440

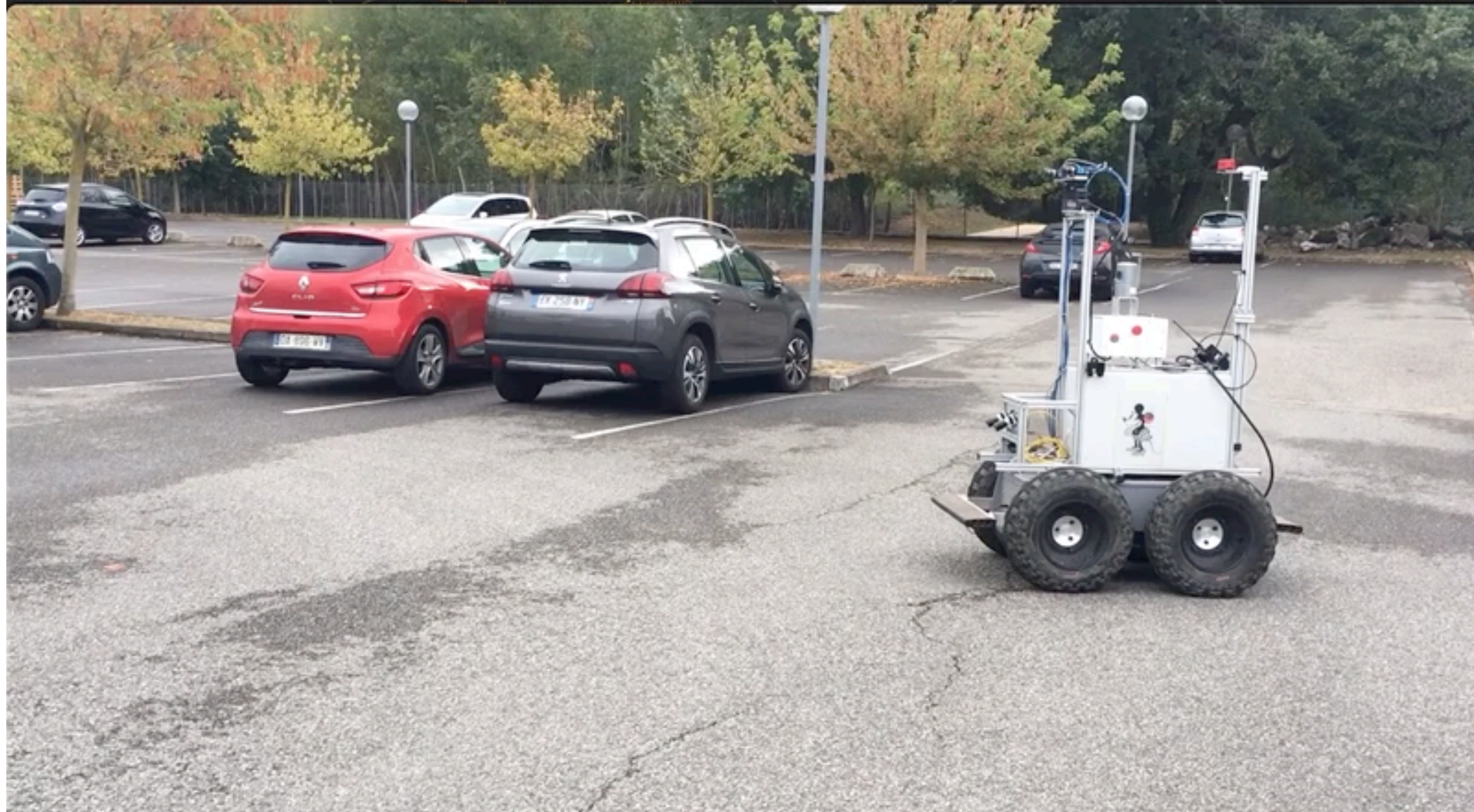
- Fast (up to 8 m/s)
- GPS
- Gyro (measures theta/wz)
- IMU (angular velocities and accelerations)
- Velodyne LIDAR
- 2 recent CPUs (but I only use one)

Segway RMP 440

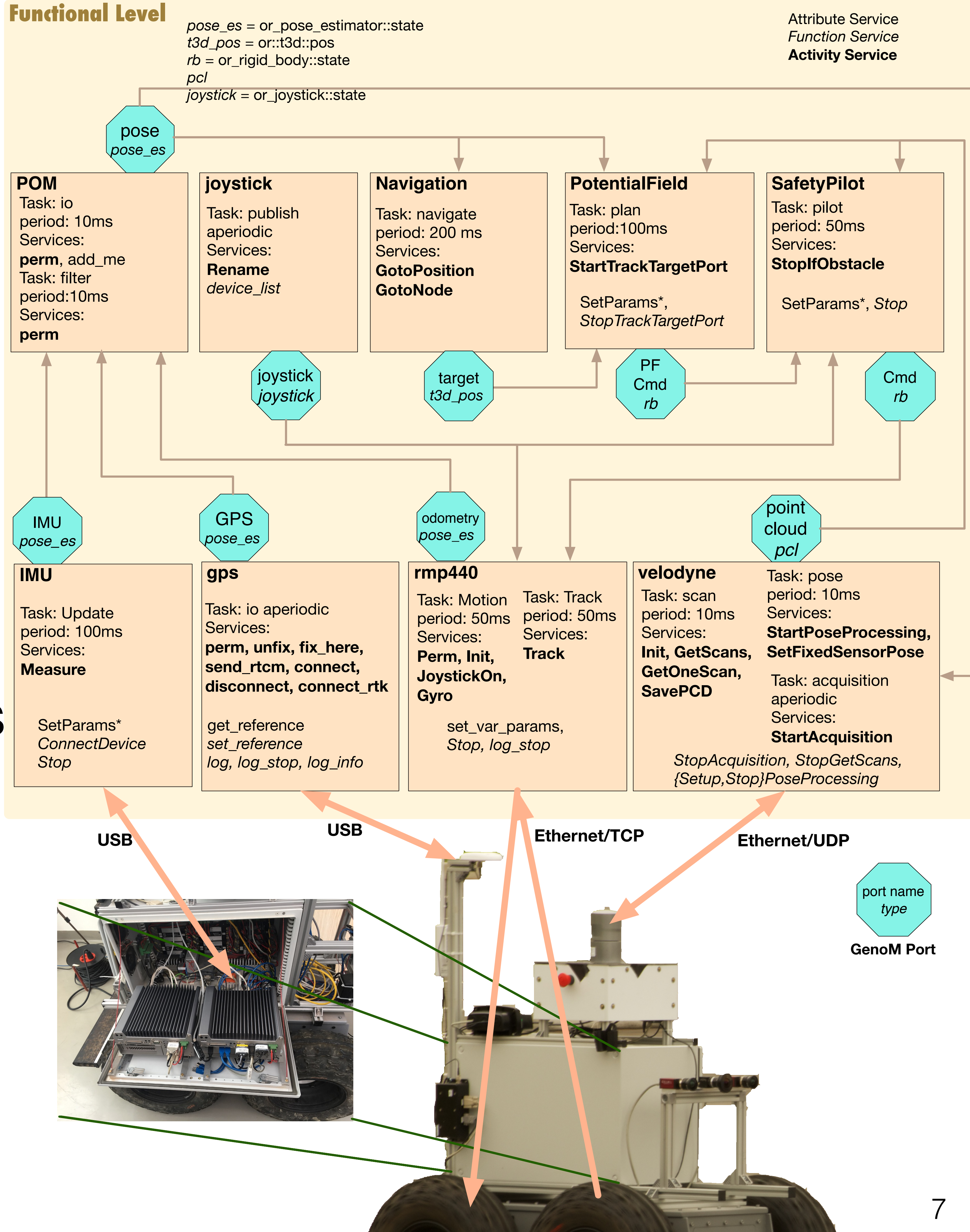
- Fast (up to 8 m/s)
- GPS
- Gyro (measures theta/wz)
- IMU (angular velocities and accelerations)
- Velodyne LIDAR
- 2 recent CPUs (but I only use one)



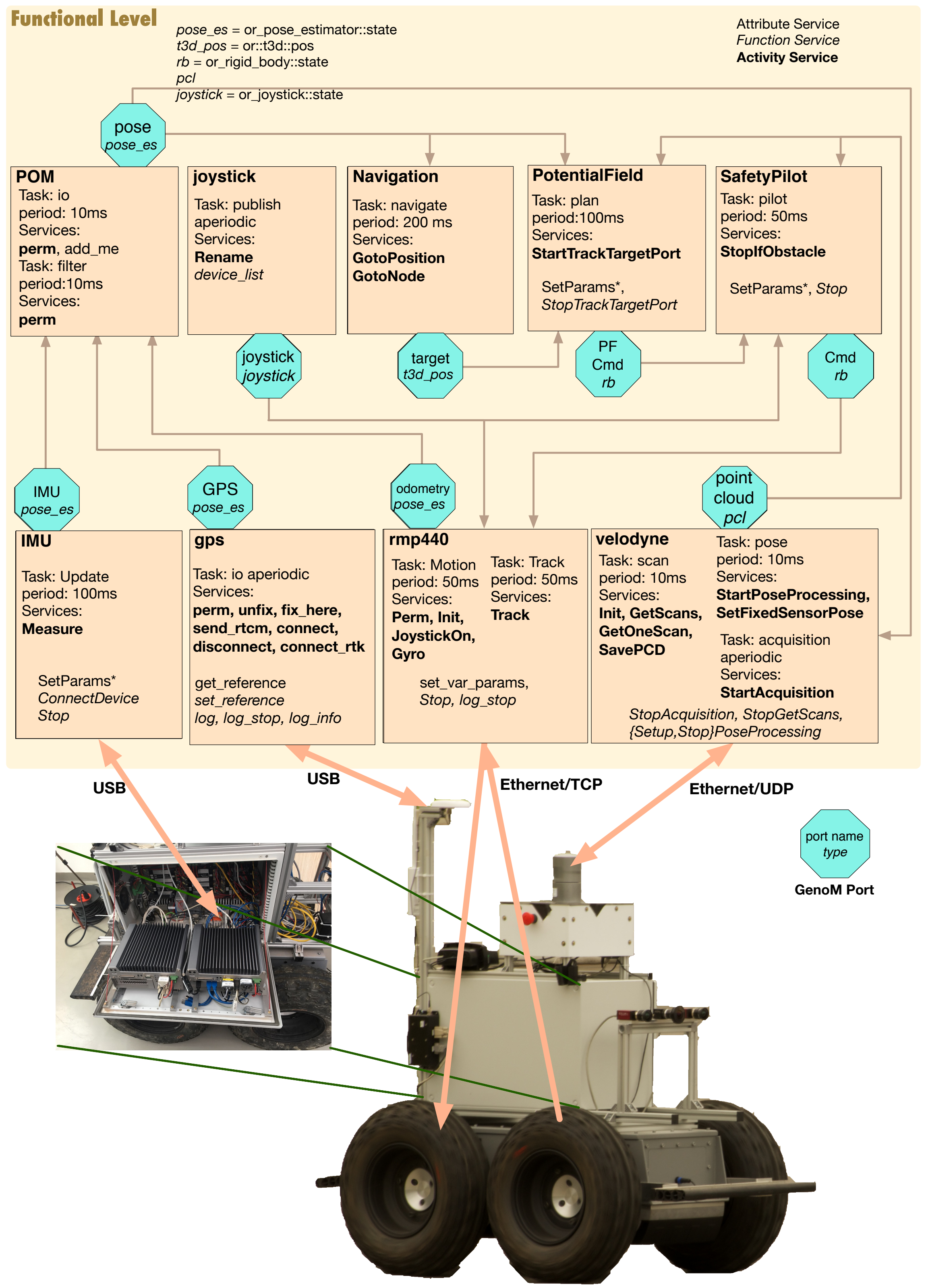


Segway RMP 440

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

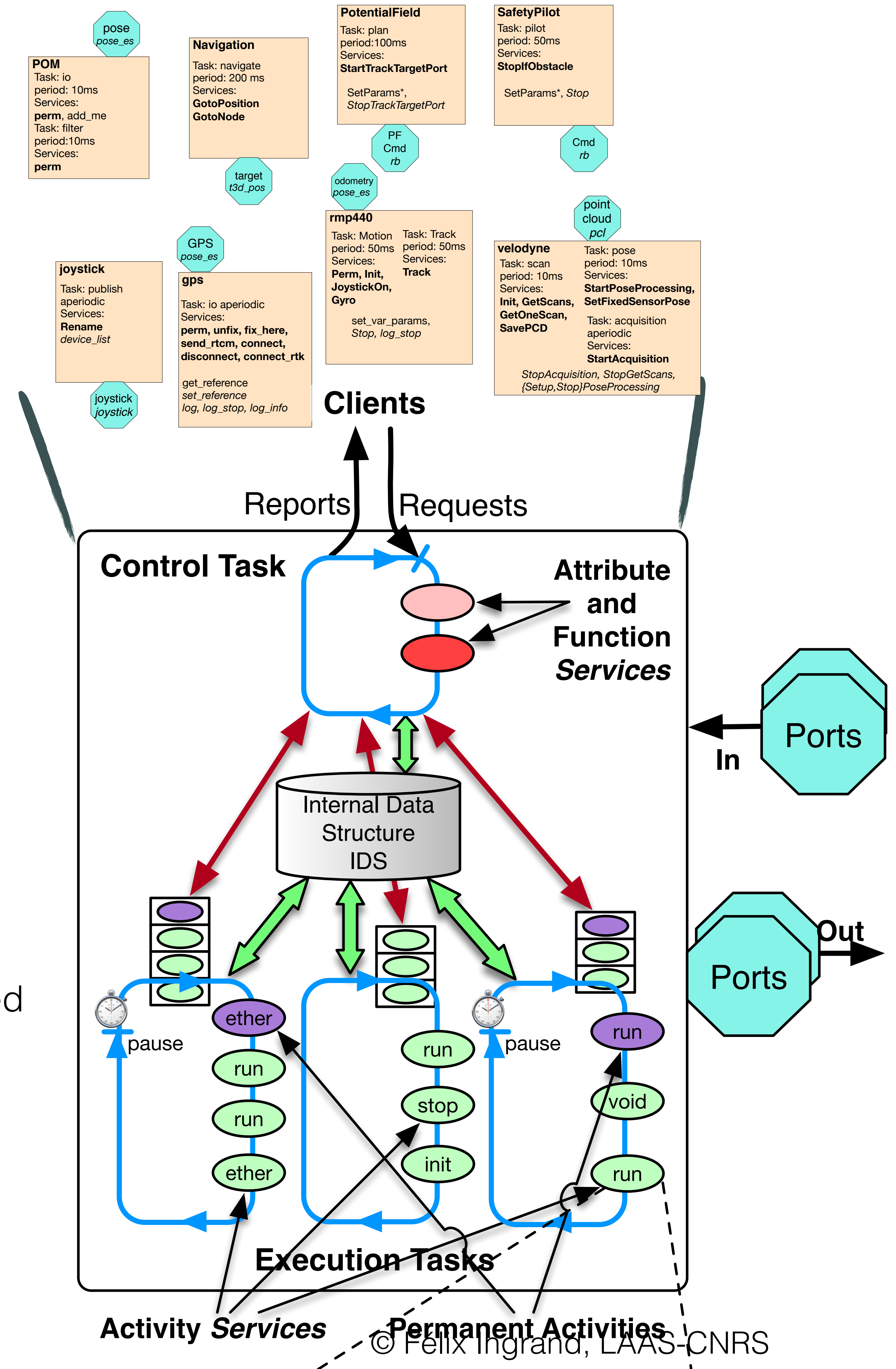

Functional Level
pose_es = or_pose_estimator::state
t3d_pos = or::t3d::pos
rb = or_rigid_body::state
pcl
joystick = or_joystick::state
Attribute Service
Function Service
Activity Service
pose pose_es
POM Task: io period: 10ms Services: perm, add_me Task: filter period:10ms Services: perm
joystick Task: 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 components specification for Minnie


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

GenoM internal

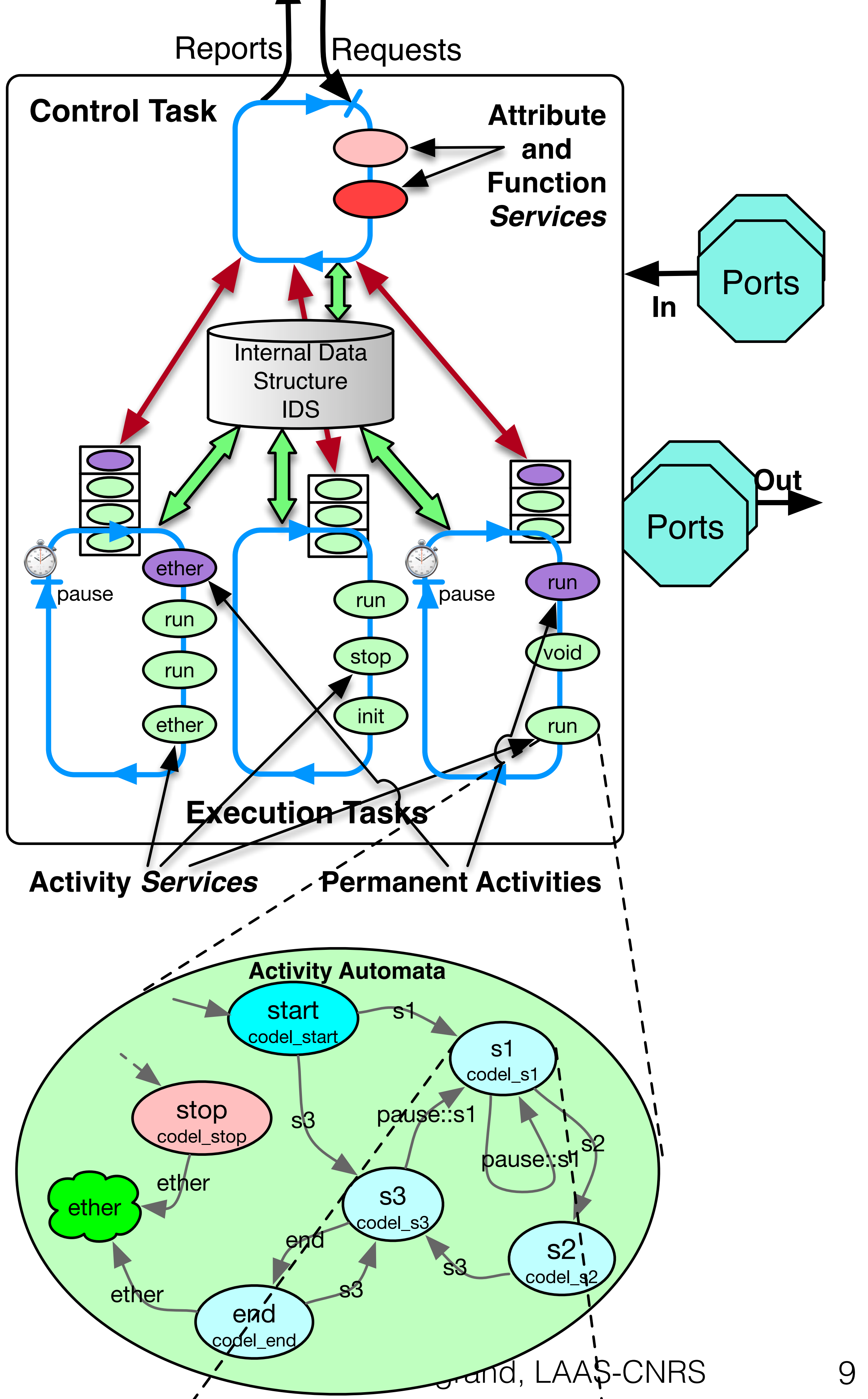
- A program with I/O
  - control: requests to start services/reports their results
  - data: ports in (to import external data) and out (to export data)
- A cyclic event based control task (aperiodic)
- One or more cyclic execution tasks, periodic or aperiodic
- It provides services (short and long computation) 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)



pose pose_es
POM Task: io period: 10ms Services: perm, add_me Task: filter period:10ms Services: perm
Navigation Task: navigate period: 200 ms Services: GotoPosition GotoNode
target t3d_pos
PotentialField Task: plan period:100ms Services: StartTrackTargetPort SetParams*, StopTrackTargetPort
PF Cmd rb
SafetyPilot Task: pilot period: 50ms Services: StopIfObstacle SetParams*, Stop
Cmd rb
odometry pose_es
rmp440 Task: Motion period: 50ms Services: Perm, Init, JoystickOn, Gyro Task: Track period: 50ms Services: Track set_var_params, Stop, log_stop
point cloud pcl
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
GPS pose_es
gps Task: io aperiodic Services: perm, unfix, fix_here, send_rtcm, connect, disconnect, connect_rtk get_reference set_reference log, log_stop, log_info
joystick Task: publish aperiodic Services: Rename device_list
joystick joystick
Clients
Reports
Requests
Control Task
Attribute and Function Services
Ports
In
Internal Data Structure IDS
Ports
Out
pause
ether
run
run
ether
run
stop
init
pause
run
void
run
Execution Tasks
Activity Services
Permanent Activities

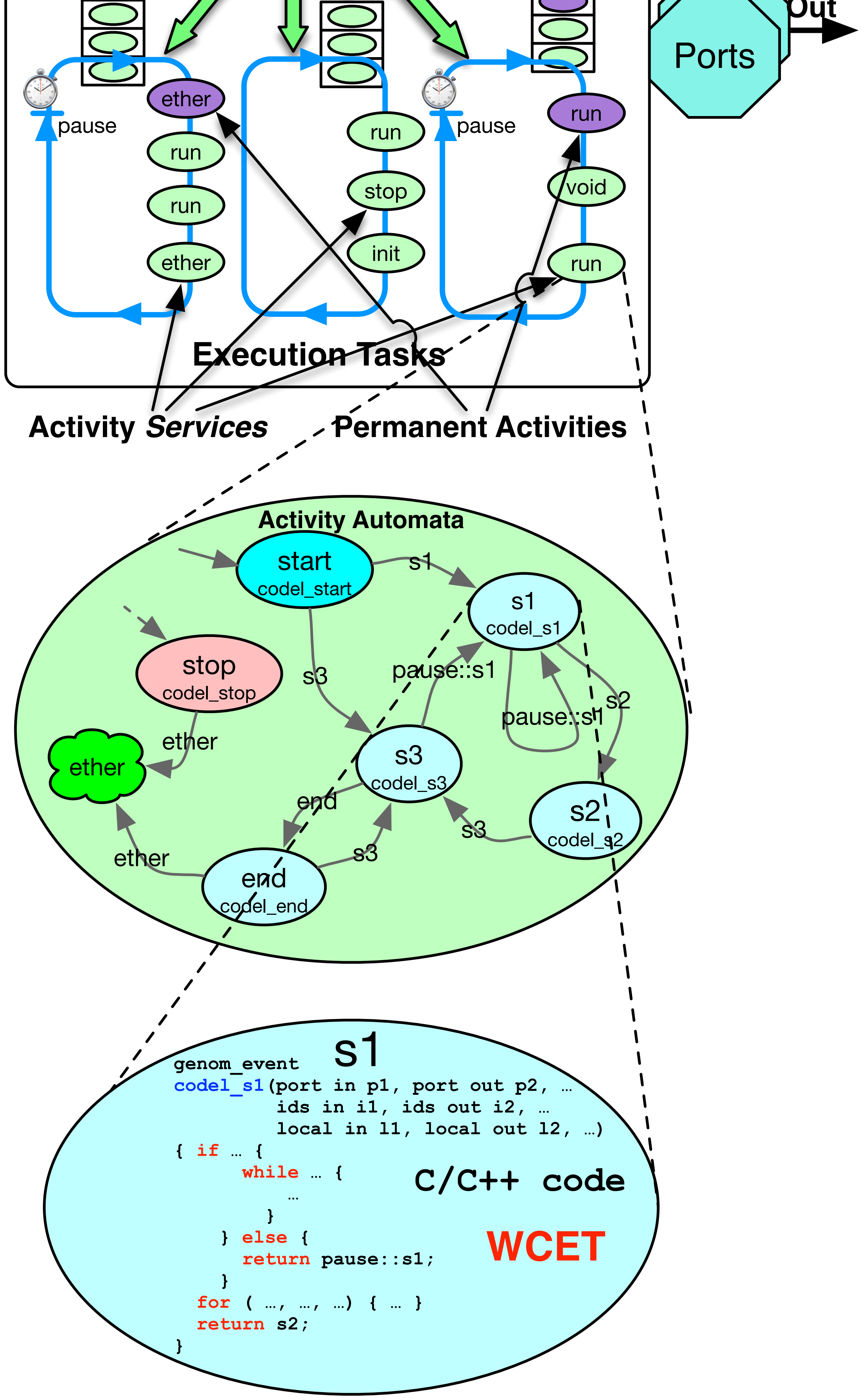
GenoM internal

- 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



GenoM internal

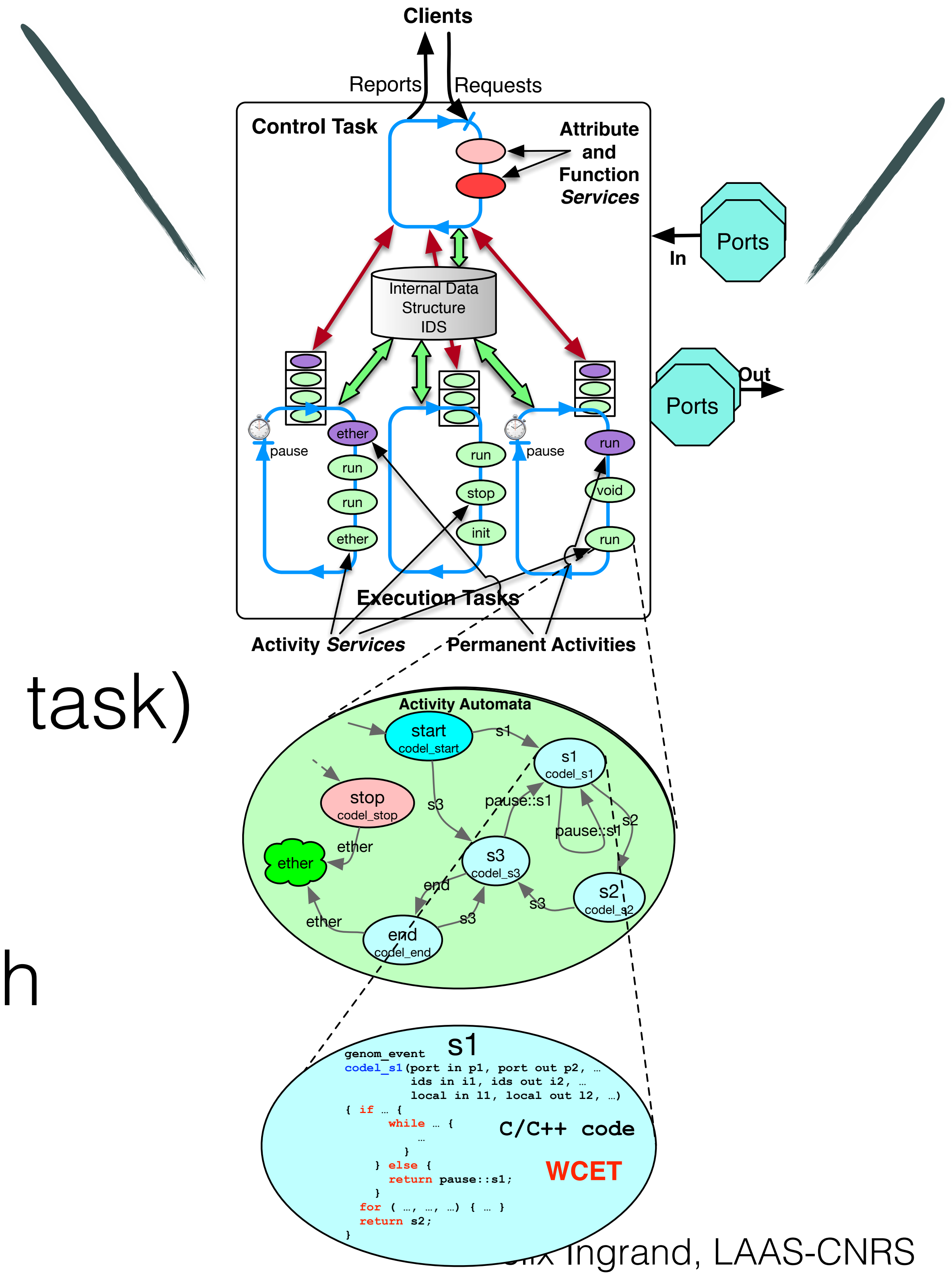
- A program with I/O
    - control: requests to start services/reports their results
    - data: ports in (to import external data) and out (to export data)
- A cyclic event based control task (aperiodic)
- One or more cyclic execution tasks, periodic or aperiodic
- It provides services (short and long 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)



```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
          …
        }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

GenoM specifications

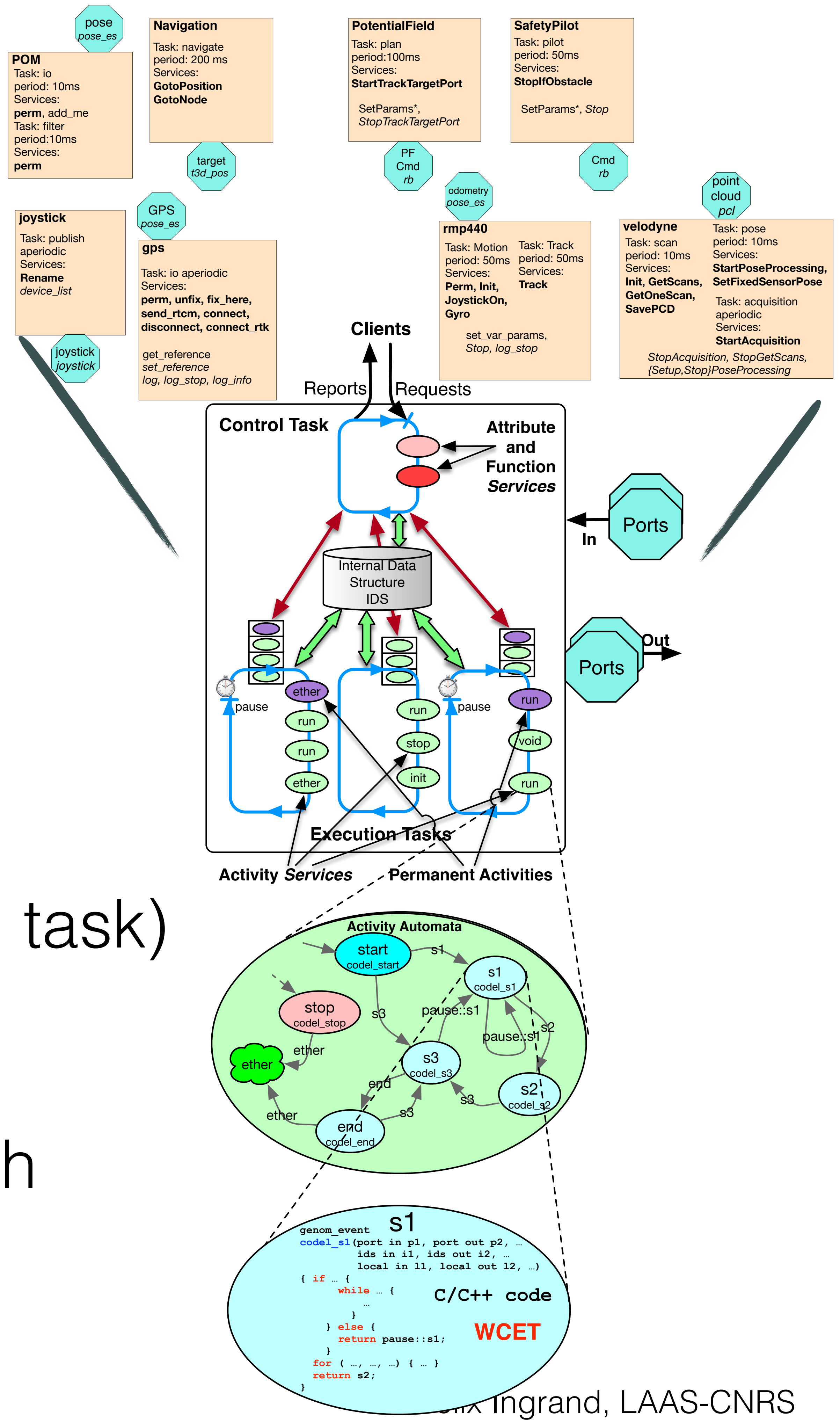
- IDS
- Ports (in & out)
- Execution Tasks
  - (periodic or aperiodic)
- Services
  - Attribute, function (control task)
  - Activity (execution task)
    - automata
    - and attached codels with WCET



```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
         …
        }
     } else {
      return pause::s1;
     }
  for ( …, …, …) { … }
  return s2;
}
```

GenoM specifications

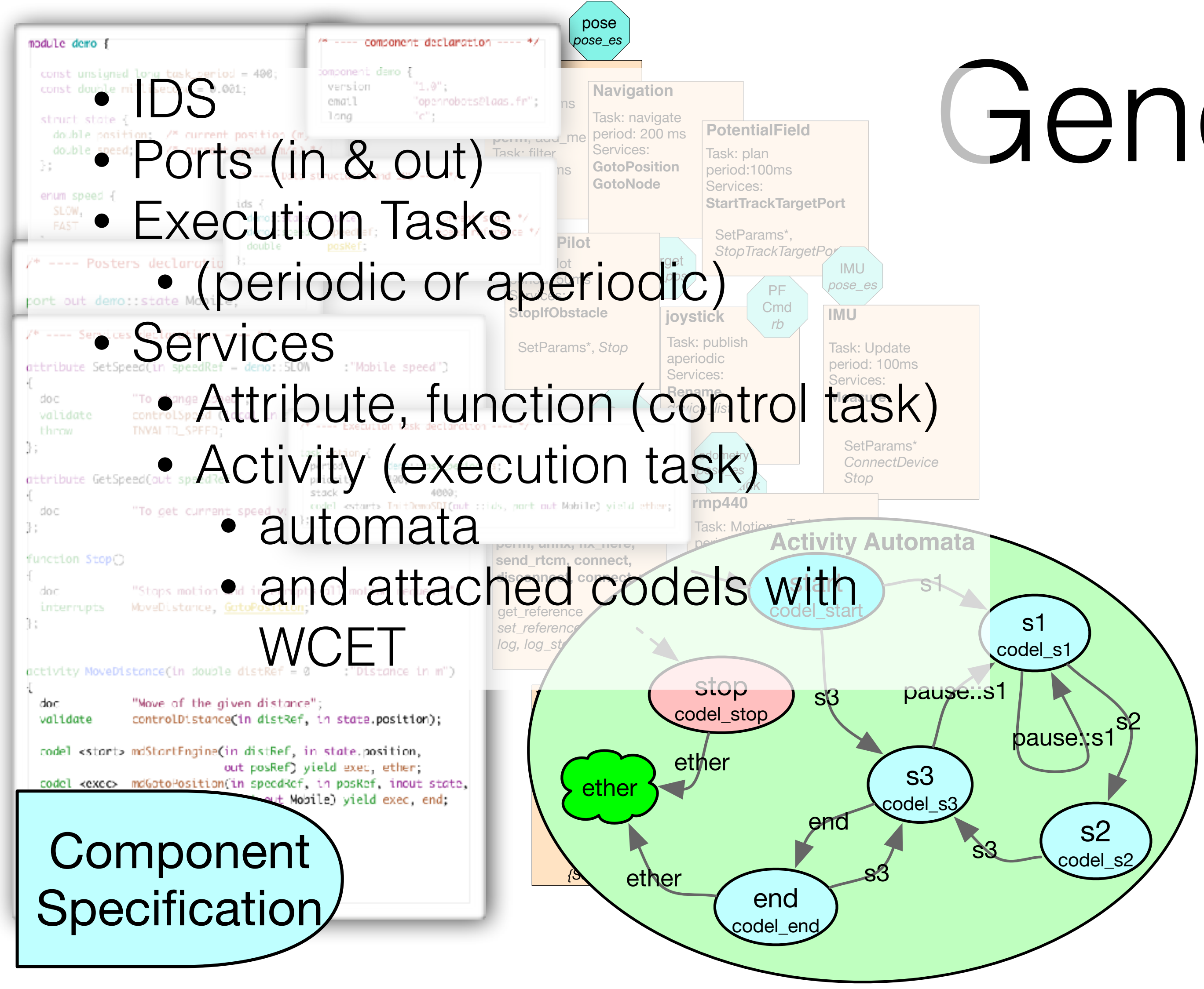
- IDS
- Ports (in & out)
- Execution Tasks
  - (periodic or aperiodic)
- Services
  - Attribute, function (control task)
  - Activity (execution task)
    - automata
    - and attached codels with WCET



pose pose_es
POM Task: io period: 10ms Services: perm, add_me Task: filter period:10ms Services: perm
Navigation Task: navigate period: 200 ms Services: GotoPosition GotoNode
target t3d_pos
PotentialField Task: plan period:100ms Services: StartTrackTargetPort SetParams*, StopTrackTargetPort
PF Cmd rb
SafetyPilot Task: pilot period: 50ms Services: StopIfObstacle SetParams*, Stop
Cmd rb
odometry pose_es
point cloud pcl
joystick Task: publish aperiodic Services: Rename device_list
joystick joystick
GPS pose_es
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
Clients
Reports
Requests
Control Task
Attribute and Function Services
In
Ports
Out
Ports
Internal Data Structure IDS
pause
ether
run
run
ether
run
stop
init
pause
run
void
run
Execution Tasks
Activity Services
Permanent Activities
Activity Automata
start codel_start
s1
s1 codel_s1
stop codel_stop
s3
pause::s1
s2
pause::s1
ether
ether
s3 codel_s3
end
s3
s2 codel_s2
ether
s3
end codel_end
s1
C/C++ code
WCET

```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
          …
        }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

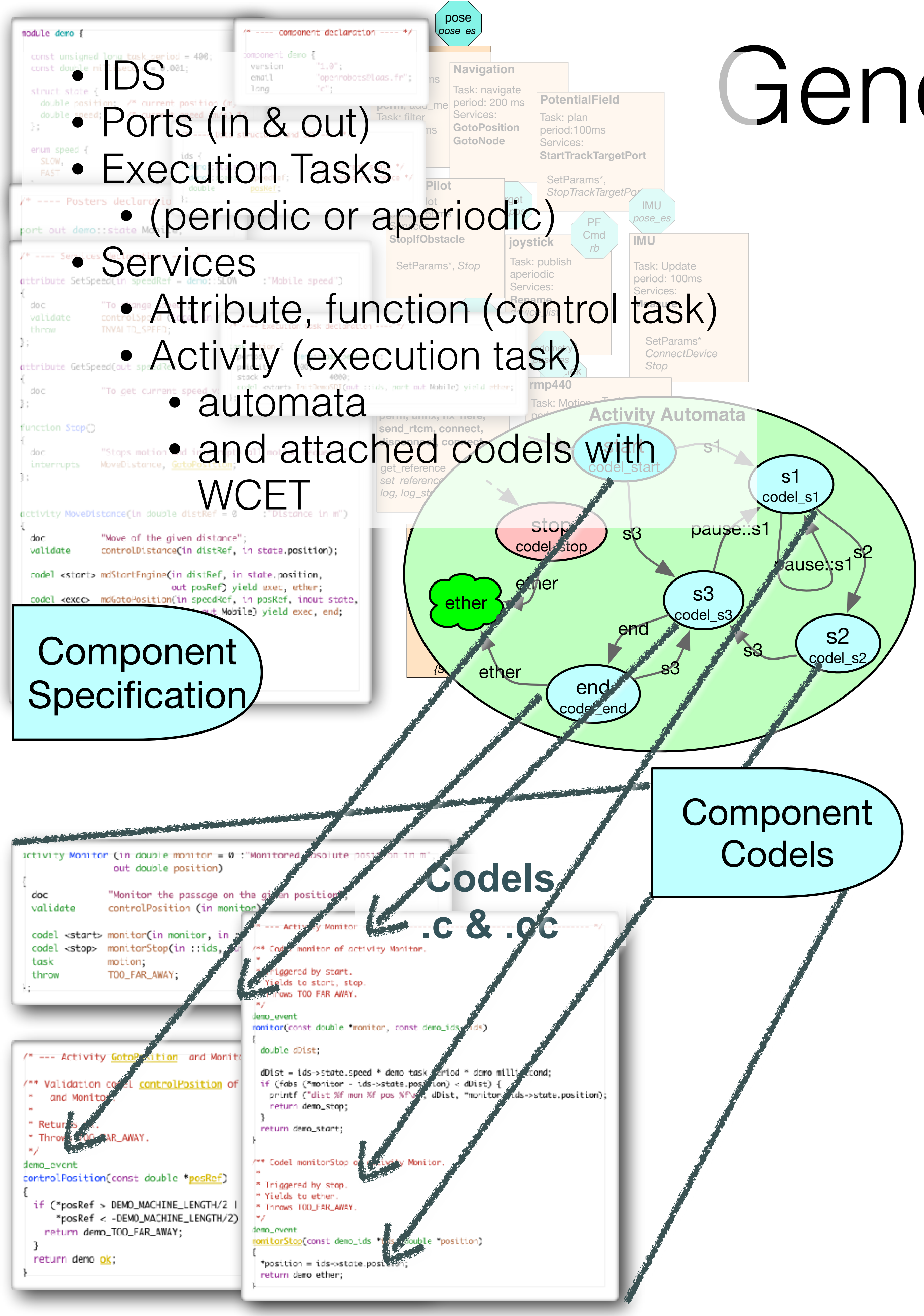
GenoM workflow



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

GenoM workflow

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



```
doc          "Move of the given distance";
validate     controlDistance(in distRef, in state.position);

codel <start> mdStartEngine(in distRef, in state.position,
                            out posRef) yield exec, ether;
codel <exec>  mdGotoPosition(in speedRef, in posRef, inout state,
                            out Mobile) yield exec, end;
```

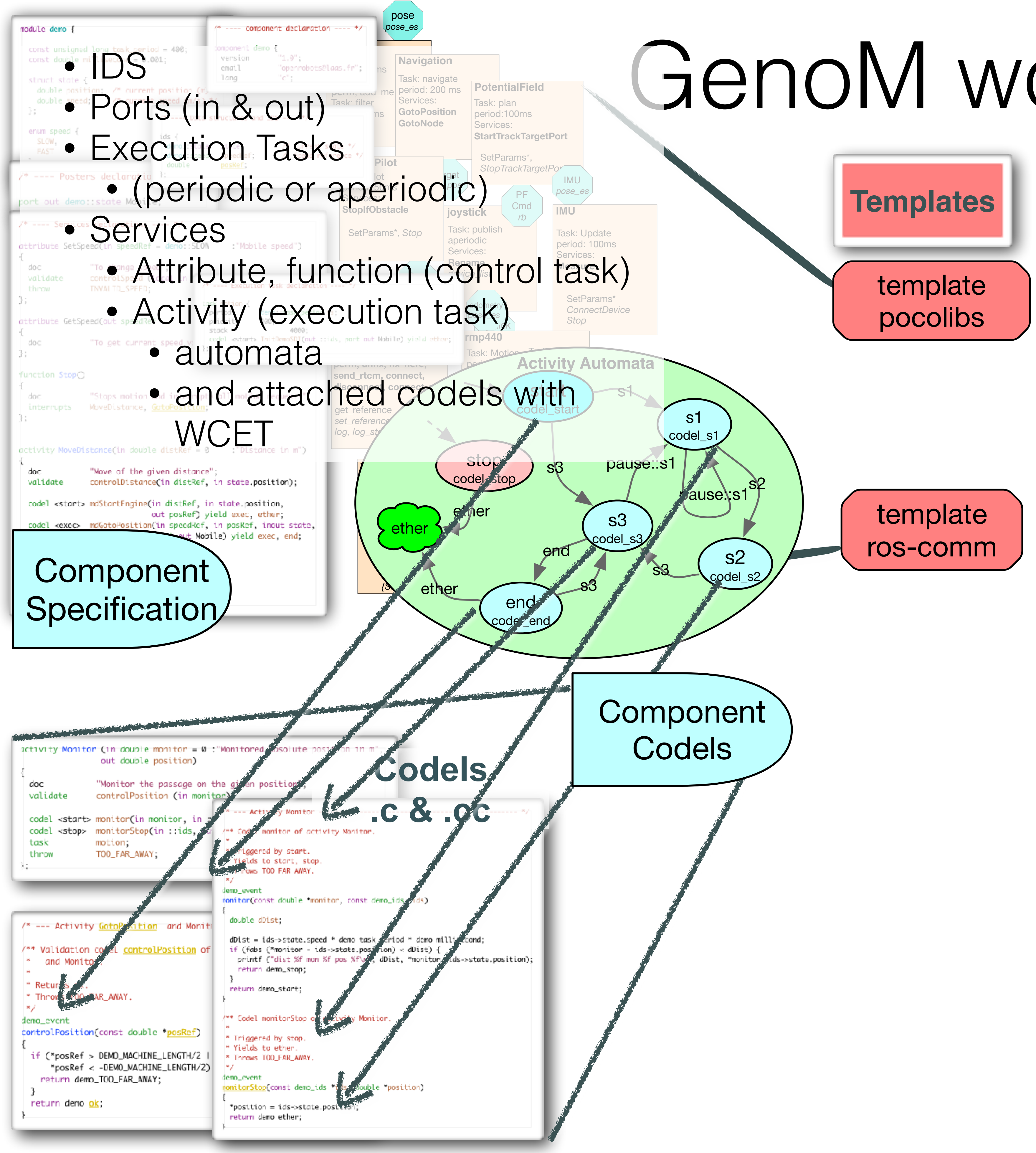
```
activity Monitor (in double monitor = 0 :"Monitored absolute position in m",
                  out double position)
{
  doc             "Monitor the passage on the given position";
  validate        controlPosition (in monitor);

  codel <start>   monitor(in monitor, in
  codel <stop>    monitorStop(in ::ids,
  task            motion;
  throw           TOO_FAR_AWAY;
};
```

```
/* --- Activity GotoPosition  and Monit

/** Validation codel controlPosition of
 *   and Monitor.
 *
 * Returns OK.
 * Throws TOO_FAR_AWAY.
 */
demo_event
controlPosition(const double *posRef)
{
  if (*posRef > DEMO_MACHINE_LENGTH/2 |
      *posRef < -DEMO_MACHINE_LENGTH/2)
    return demo_TOO_FAR_AWAY;
  }
  return demo ok;
}
```

```
/* --- Activity Monitor ---------------------------------------------- */

/** Codel monitor of activity Monitor.
 *
 * Triggered by start.
 * Yields to start, stop.
 * Throws TOO FAR AWAY.
 */
demo_event
monitor(const double *monitor, const demo_ids *ids)
{
  double dDist;

  dDist = ids->state.speed * demo task period * demo millisecond;
  if (fabs (*monitor - ids->state.position) < dDist) {
    printf ("dist %f mon %f pos %f\n", dDist, *monitor, ids->state.position);
    return demo_stop;
  }
  return demo_start;
}

/** Codel monitorStop of activity Monitor.
 *
 * Triggered by stop.
 * Yields to ether.
 * Throws TOO_FAR_AWAY.
 */
demo_event
monitorStop(const demo_ids *ids, double *position)
{
  *position = ids->state.position;
  return demo ether;
}
```

GenoM workflow

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



```
activity MoveDistance(in double distRef = 0    :"Distance in m")
{
  doc          "Move of the given distance";
  validate     controlDistance(in distRef, in state.position);

  codel <start> mdStartEngine(in distRef, in state.position,
                               out posRef) yield exec, ether;
  codel <exec>  mdGotoPosition(in speedRef, in posRef, inout state,
                               out Mobile) yield exec, end;
```

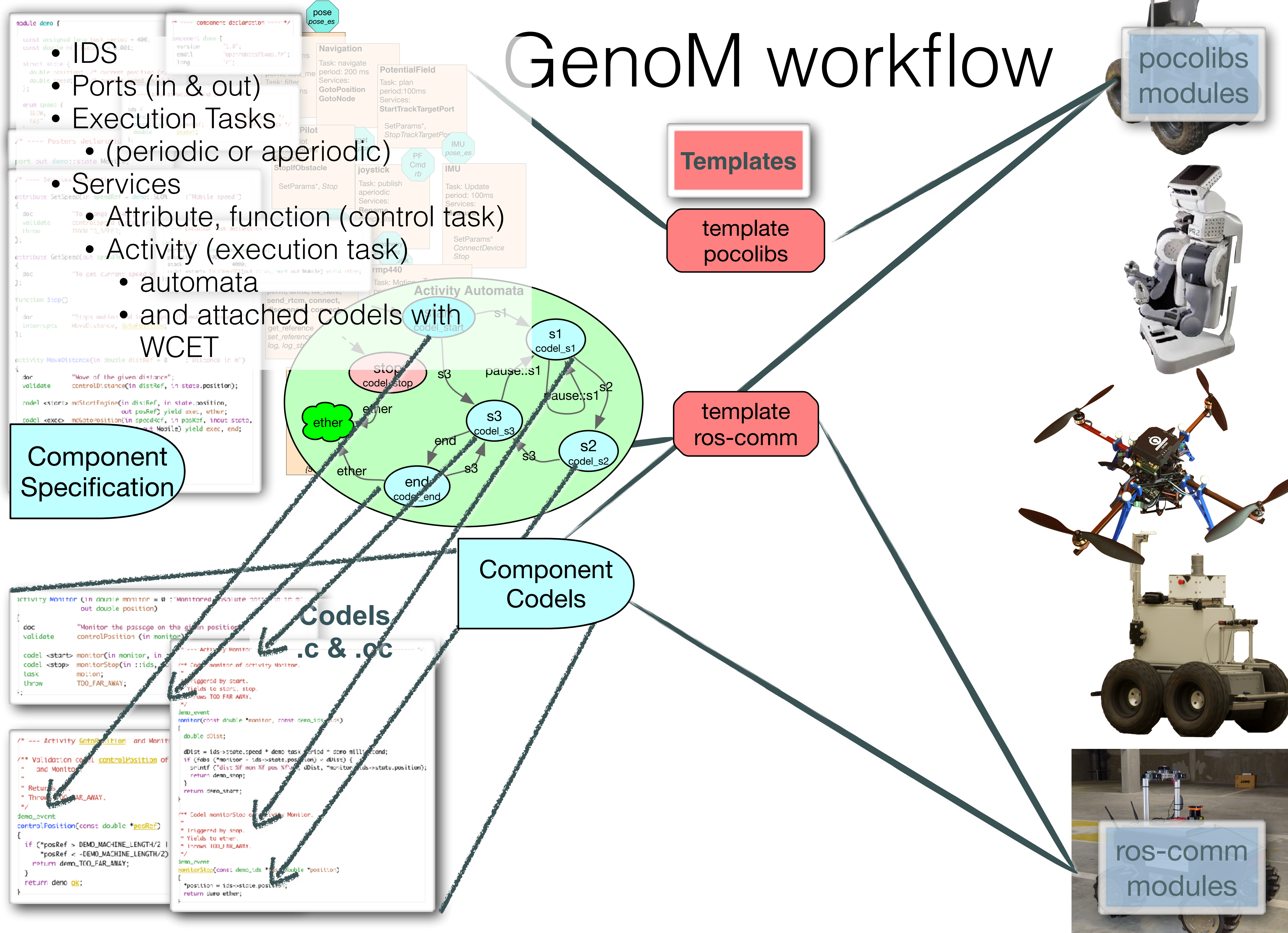
```
activity Monitor (in double monitor = 0 :"Monitored absolute position in m",
                  out double position)
{
  doc          "Monitor the passage on the given position";
  validate     controlPosition (in monitor);

  codel <start> monitor(in monitor, in [illegible]
  codel <stop>  monitorStop(in ::ids, [illegible]
  task          motion;
  throw         TOO_FAR_AWAY;
};
```

```
/* --- Activity GotoPosition  and Monit[illegible]

/** Validation codel controlPosition of [illegible]
 *   and Monitor.
 *
 * Returns [illegible]
 * Throws TOO_FAR_AWAY.
 */
demo_event
controlPosition(const double *posRef)
{
  if (*posRef > DEMO_MACHINE_LENGTH/2 ||
      *posRef < -DEMO_MACHINE_LENGTH/2)
    return demo_TOO_FAR_AWAY;
  }
  return demo_ok;
}
```

```
/* --- Activity Monitor ---------------------------------------------- */

/** Codel monitor of activity Monitor.
 *
 * Triggered by start.
 * Yields to start, stop.
 * Throws TOO FAR AWAY.
 */
demo_event
monitor(const double *monitor, const demo_ids *ids)
{
  double dDist;

  dDist = ids->state.speed * demo task_period * demo millisecond;
  if (fabs (*monitor - ids->state.position) < dDist) {
    printf ("dist %f mon %f pos %f\n", dDist, *monitor, ids->state.position);
    return demo_stop;
  }
  return demo_start;
}

/** Codel monitorStop of activity Monitor.
 *
 * Triggered by stop.
 * Yields to ether.
 * Throws TOO_FAR_AWAY.
 */
demo_event
monitorStop(const demo_ids *ids, double *position)
{
  *position = ids->state.position;
  return demo ether;
}
```

GenoM workflow

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

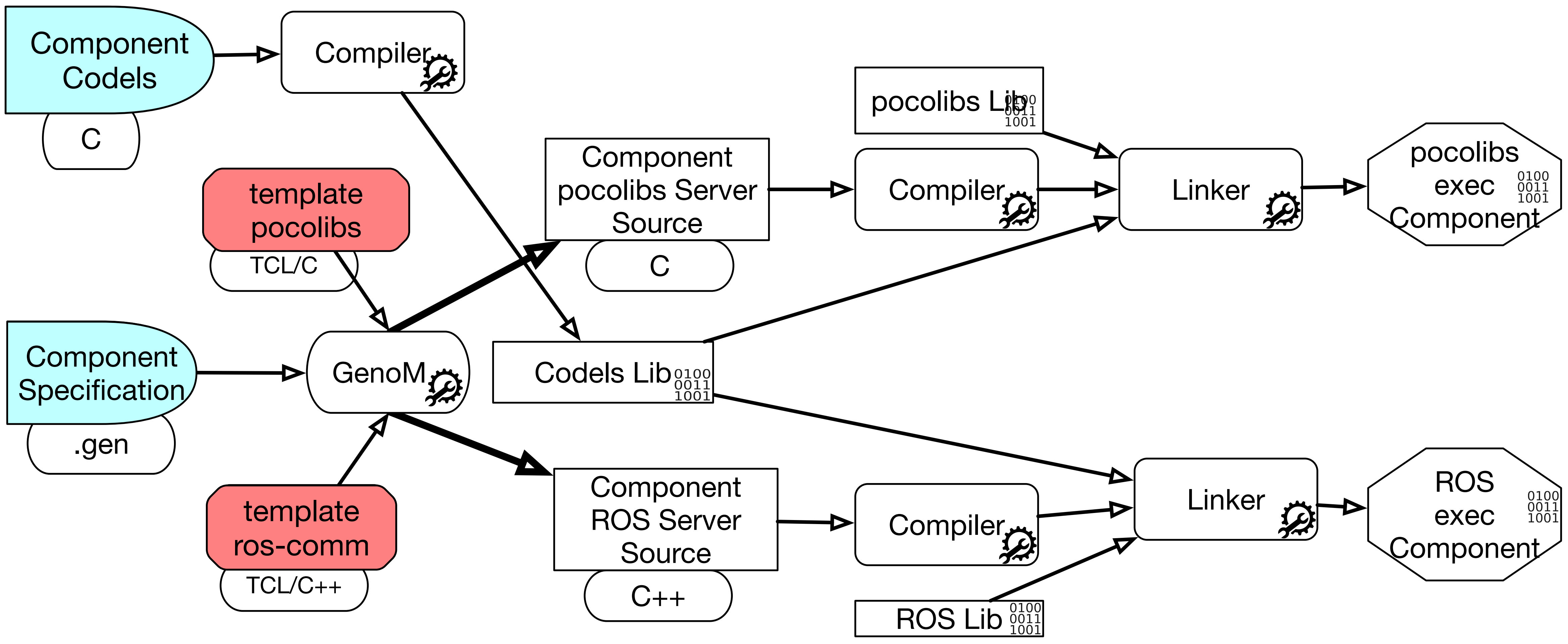


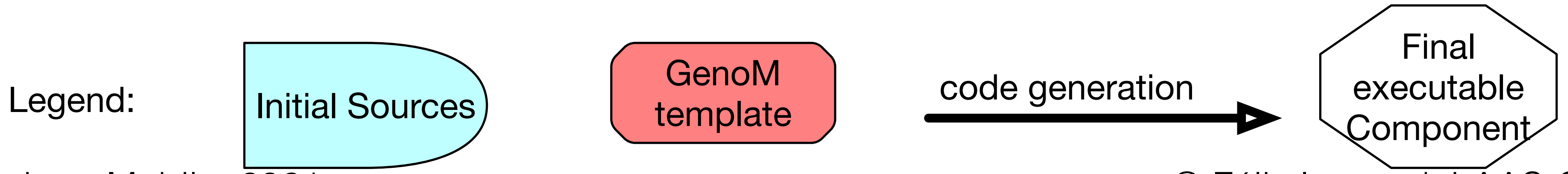
```
activity Monitor (in double monitor = 0 :"Monitored absolute position in m",
                  out double position)
{
  doc          "Monitor the passage on the given position",
  validate     controlPosition (in monitor);

  codel <start> monitor(in monitor, in
  codel <stop>  monitorStop(in ::ids,
  task          motion;
  throw         TOO_FAR_AWAY;
};
```

```
/* --- Activity GotoPosition  and Monit

/** Validation codel controlPosition of
 *   and Monitor.
 *
 * Returns
 * Throws TOO_FAR_AWAY.
 */
demo_event
controlPosition(const double *posRef)
{
  if (*posRef > DEMO_MACHINE_LENGTH/2 |
      *posRef < -DEMO_MACHINE_LENGTH/2)
    return demo_TOO_FAR_AWAY;
  }
  return demo ok;
}
```

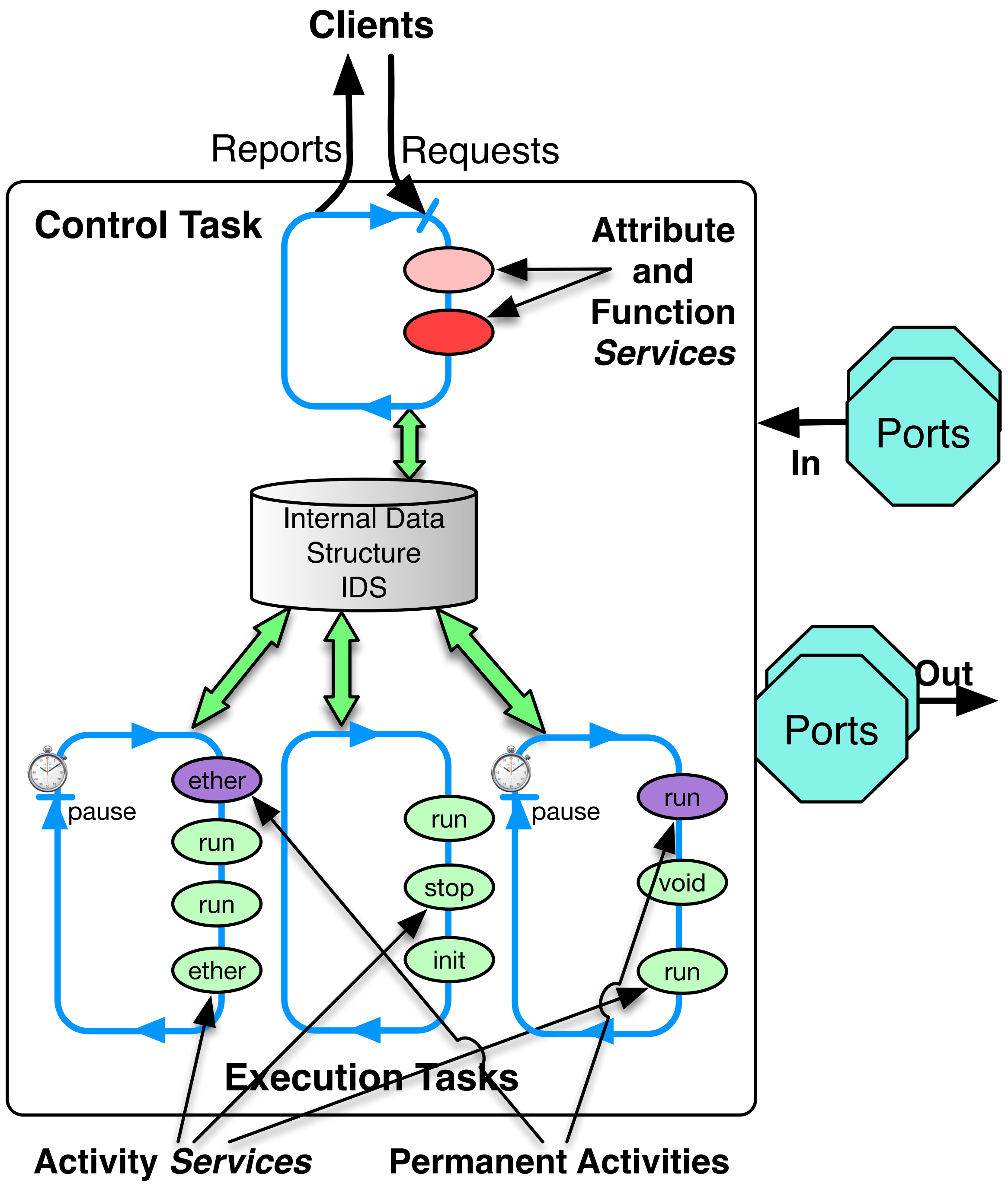
```
/* --- Activity Monitor ------------------------------------------------- */

/** Codel monitor of activity Monitor.
 *
 * Triggered by start.
 * Yields to start, stop.
 * Throws TOO FAR AWAY.
 */
demo_event
monitor(const double *monitor, const demo_ids *ids)
{
  double dDist;

  dDist = ids->state.speed * demo task period * demo millisecond;
  if (fabs (*monitor - ids->state.position) < dDist) {
    printf ("dist %f mon %f pos %f\n", dDist, *monitor, ids->state.position);
    return demo_stop;
  }
  return demo_start;
}

/** Codel monitorStop of activity Monitor.
 *
 * Triggered by stop.
 * Yields to ether.
 * Throws TOO_FAR_AWAY.
 */
demo_event
monitorStop(const demo_ids *ids, double *position)
{
  *position = ids->state.position;
  return demo ether;
}
```

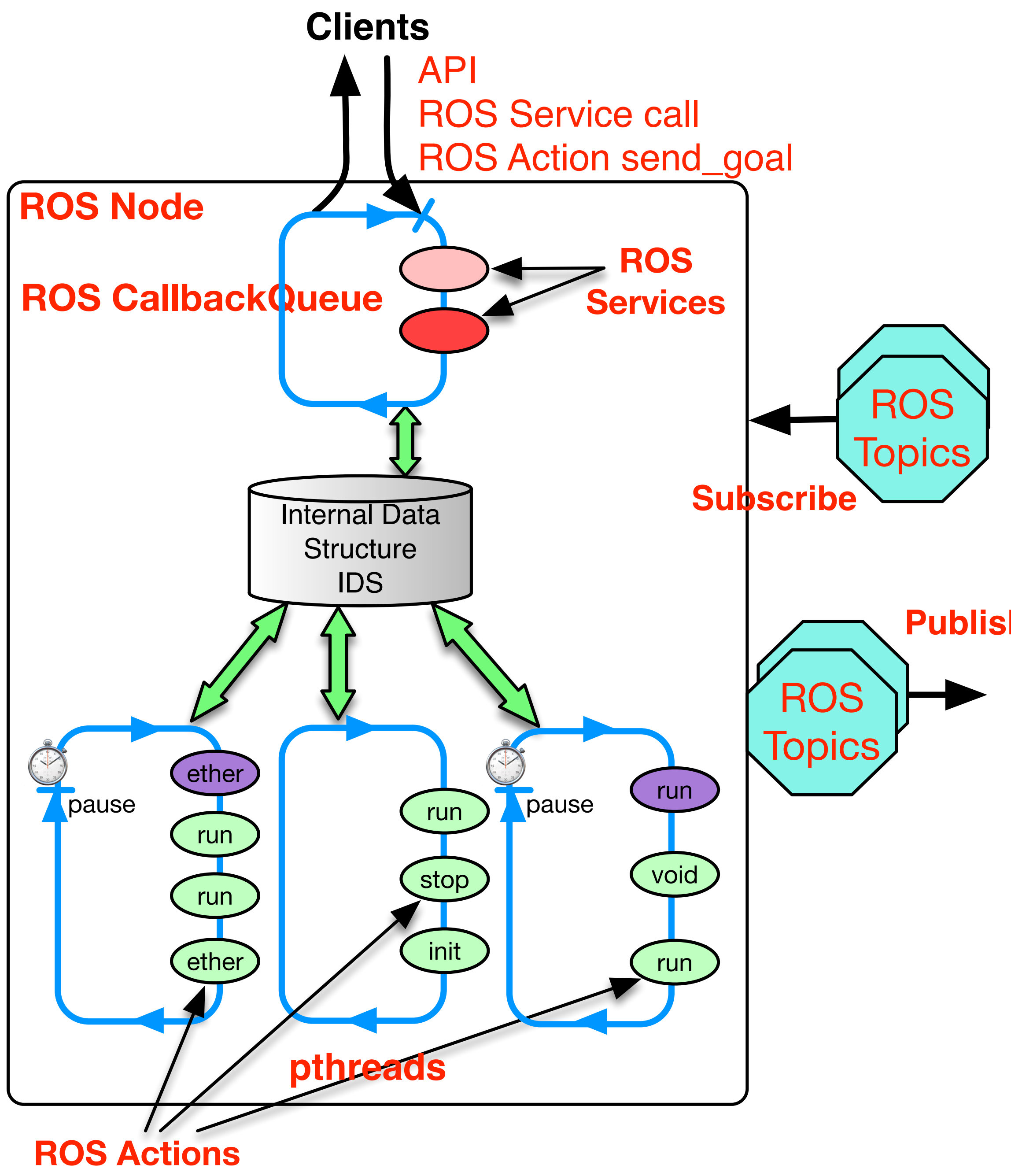
Template ROS and pocolibs


Component Codels
C
Compiler
template pocolibs
TCL/C
Component Specification
.gen
GenoM
template ros-comm
TCL/C++
Component pocolibs Server Source
C
Codels Lib 0100 0011 1001
Component ROS Server Source
C++
pocolibs Lib 0100 0011 1001
Compiler
Linker
pocolibs exec Component 0100 0011 1001
Compiler
Linker
ROS Lib 0100 0011 1001
ROS exec Component 0100 0011 1001



ROS implementation





+ internal algorithms written in C++

How is GenoM helping us for V&V?

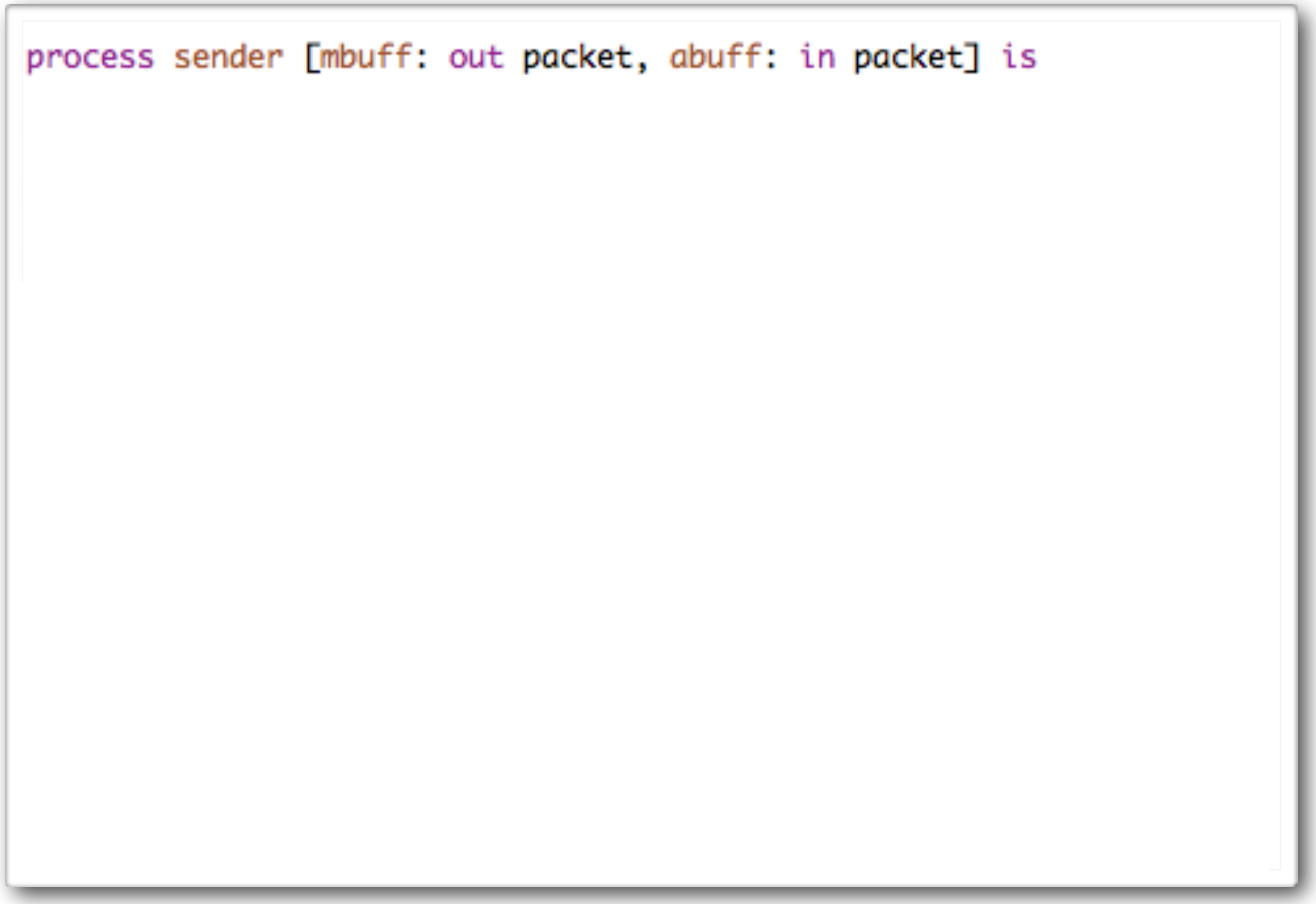
- ➡ After all, we just have to provide some “painful” specifications…
- ➡ where is the added value? how is this helping us?

- ☑ Codel granularity (better parallelism specification)
- ☑ Data access (and sharing) is fully specified (ports, IDS, and nothing else)
- ☑ Automata specification provides execution sequence and time/period management
- ☑ Tasks are clearly specified (how many, periodic, sporadic)
- ☑ Template mechanism…

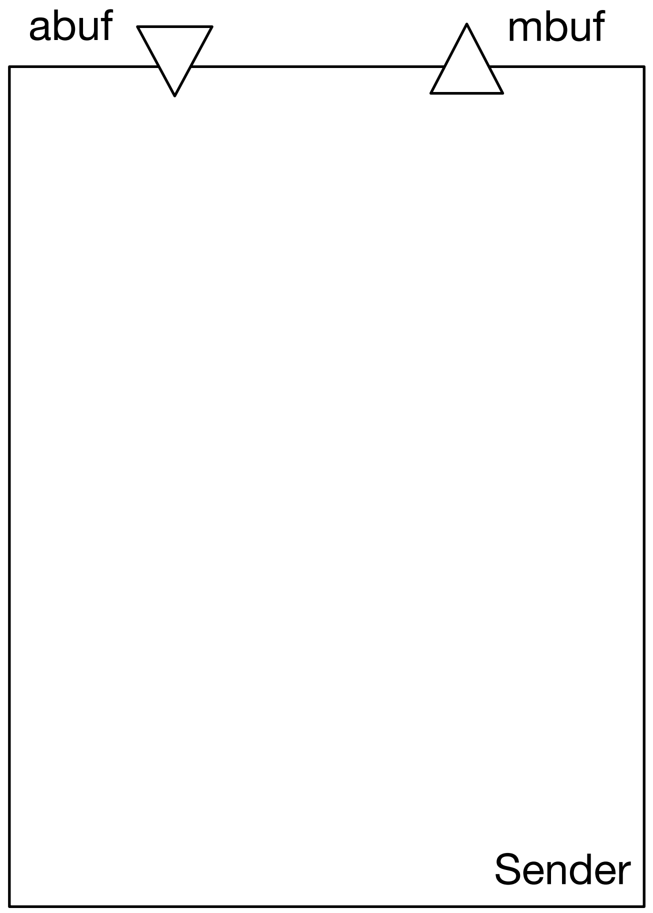
Verification & Validation with Fiacre

- Formally defined (semantics, compositional)
  - Types
    - Rich set of primitive data types; Strongly typed
- Processes
  - Parameterized labelled automata
  - Symbolic state transitions; high-level commands
  - Ports for communication and/or synchronization
  - May share variables with other processes
- Components
  - Hierarchically defined
  - Specify interactions between sub-components
  - Constrain interactions (time, priorities)
  - Control scope of shared variables

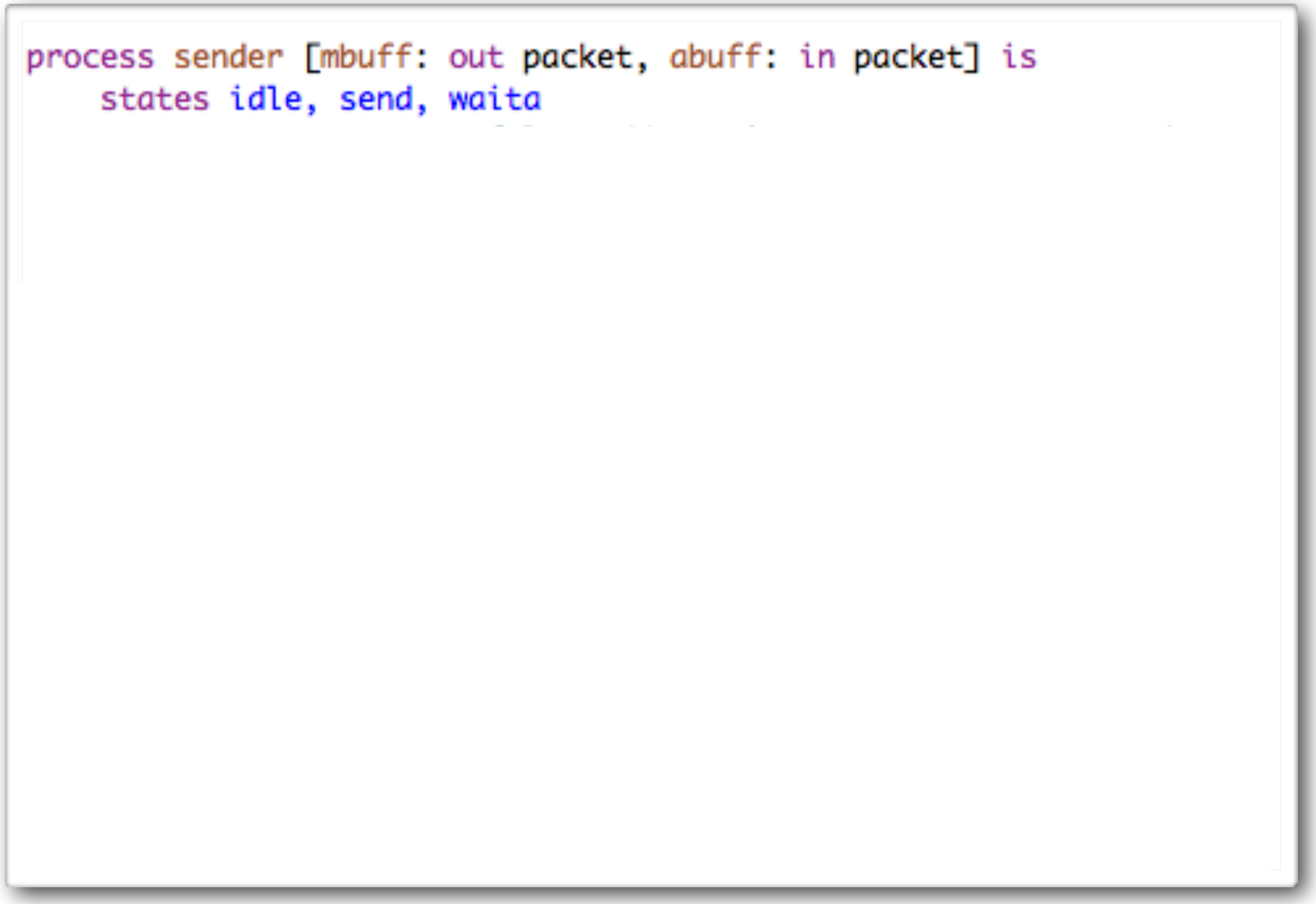
Fiacre process example



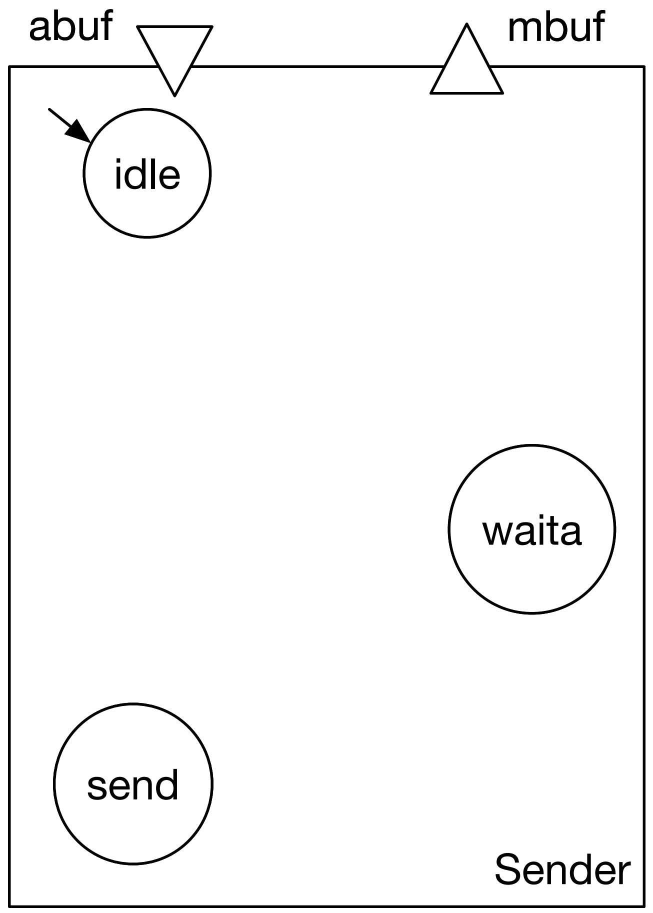
```
process sender [mbuff: out packet, abuff: in packet] is
```



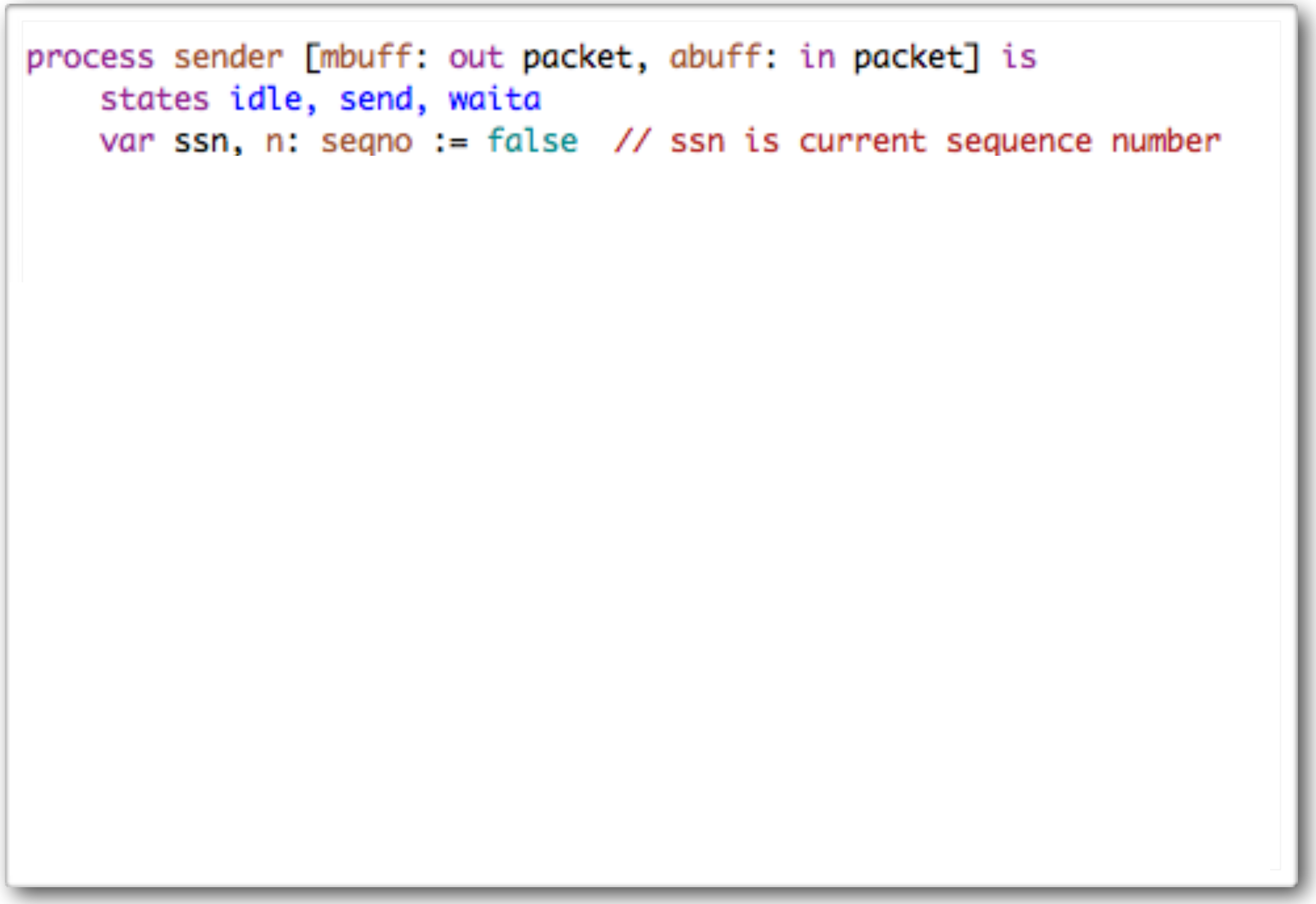
Fiacre process example



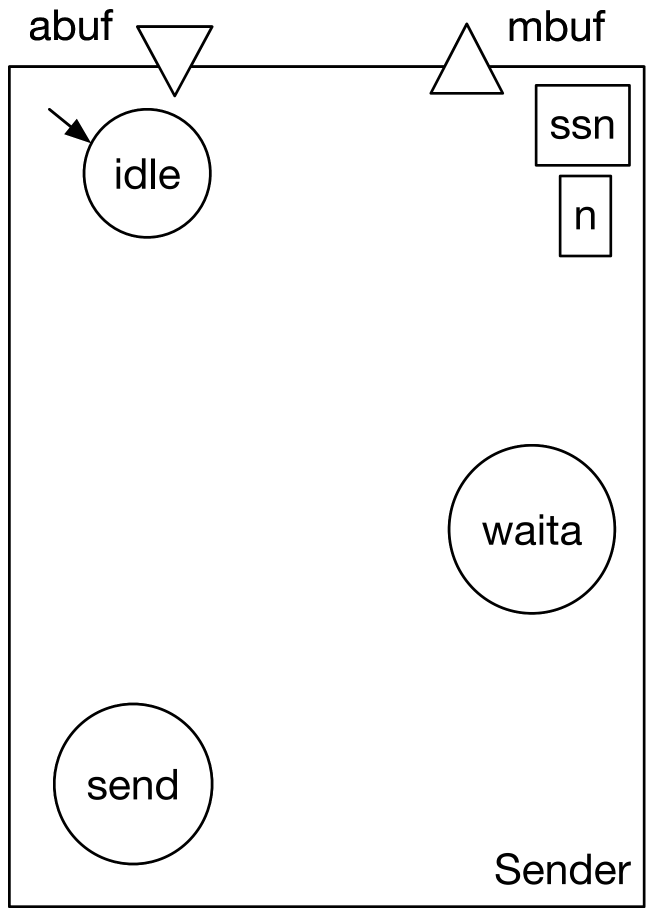
```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
```



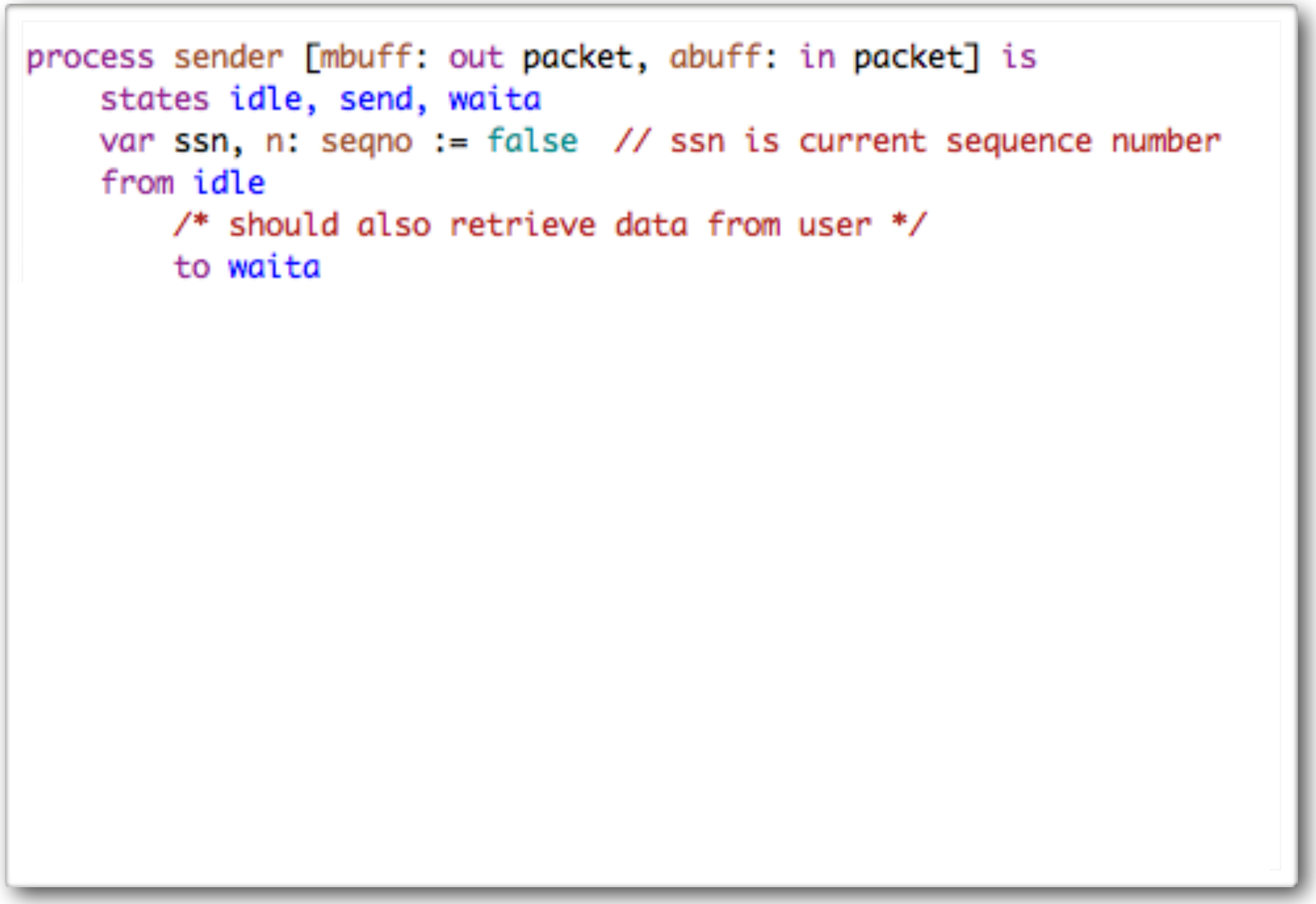
Fiacre process example



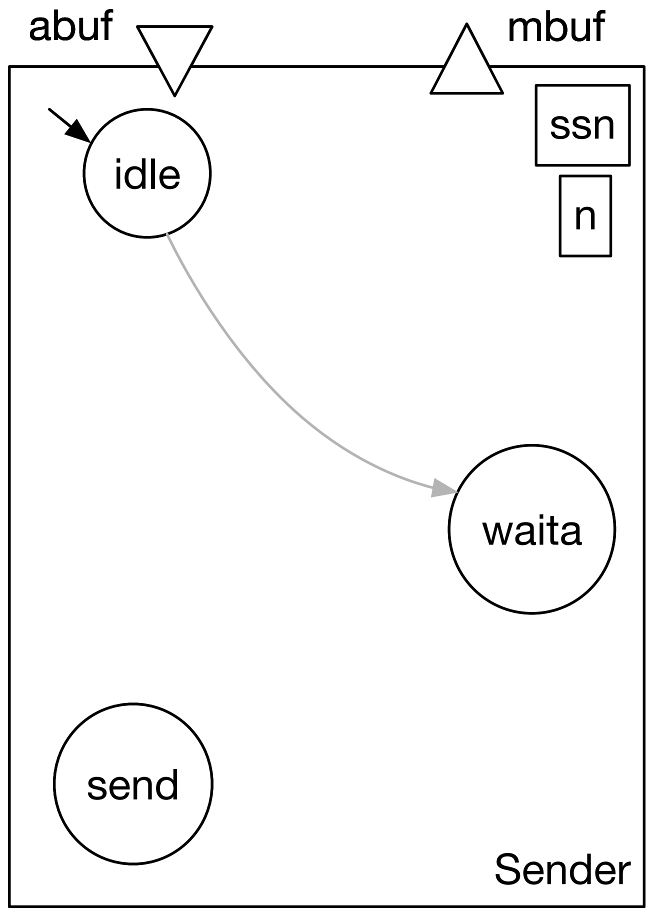
```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
```



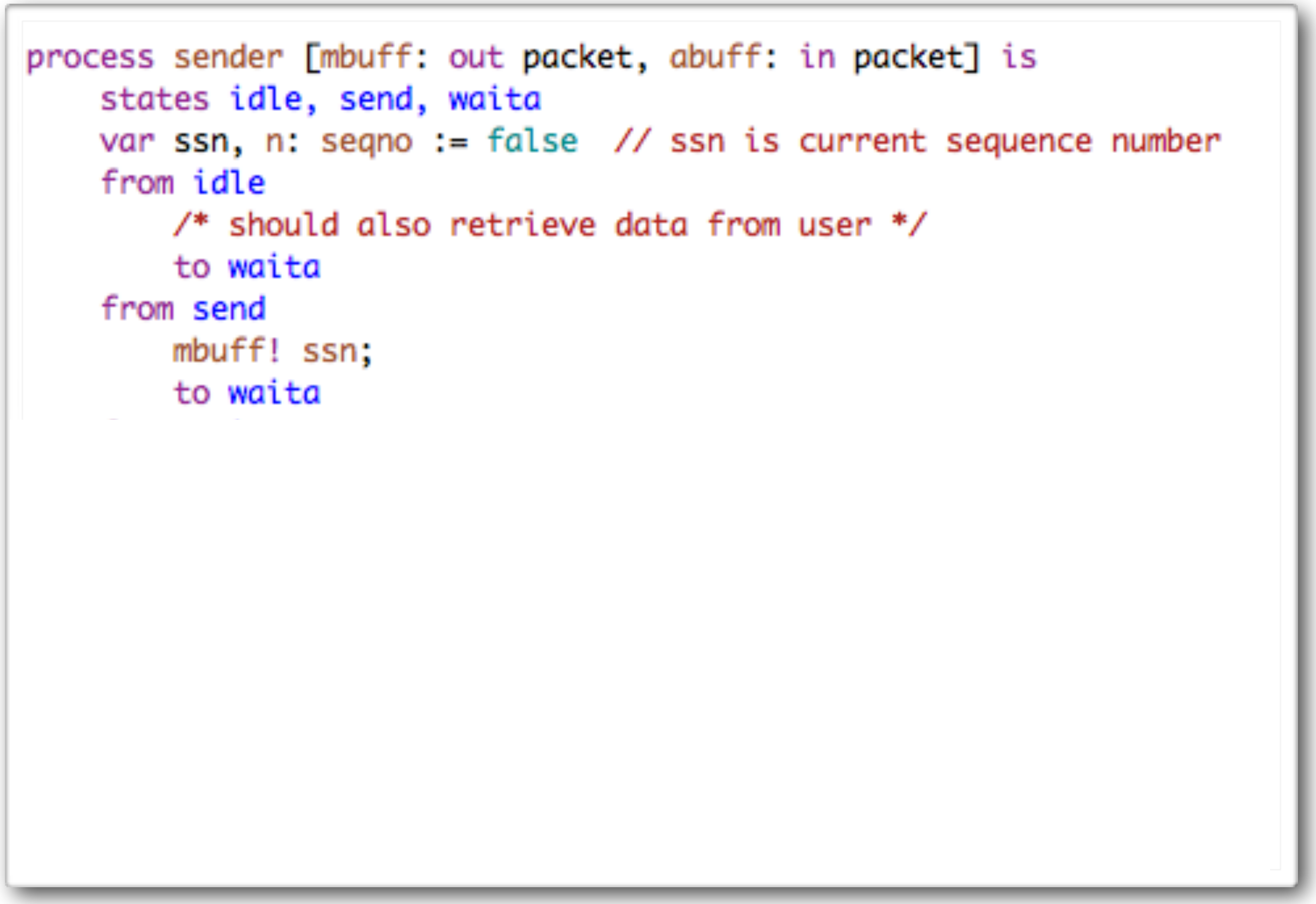
Fiacre process example



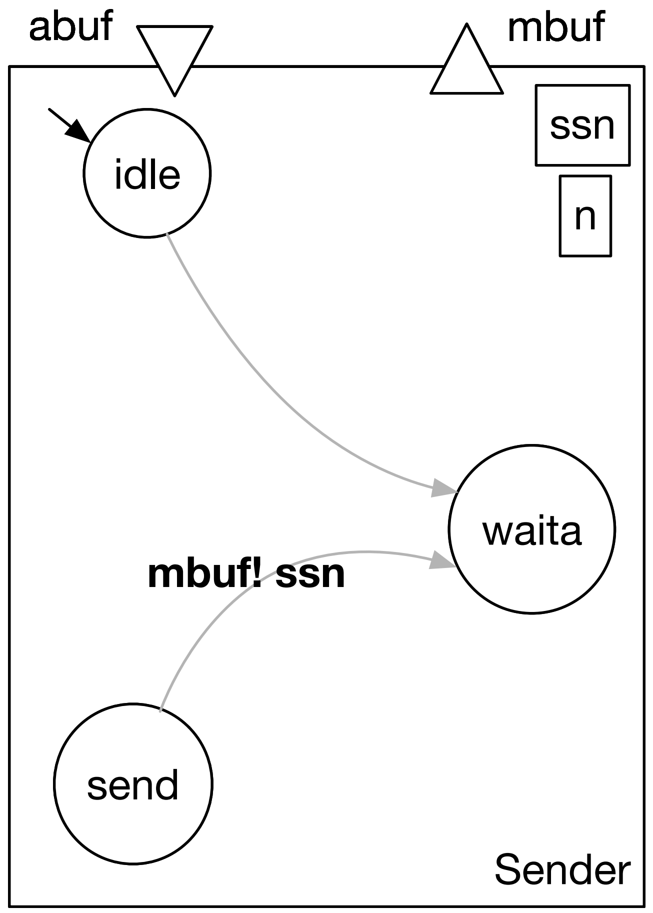
```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
```



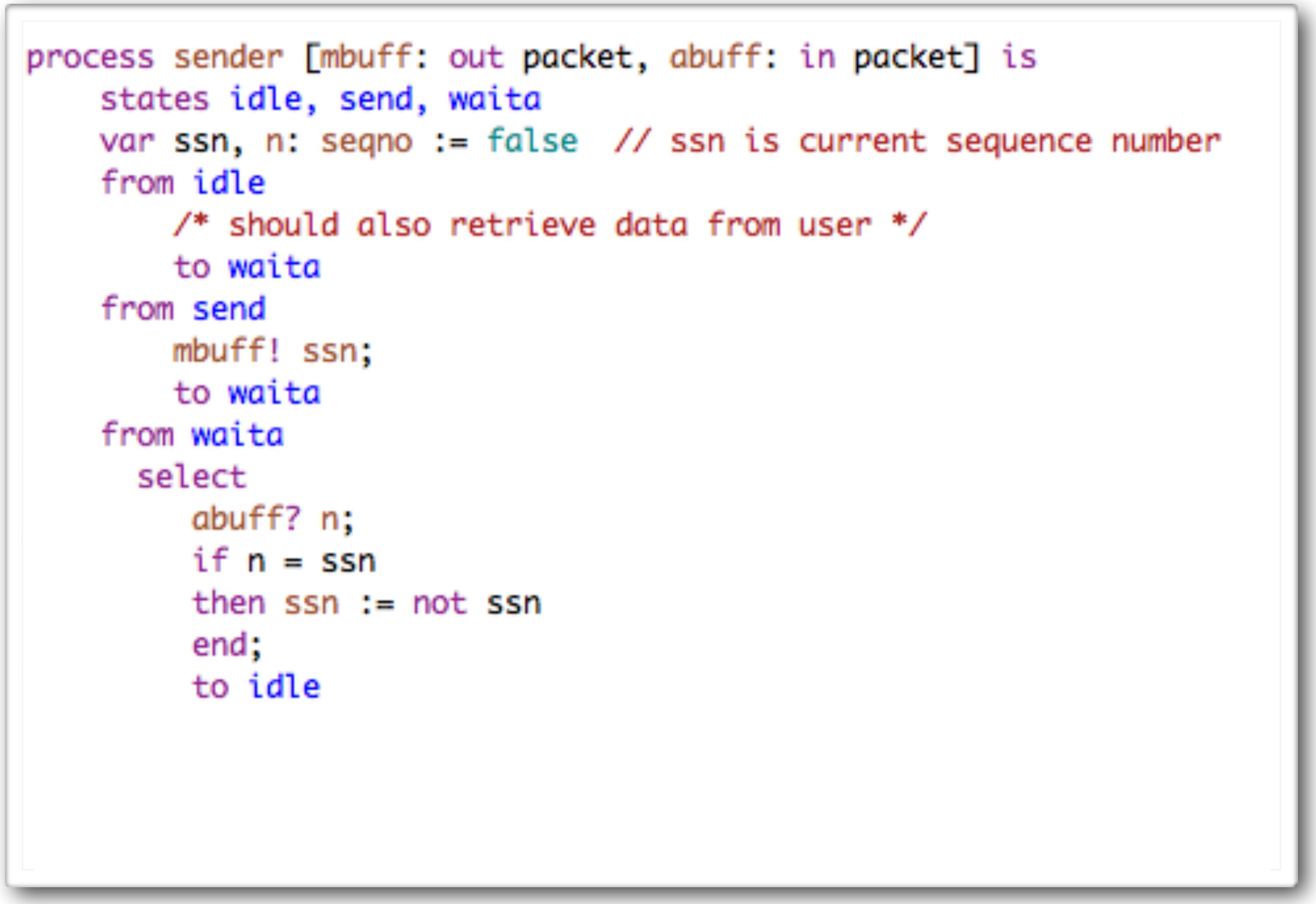
Fiacre process example



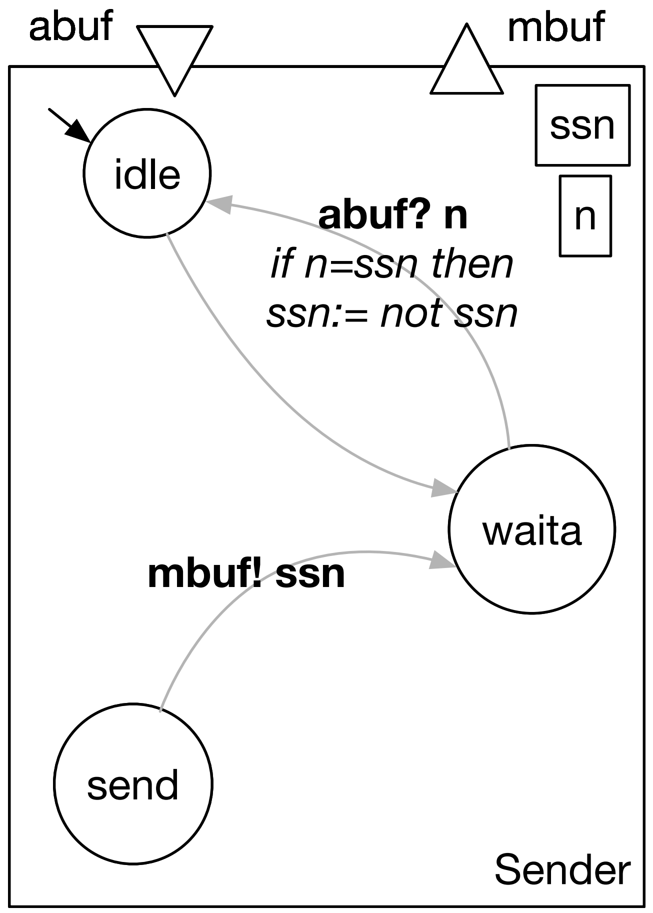
```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
```



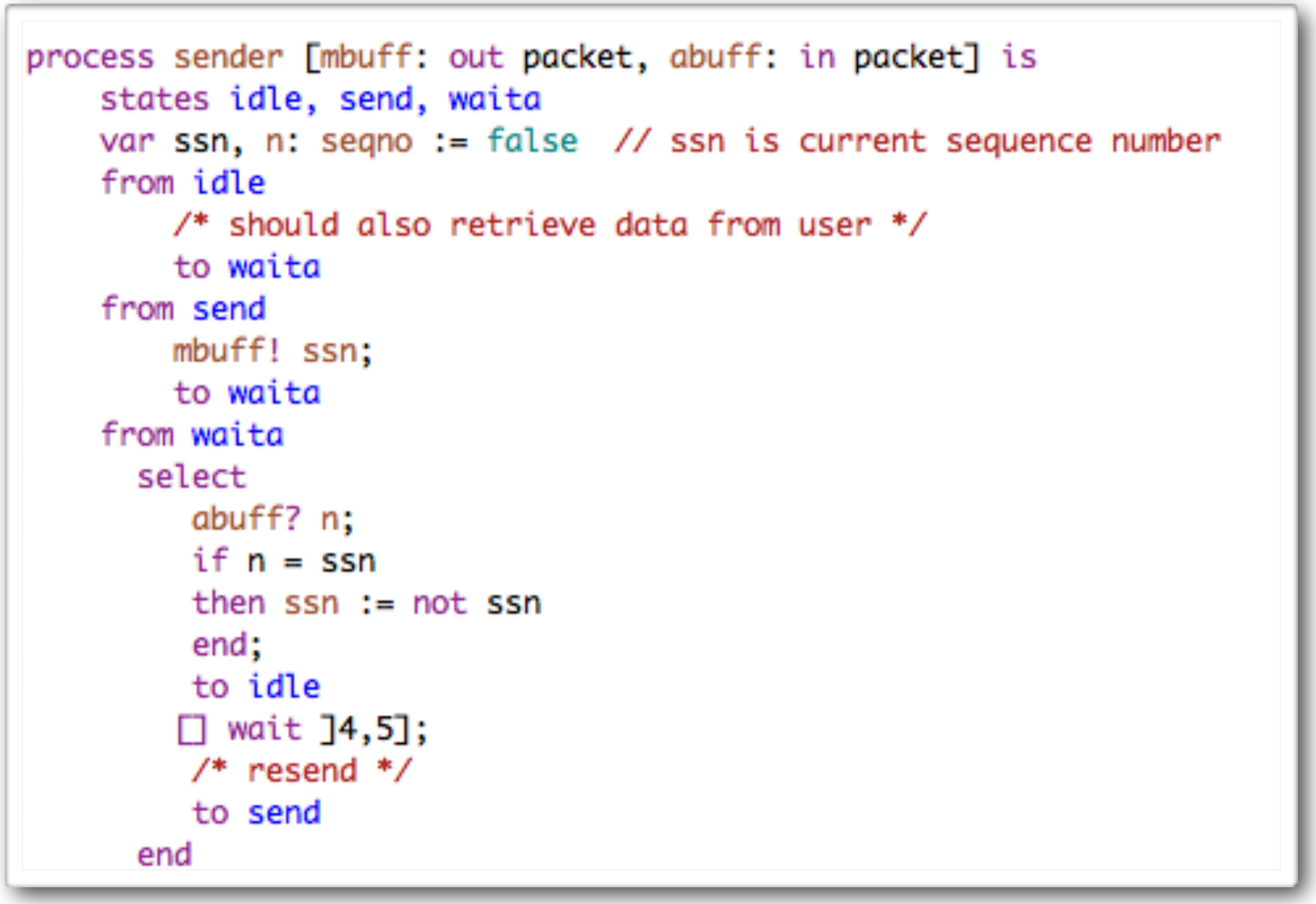
Fiacre process example



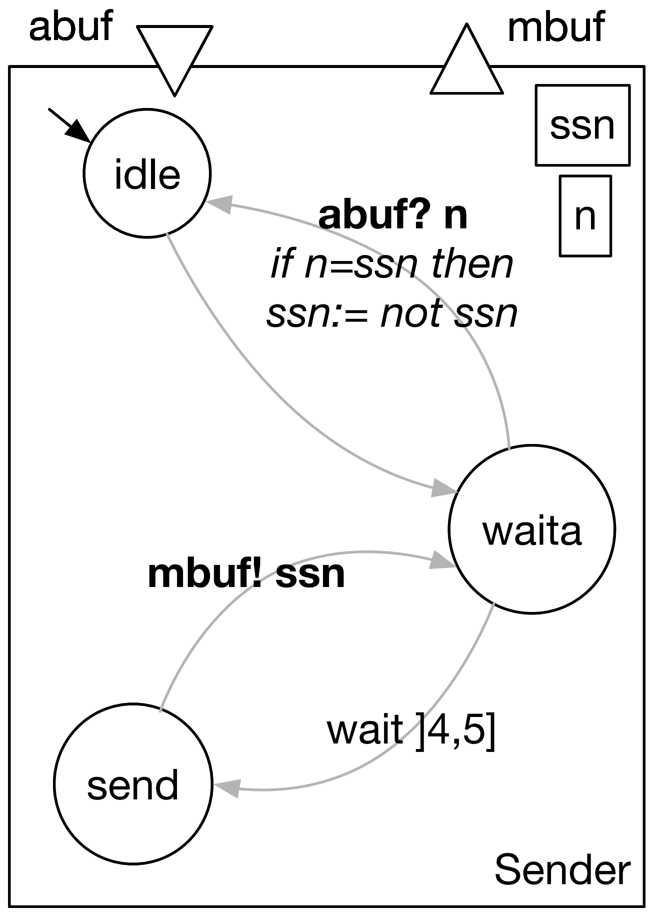
```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
```



Fiacre process example

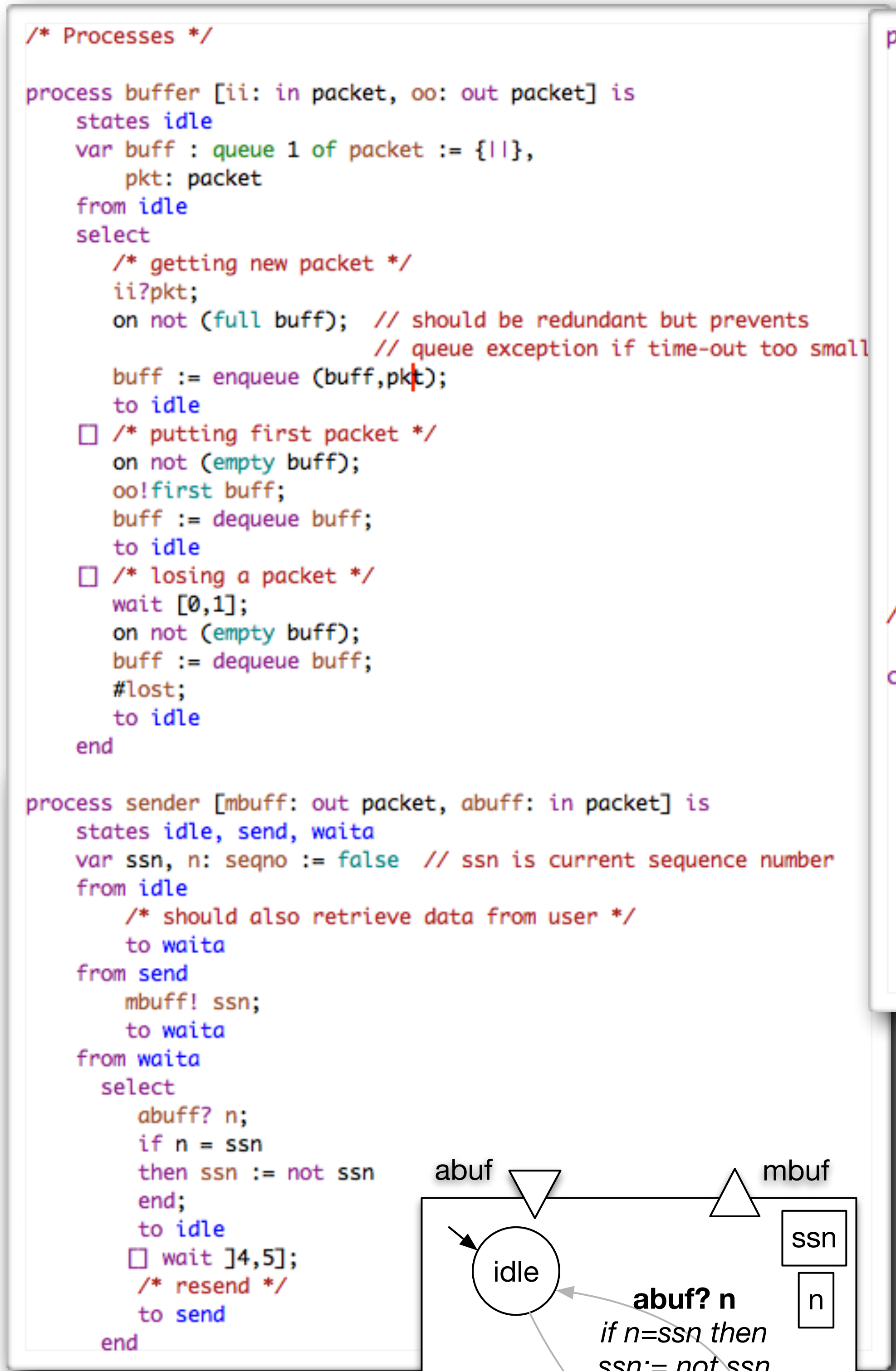


```
process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
        ☐ wait ]4,5];
         /* resend */
         to send
      end
```

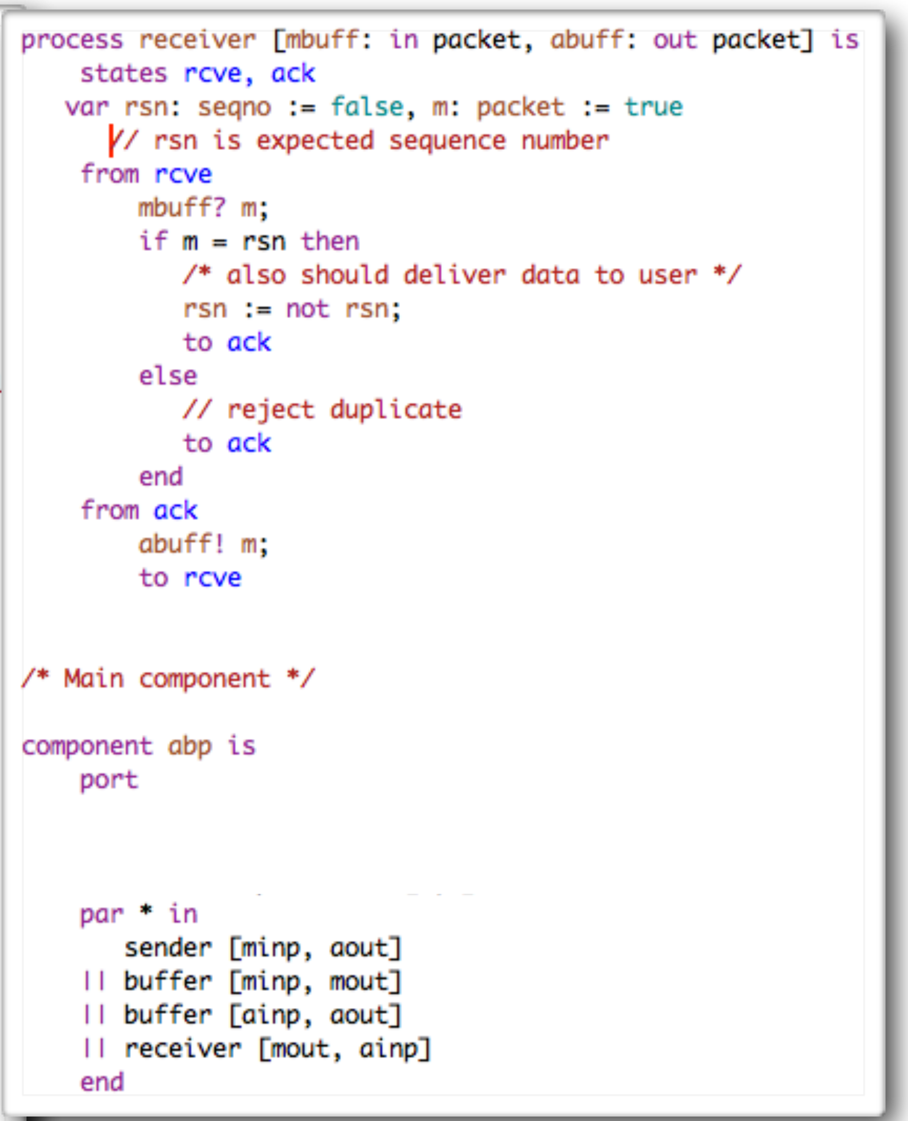


Fiacre component example

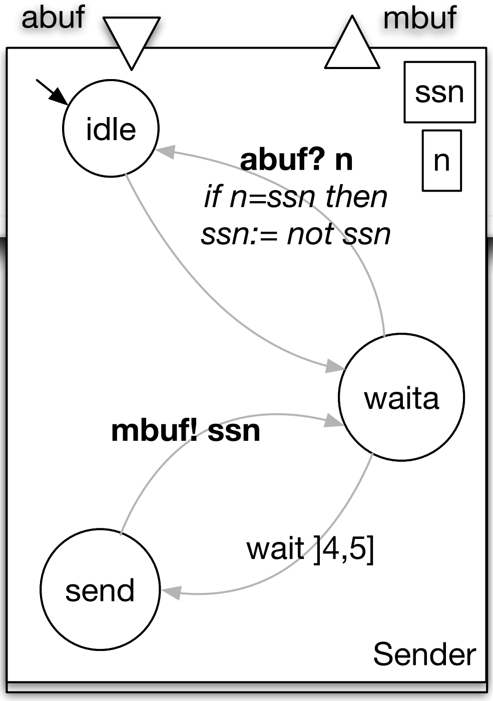
Alternating Bit Protocol

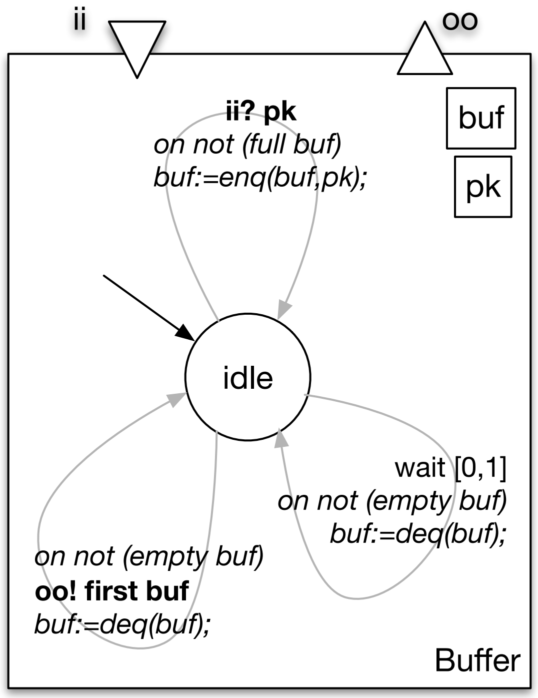


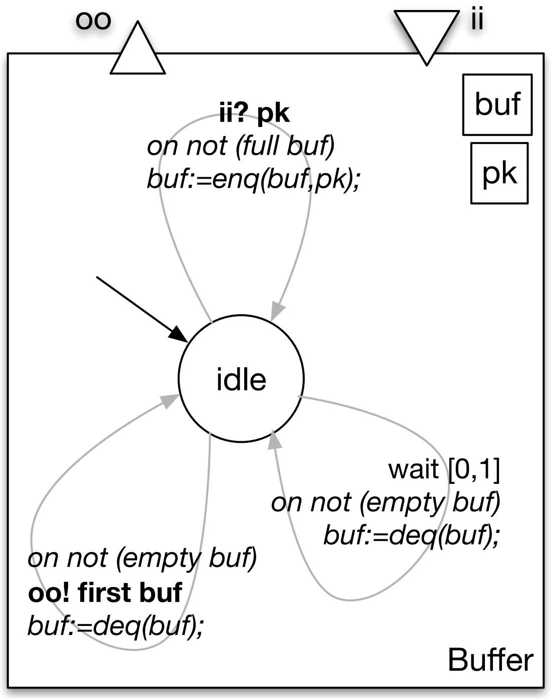
```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    [] /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    [] /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
       [] wait ]4,5];
         /* resend */
         to send
      end
```

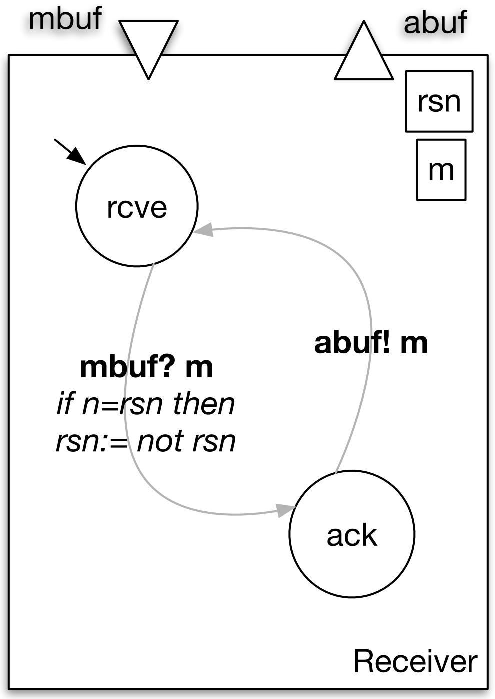


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port



    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```



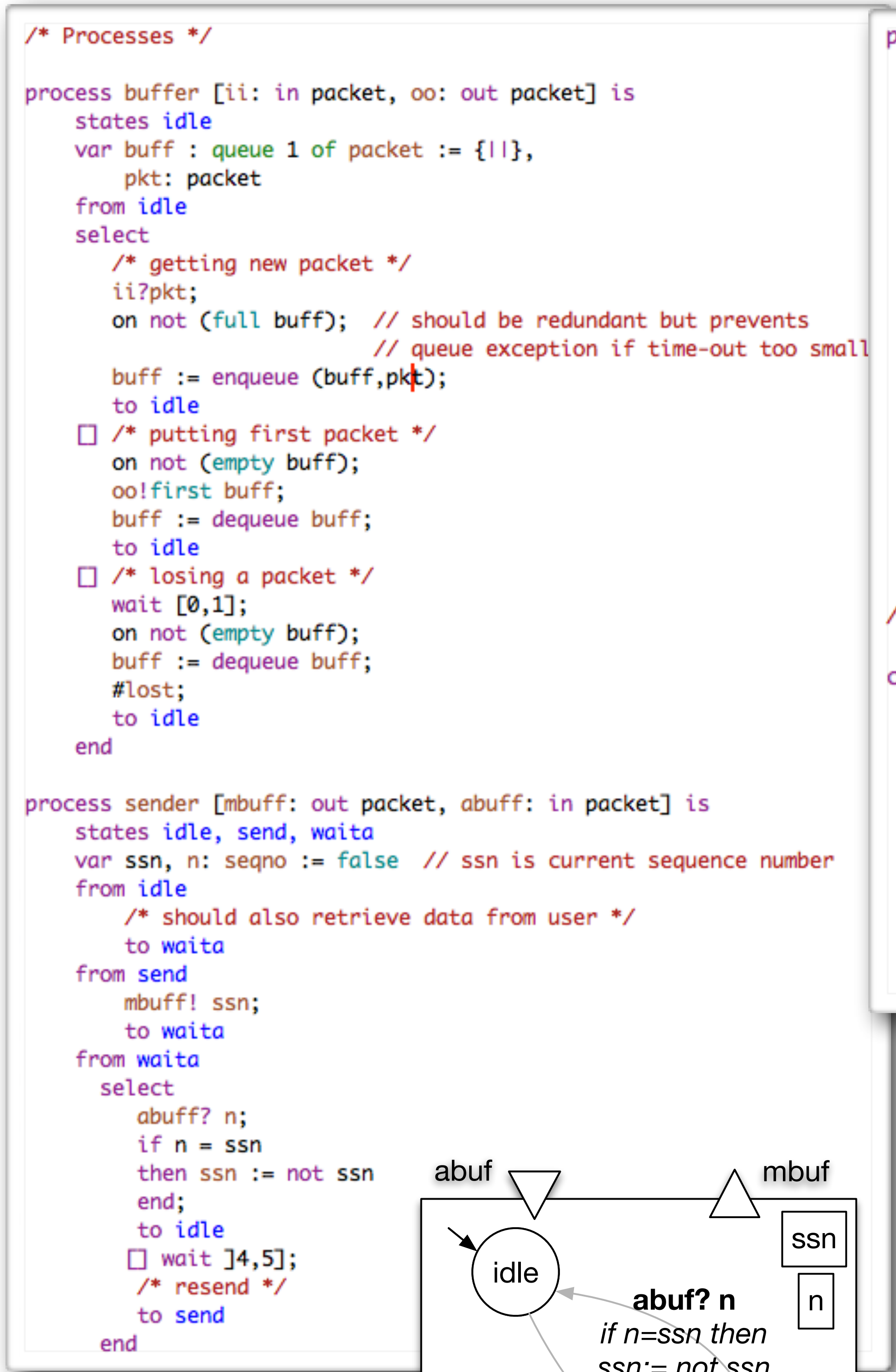




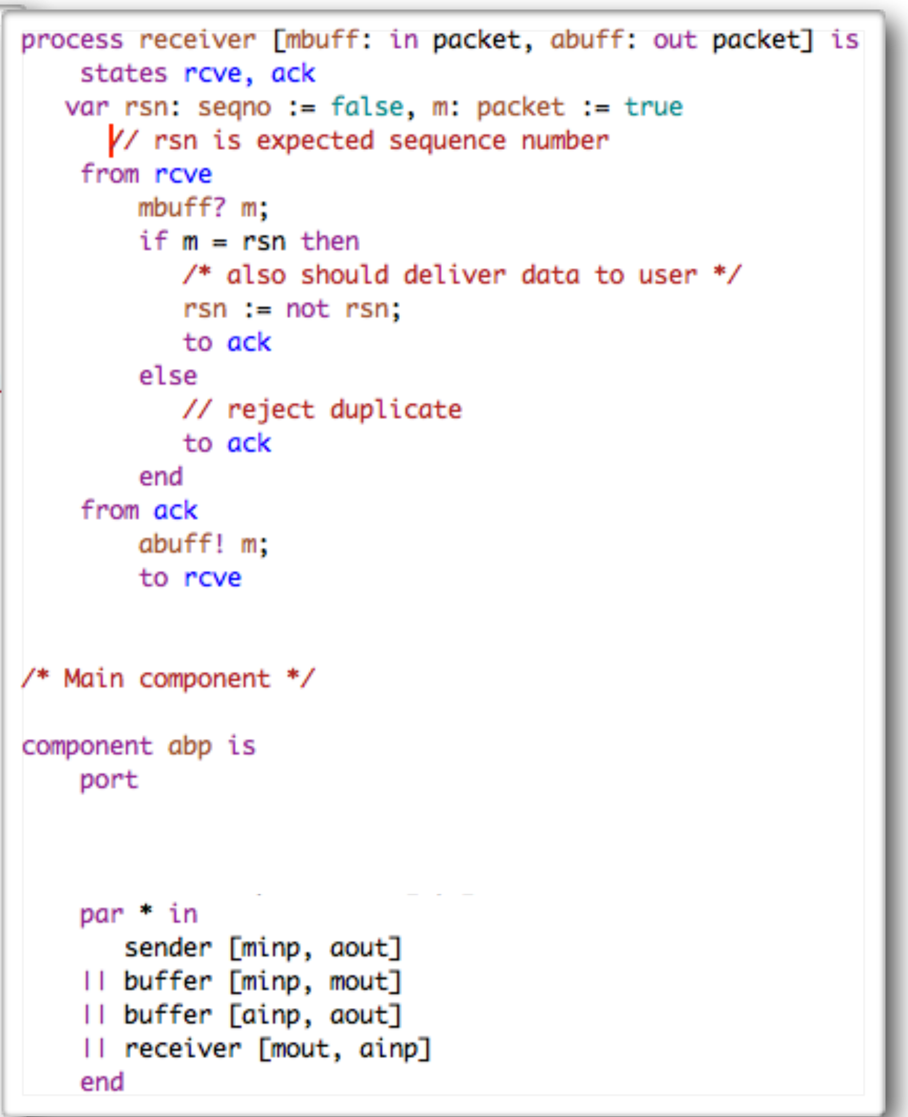


Fiacre component example

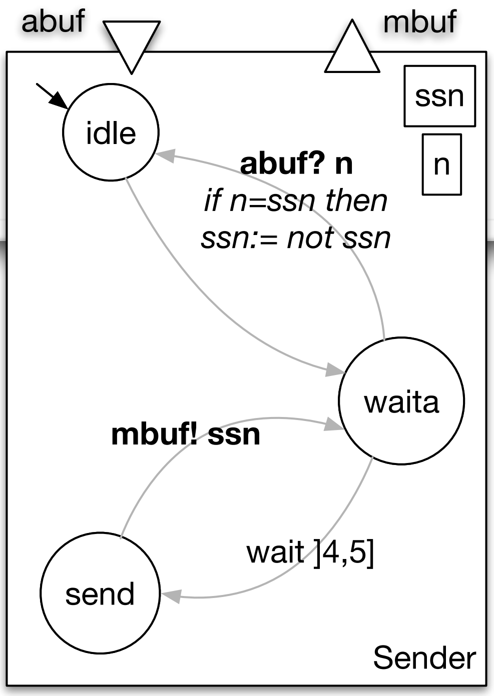
Alternating Bit Protocol

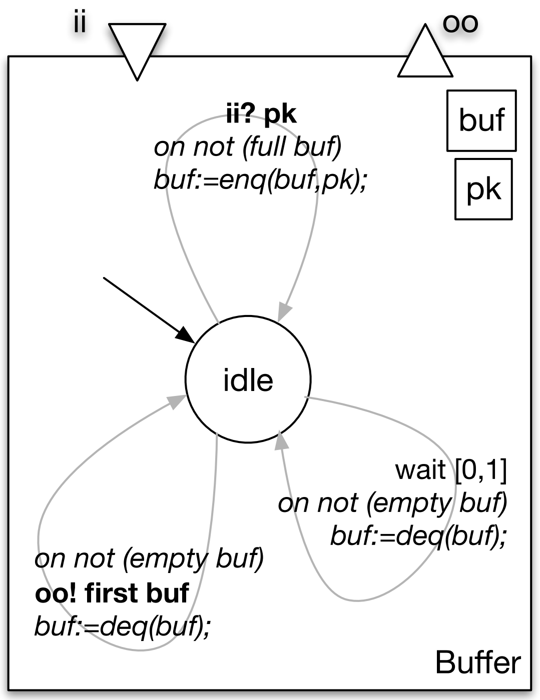


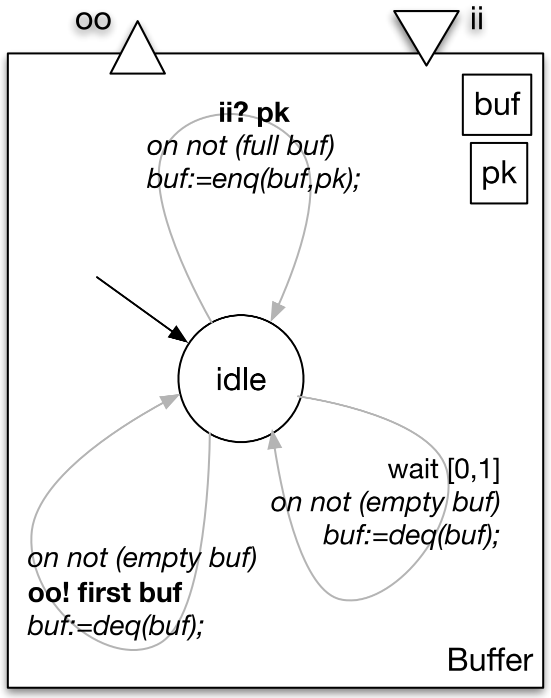
```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    □  /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    □  /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
       □ wait ]4,5];
         /* resend */
         to send
      end
```

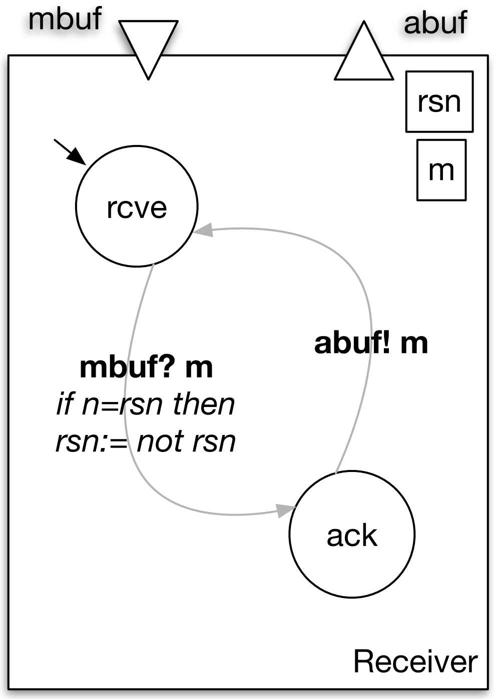


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port



    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```



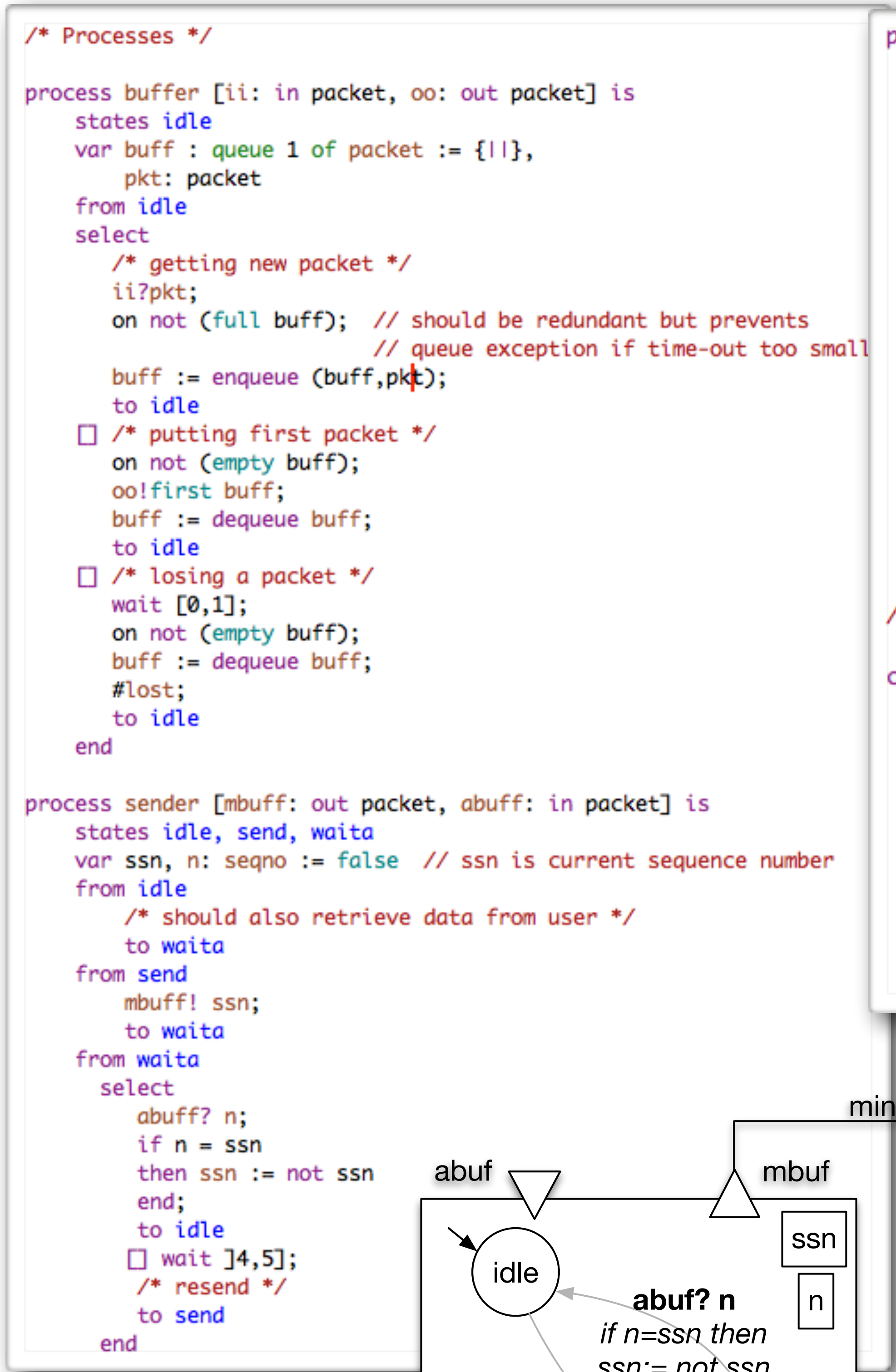




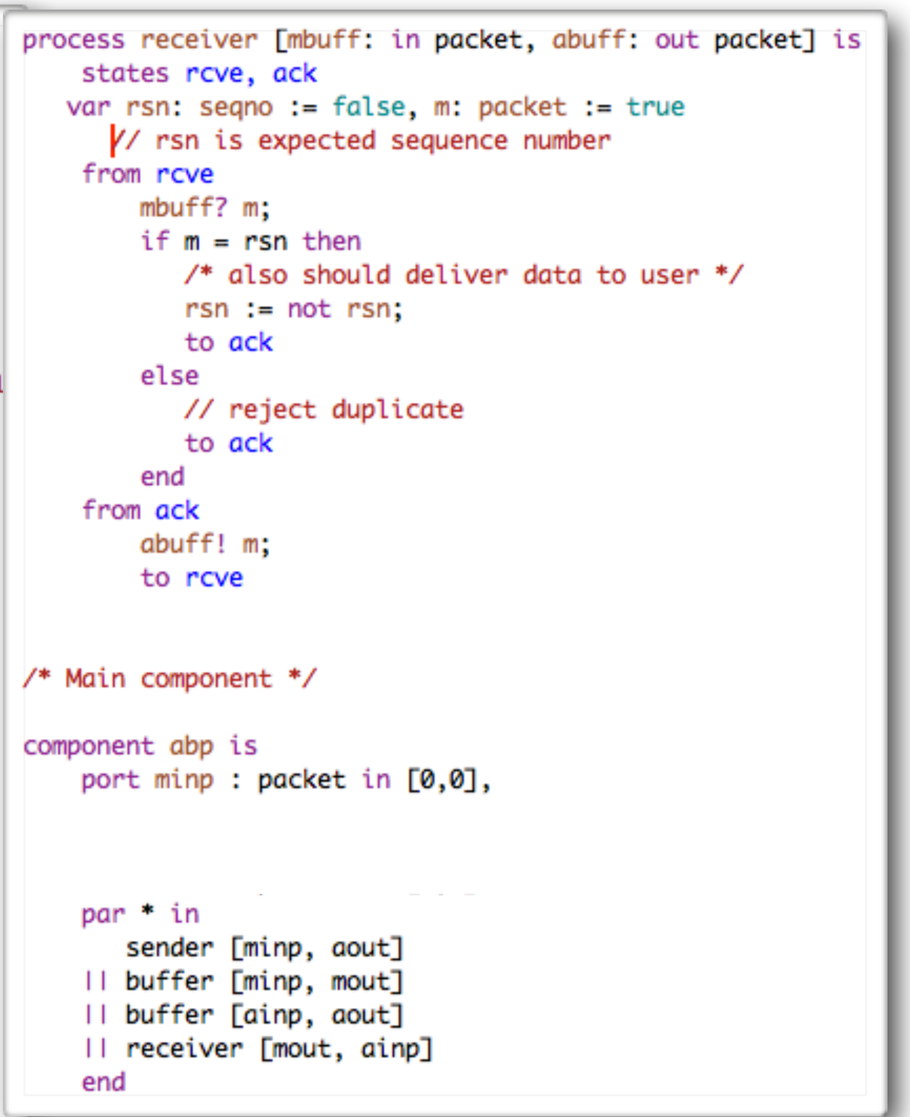


Fiacre component example

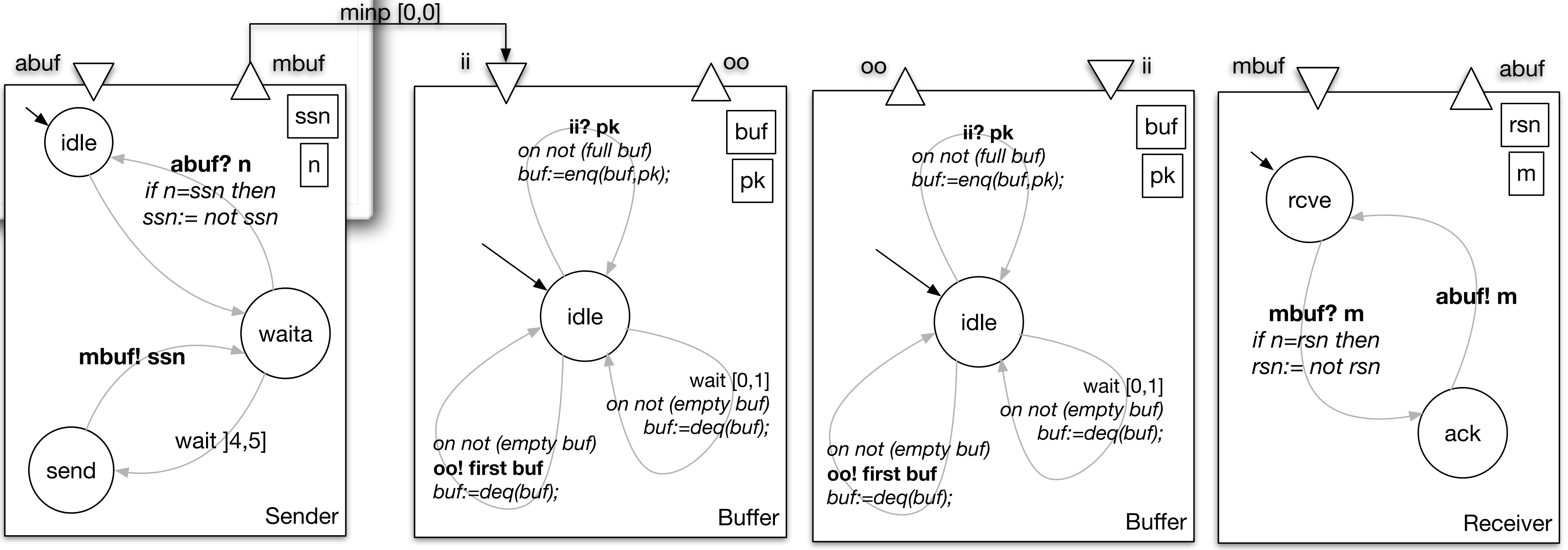
Alternating Bit Protocol



```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    [] /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    [] /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
      [] wait ]4,5];
         /* resend */
         to send
      end
```

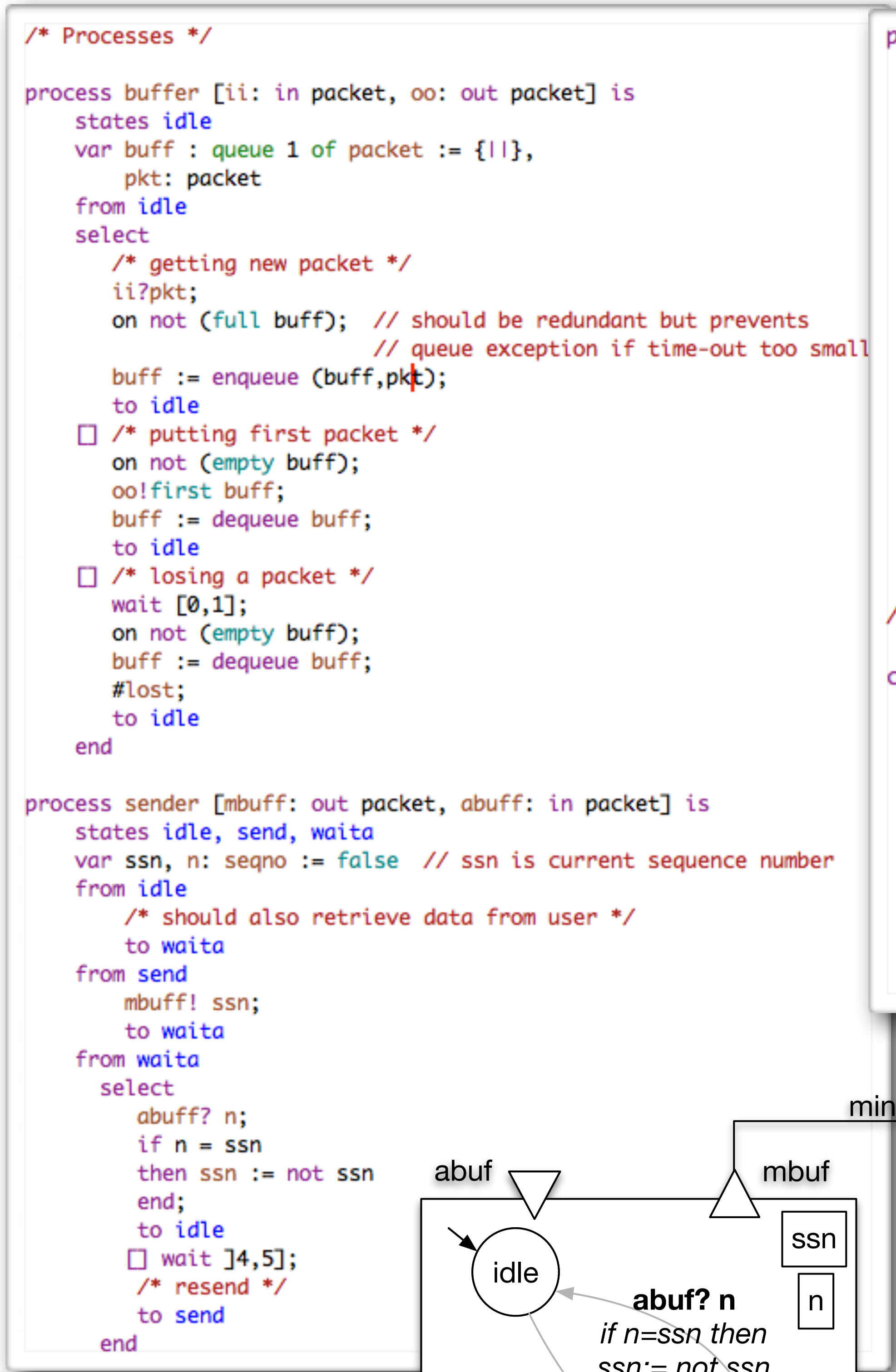


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port minp : packet in [0,0],



    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```

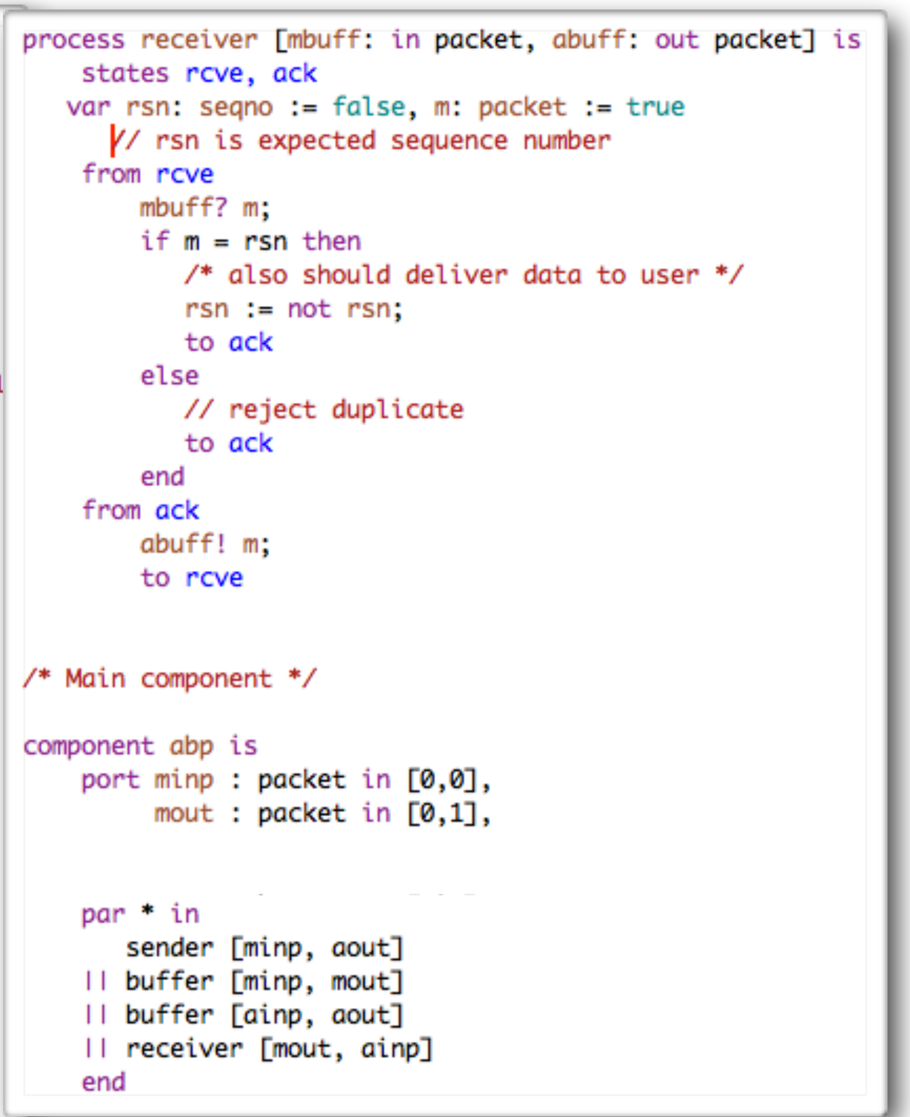


Fiacre component example

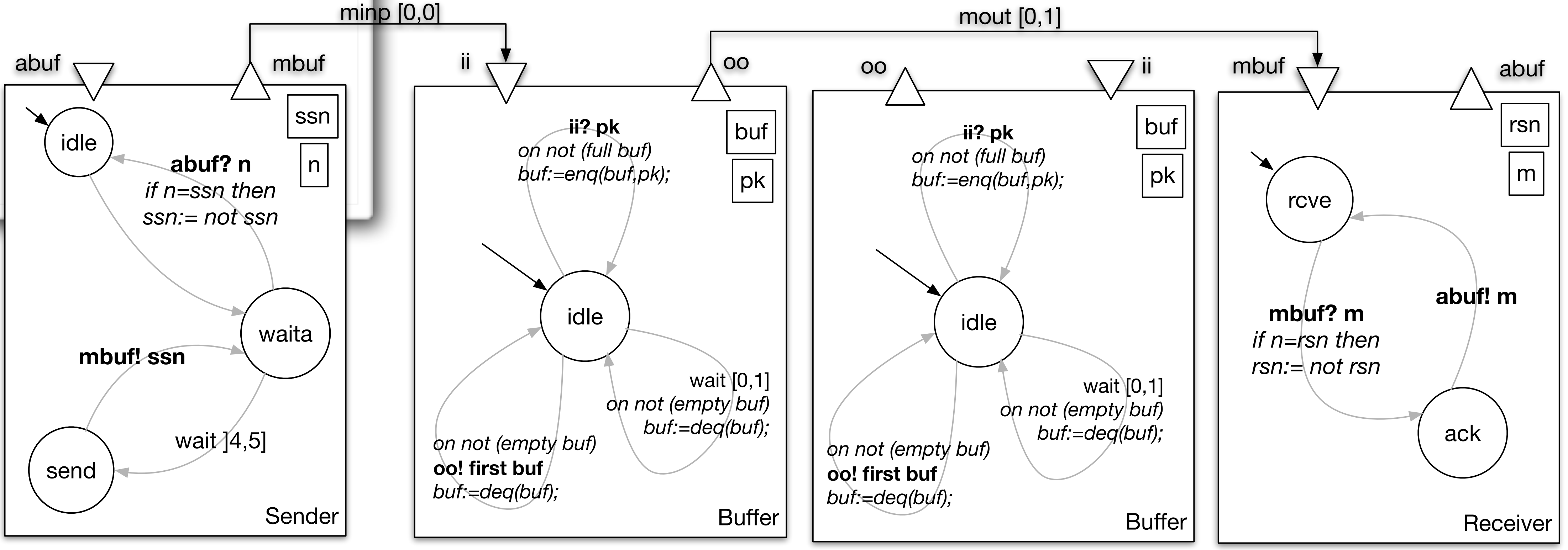
Alternating Bit Protocol



```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    [] /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    [] /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
       [] wait ]4,5];
         /* resend */
         to send
      end
```

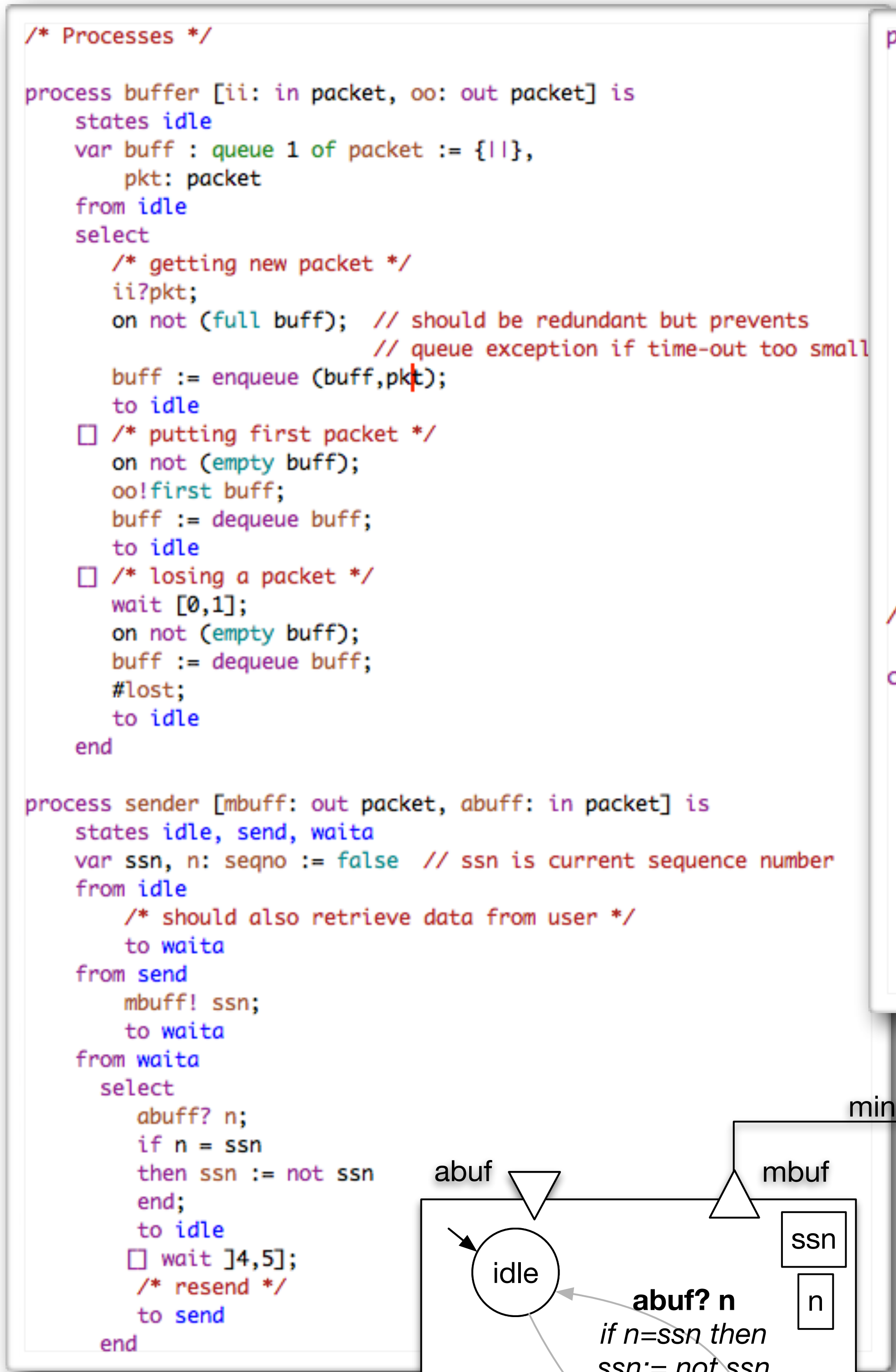


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port minp : packet in [0,0],
         mout : packet in [0,1],


    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```

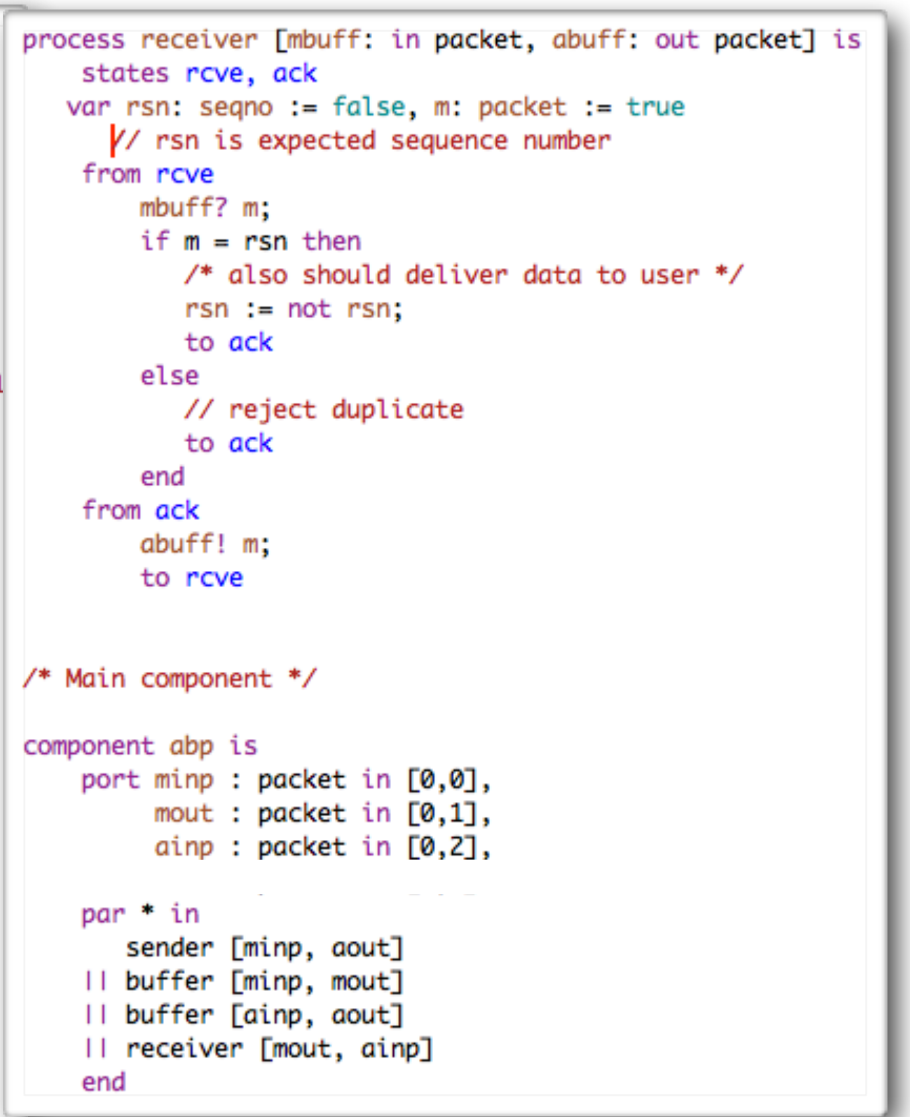


Fiacre component example

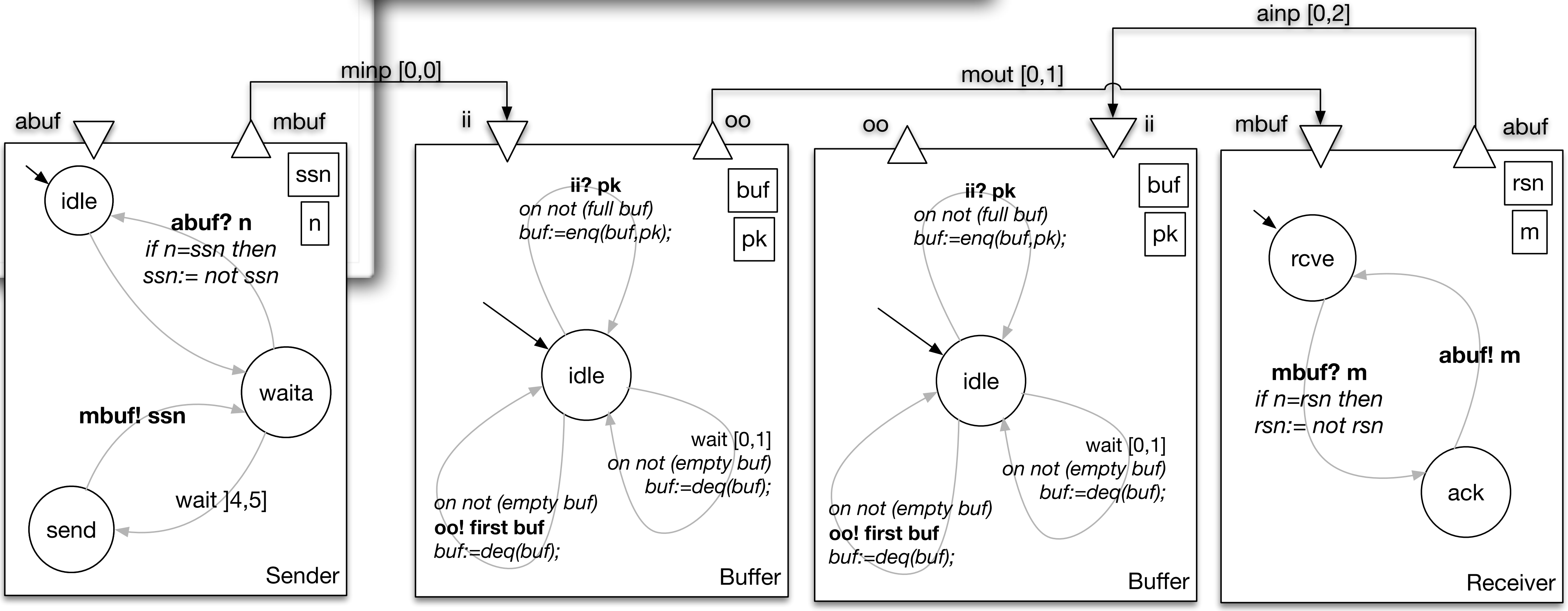
Alternating Bit Protocol



```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    ☐ /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    ☐ /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
       /* should also retrieve data from user */
       to waita
    from send
       mbuff! ssn;
       to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
      ☐ wait ]4,5];
         /* resend */
         to send
      end
```

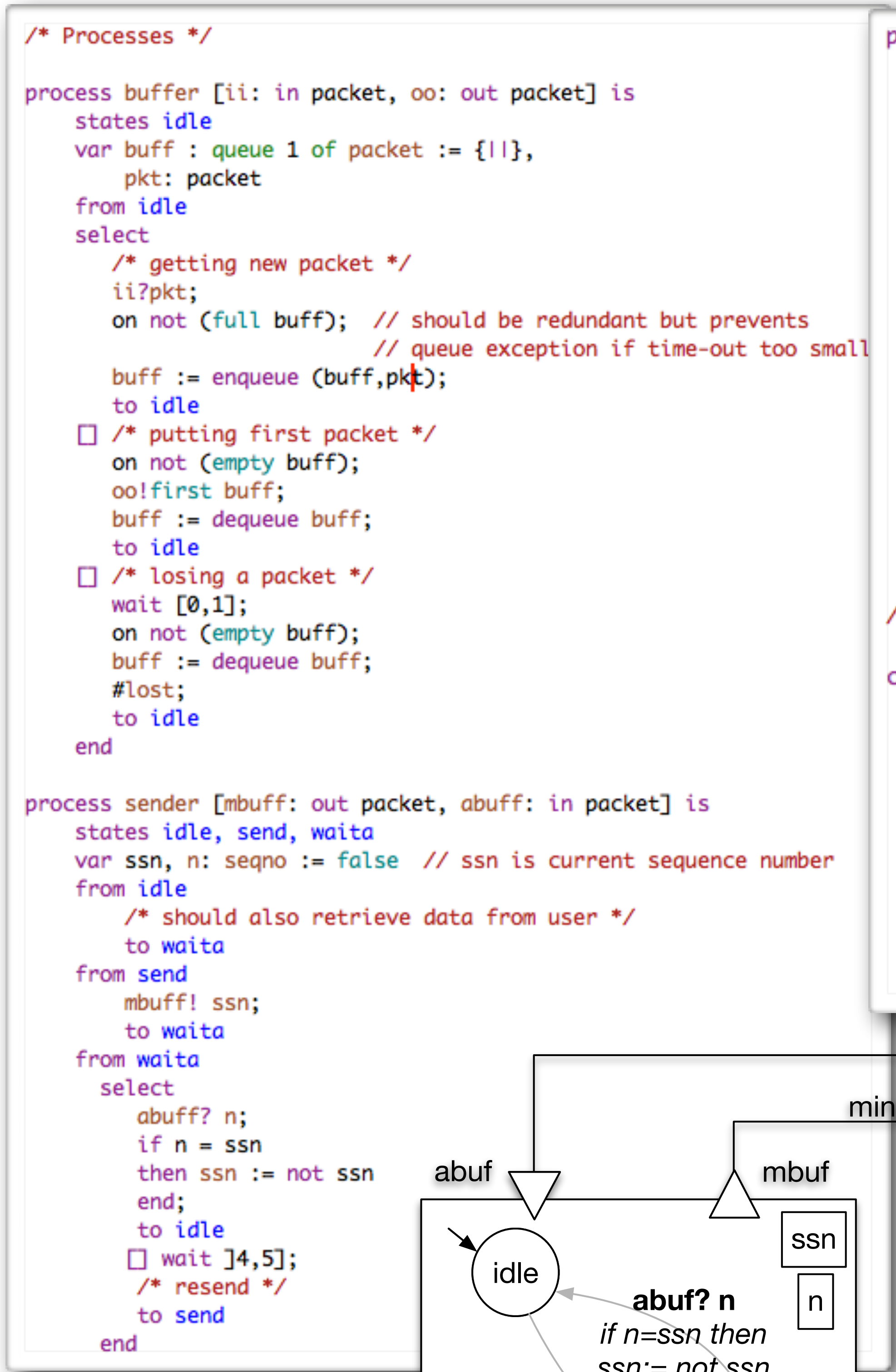


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
     // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port minp : packet in [0,0],
         mout : packet in [0,1],
         ainp : packet in [0,2],

    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```

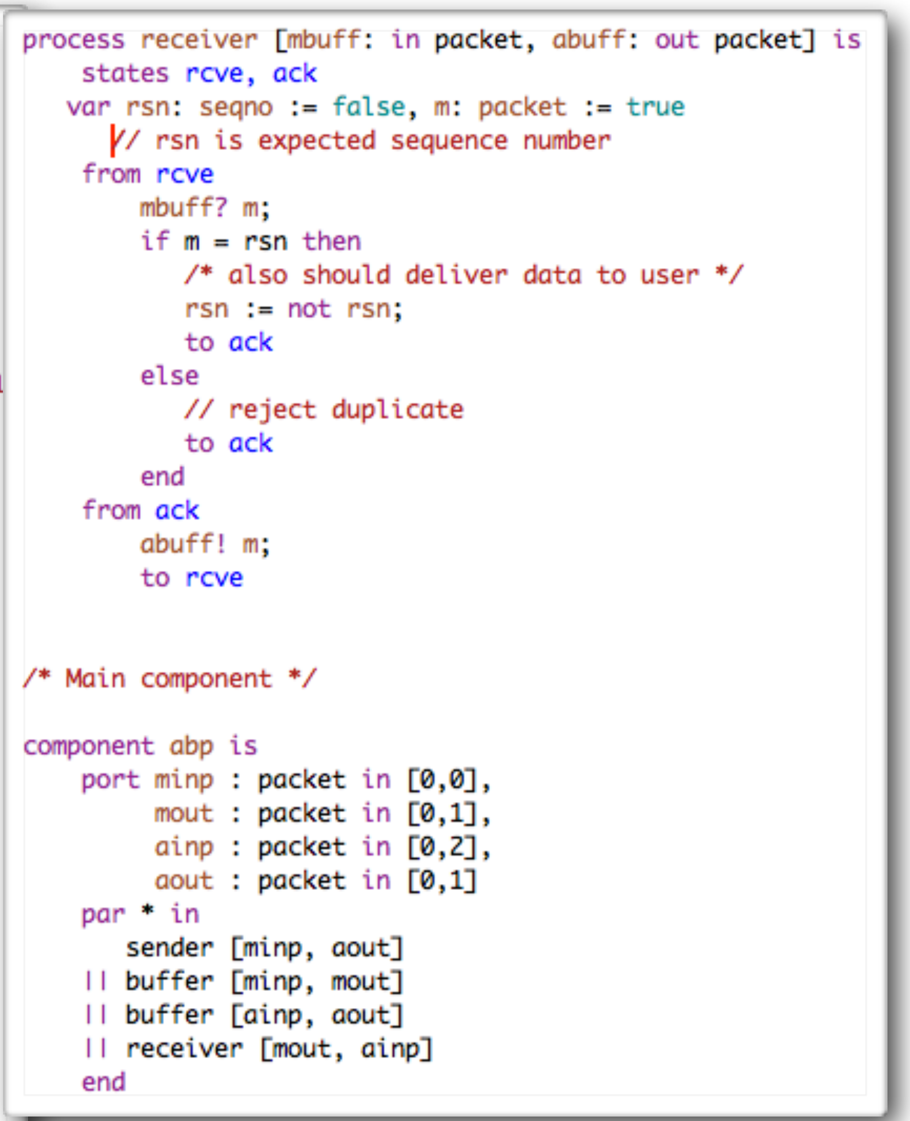


Fiacre component example

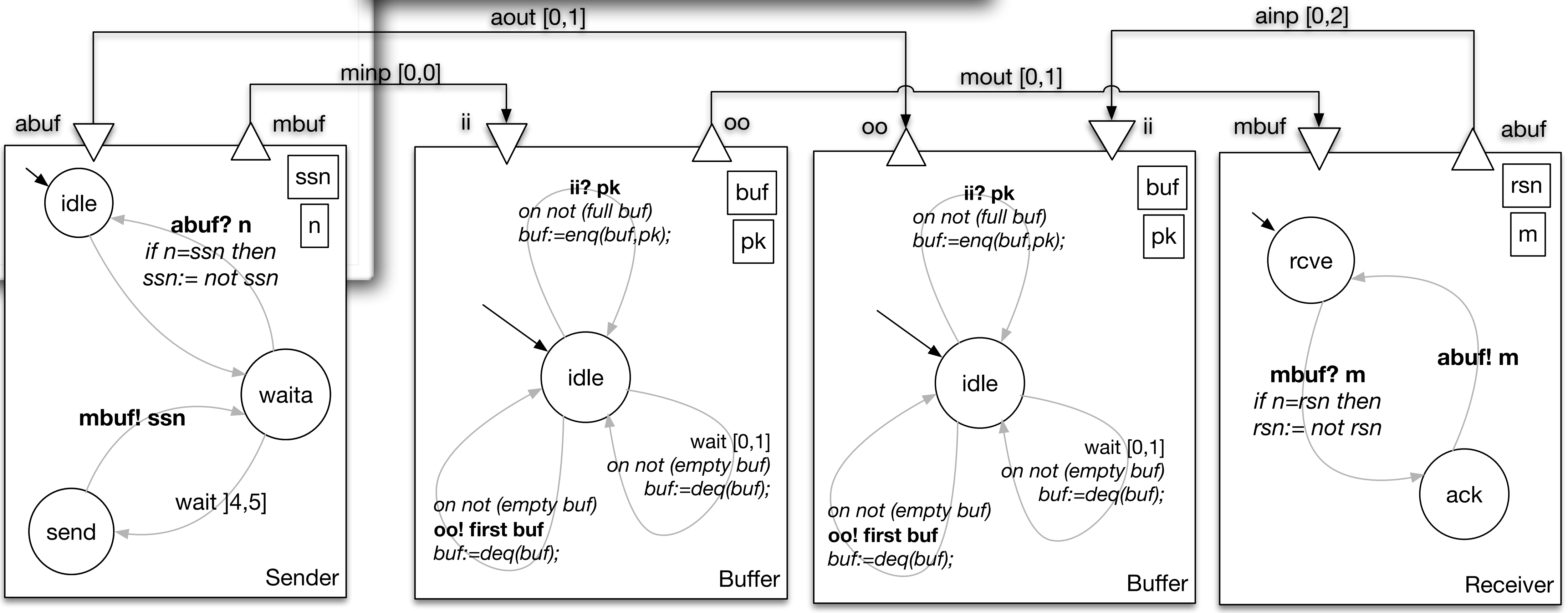
Alternating Bit Protocol



```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    [] /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    [] /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
       /* should also retrieve data from user */
       to waita
    from send
       mbuff! ssn;
       to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
      [] wait ]4,5];
         /* resend */
         to send
      end
```

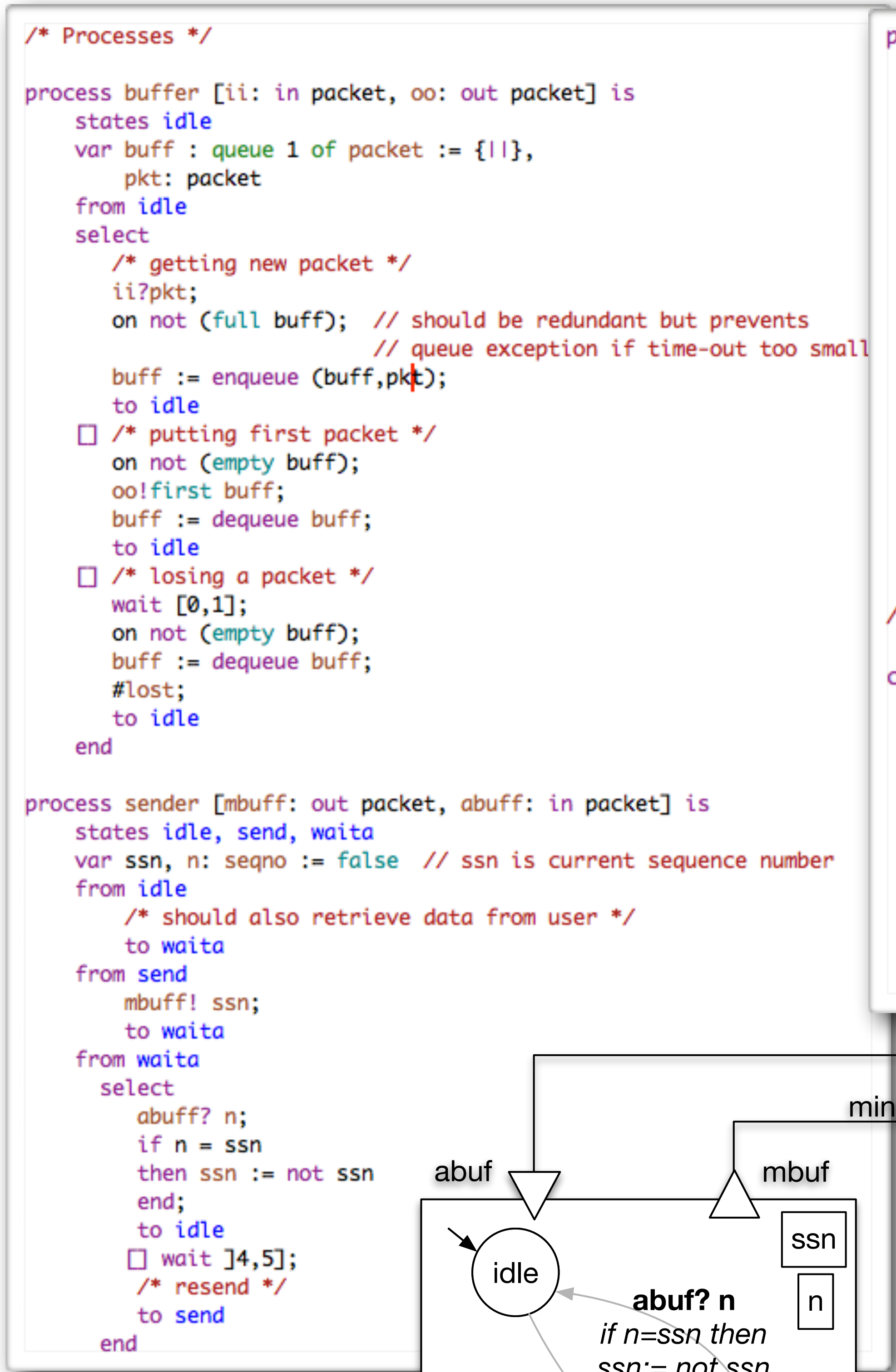


```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port minp : packet in [0,0],
         mout : packet in [0,1],
         ainp : packet in [0,2],
         aout : packet in [0,1]
    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```

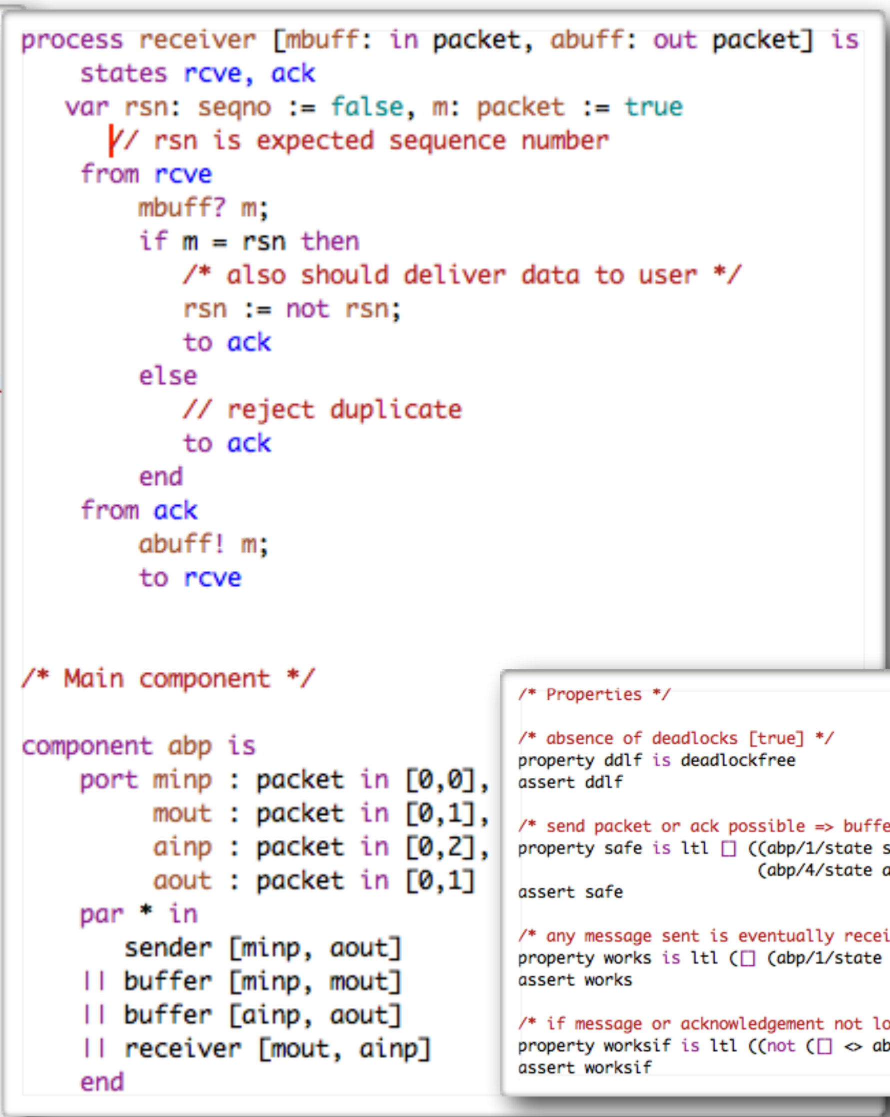


Fiacre component example

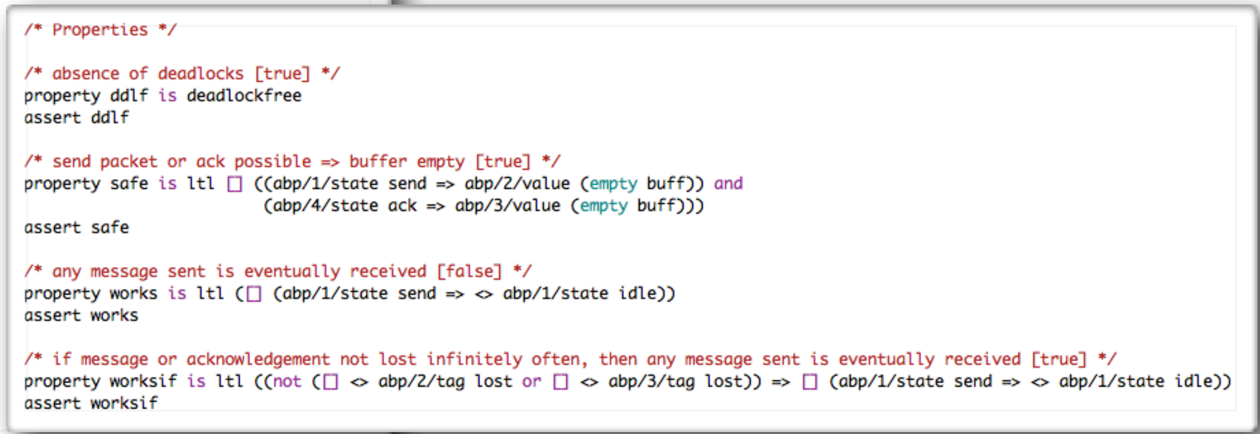
Alternating Bit Protocol



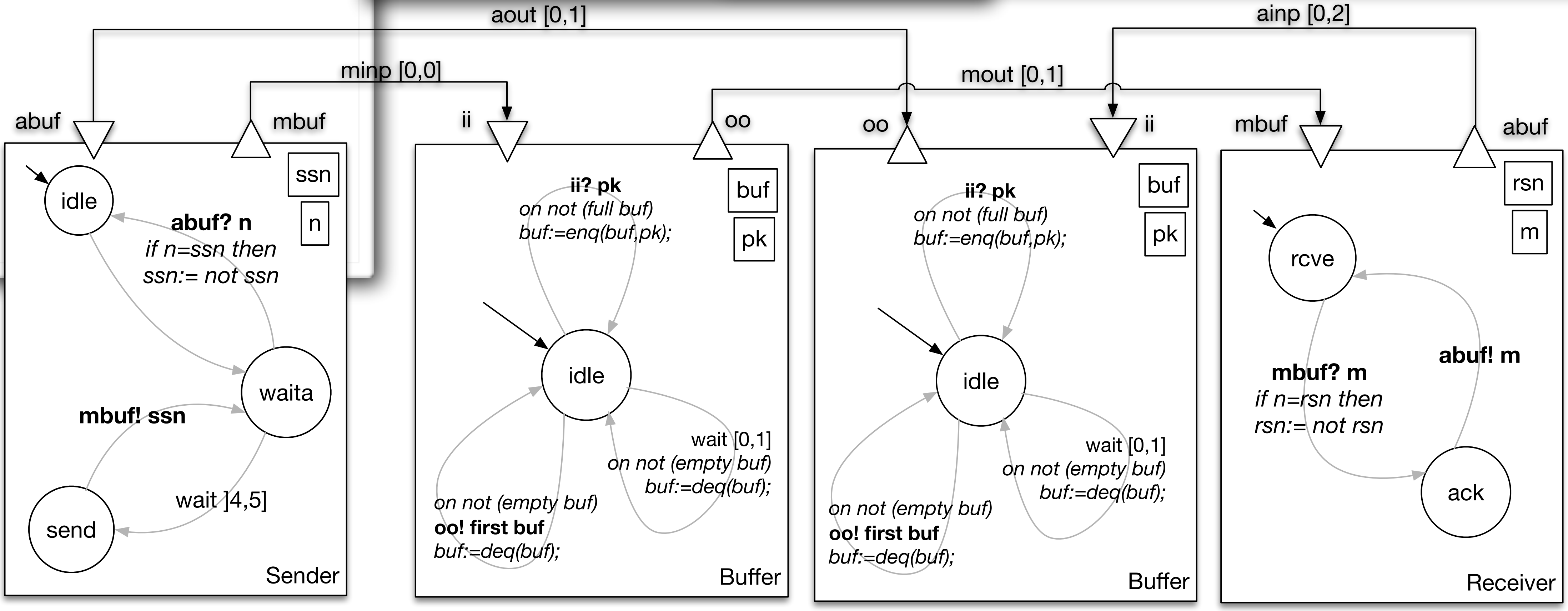
```
/* Processes */

process buffer [ii: in packet, oo: out packet] is
    states idle
    var buff : queue 1 of packet := {||},
        pkt: packet
    from idle
    select
       /* getting new packet */
       ii?pkt;
       on not (full buff);  // should be redundant but prevents
                            // queue exception if time-out too small
       buff := enqueue (buff,pkt);
       to idle
    [] /* putting first packet */
       on not (empty buff);
       oo!first buff;
       buff := dequeue buff;
       to idle
    [] /* losing a packet */
       wait [0,1];
       on not (empty buff);
       buff := dequeue buff;
       #lost;
       to idle
    end

process sender [mbuff: out packet, abuff: in packet] is
    states idle, send, waita
    var ssn, n: seqno := false  // ssn is current sequence number
    from idle
        /* should also retrieve data from user */
        to waita
    from send
        mbuff! ssn;
        to waita
    from waita
      select
         abuff? n;
         if n = ssn
         then ssn := not ssn
         end;
         to idle
      [] wait ]4,5];
         /* resend */
         to send
      end
```



```
process receiver [mbuff: in packet, abuff: out packet] is
    states rcve, ack
   var rsn: seqno := false, m: packet := true
      // rsn is expected sequence number
    from rcve
        mbuff? m;
        if m = rsn then
           /* also should deliver data to user */
           rsn := not rsn;
           to ack
        else
           // reject duplicate
           to ack
        end
    from ack
        abuff! m;
        to rcve


/* Main component */

component abp is
    port minp : packet in [0,0],
         mout : packet in [0,1],
         ainp : packet in [0,2],
         aout : packet in [0,1]
    par * in
       sender [minp, aout]
    || buffer [minp, mout]
    || buffer [ainp, aout]
    || receiver [mout, ainp]
    end
```

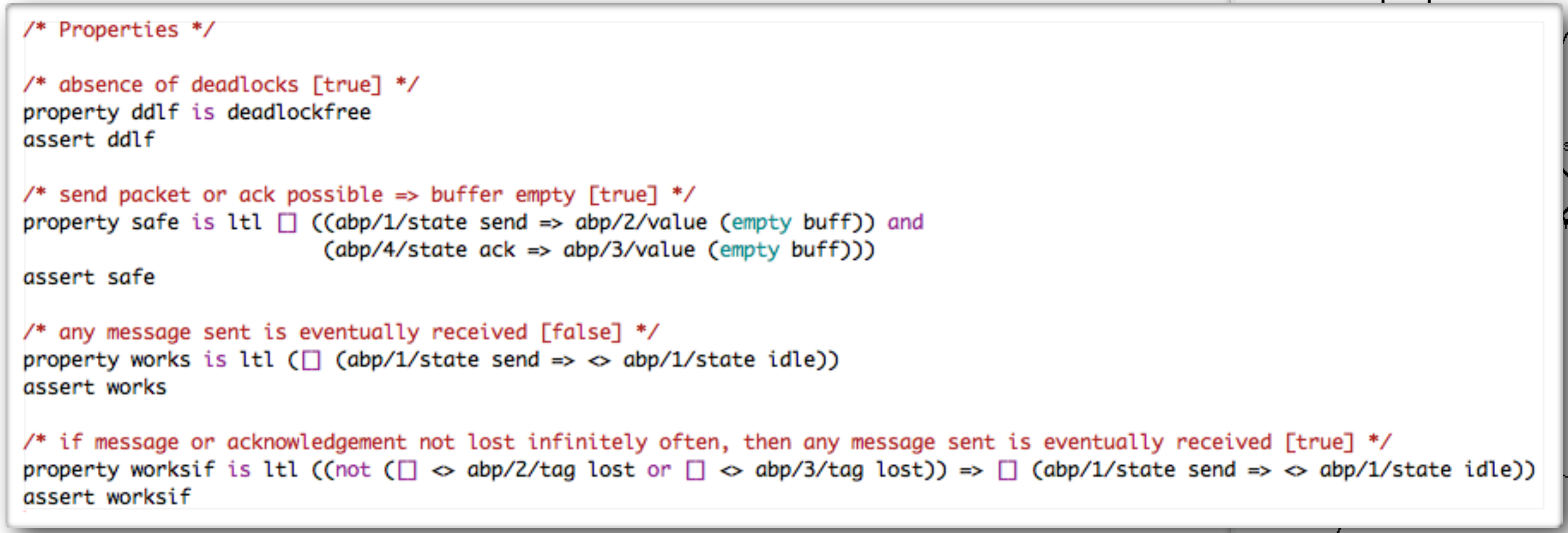


```
/* Properties */

/* absence of deadlocks [true] */
property ddlf is deadlockfree
assert ddlf

/* send packet or ack possible => buffer empty [true] */
property safe is ltl [] ((abp/1/state send => abp/2/value (empty buff)) and
                         (abp/4/state ack => abp/3/value (empty buff)))
assert safe

/* any message sent is eventually received [false] */
property works is ltl ([] (abp/1/state send => <> abp/1/state idle))
assert works

/* if message or acknowledgement not lost infinitely often, then any message sent is eventually received [true] */
property worksif is ltl ((not ([] <> abp/2/tag lost or [] <> abp/3/tag lost)) => [] (abp/1/state send => <> abp/1/state idle))
assert worksif
```

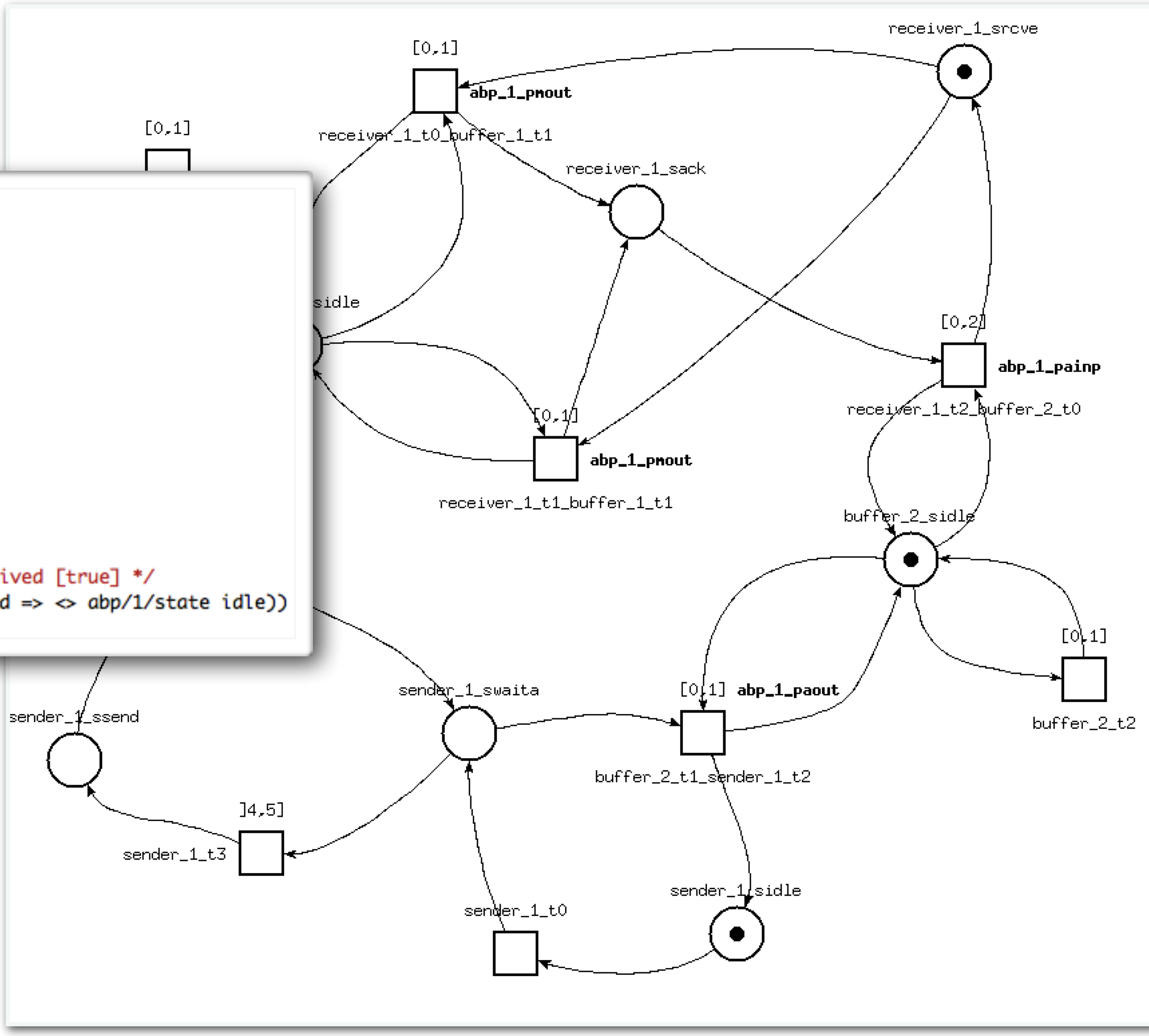


Automatically translated to Time Petri Net

Verification with Model Checking offline with **TINA**:



```
/* Properties */

/* absence of deadlocks [true] */
property ddlf is deadlockfree
assert ddlf

/* send packet or ack possible => buffer empty [true] */
property safe is ltl □ ((abp/1/state send => abp/2/value (empty buff)) and
                        (abp/4/state ack => abp/3/value (empty buff)))
assert safe

/* any message sent is eventually received [false] */
property works is ltl (□ (abp/1/state send => <> abp/1/state idle))
assert works

/* if message or acknowledgement not lost infinitely often, then any message sent is eventually received [true] */
property worksif is ltl ((not (□ <> abp/2/tag lost or □ <> abp/3/tag lost)) => □ (abp/1/state send => <> abp/1/state idle))
assert worksif
```



- temporal properties in LTL,
- … and patterns

H-FIACRE an extension for Runtime Verification, along the HIPPO Engine

- H-FIACRE is FIACRE plus these “extensions”:
  - external events can be connected to *event ports* with external C/C++ function calls
  - external functions can be called in Fiacre *task*
    - start *task(args)* (call the *task* with *args*),
    - the C/C++ *task* is executed in its own thread
    - sync *task value* (to wait for the *task* to finish and return a *value*)
- HIPPO is an engine able to run the time TTS model resulting from a H-FIACRE model

```
type tyEvt is record time : int, id : nat end
type tyDblEvt is array 2 of tyEvt

event e : tyEvt is c_click
task t (tyDblEvt) : nat is c_print

process double_event is
  states wait_first, wait_second, start_print, w
  var tmp : tyDblEvt := [{time=0,id=0}, {time=0,
  from wait_first
    e?tmp[0]; /* wait first event, assign value
    to wait_second
  from wait_second
    select
      wait [200,200];
      to wait_first
    []e?tmp[1]; /* wait second event, assign va
      to start_print
    end
  from start_print
    start t (tmp); /* start task t */
    to wait_print
  from wait_print
    sync t ret; /* wait end of task t */
    tmp := [{time=0,id=0}, {time=0,id=0}];
    to wait_first
```

H-Fiacre behavior in Fiacre

```
type tyEvt is record time : int, id : nat end
type tyDblEvt is array 2 of tyEvt

event e : tyEvt is c_click
task t (tyDblEvt) : nat is c_print

process double_event is
  states wait_first, wait_second, start_print, wait_print
  var tmp : tyDblEvt := [{time=0,id=0}, {time=0,id=0}], ret : nat
  from wait_first
    e?tmp[0]; /* wait first event, assign value to tmp[0] */
    to wait_second
  from wait_second
    select
      wait [200,200];
      to wait_first
    []e?tmp[1]; /* wait second event, assign value to tmp[1] */
      to start_print
    end
  from start_print
    start t (tmp); /* start task t */
    to wait_print
  from wait_print
    sync t ret; /* wait end of task t */
    tmp := [{time=0,id=0}, {time=0,id=0}];
    to wait_first
```

```
process p_task_t [
    t_SyncGlobal : none,
    t_activate_1, t_activate_2, ..., t_activate_n   : tyIn,
    t_terminated_1, t_activate_2, ... t_activate_n  : tyOut
  ] is
  states waiting, running, synchronizing, terminating
  var param : tyIn, ret : tyOut
  from waiting
    select
       t_activate_1?param; to running
    [] t_activate_2?param; to running
    ...
    [] t_activate_n?param; to running
    end
  from running
    ret := c_foo(param); /* The computational function is called */
    wait[$bcrt, $wcrt];  /* simulate the WCRT */
    to synchronizing
  from synchronizing
    t_SyncGlobal; /* Synchronization with the global tick */
    to terminating
  from terminating
    select /* The return value are written */
       t_terminated_1 ! ret; to waiting
    [] t_terminated_2 ! ret; to waiting
    ...
    [] t_terminated_n ! ret; to waiting
    end
```

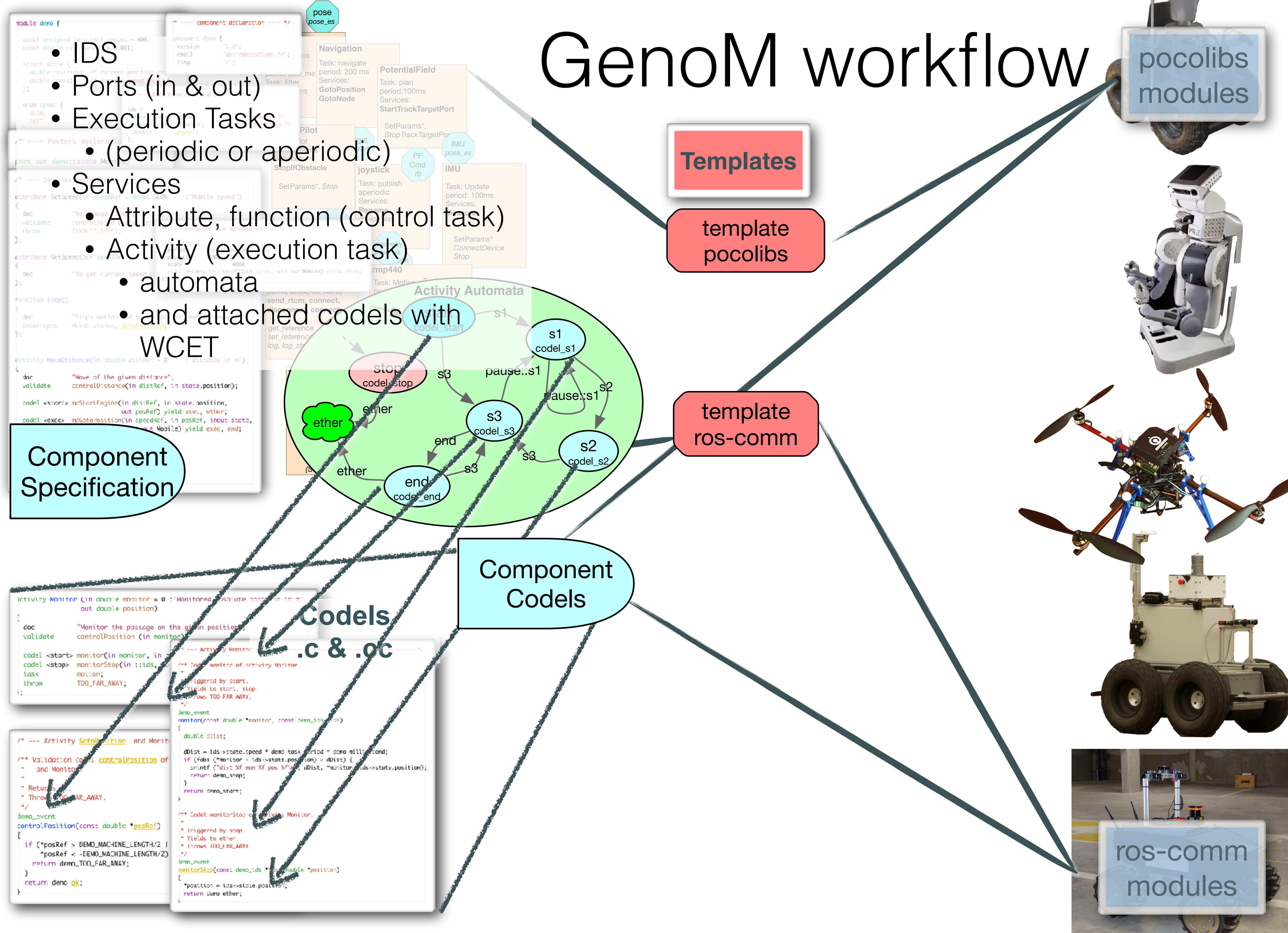
H-Fiacre behavior in Fiacre

```
type tyEvt is record time : int, id : nat end
type tyDblEvt is array 2 of tyEvt

event e : tyEvt is c_click
task t (tyDblEvt) : nat is c_print

process double_event is
  states wait_first, wait_second, start_print, wait_print
  var tmp : tyDblEvt := [{time=0,id=0}, {time=0,id=0}], ret : nat
  from wait_first
    e?tmp[0]; /* wait first event, assign value to tmp[0] */
    to wait_second
  from wait_second
    select
      wait [200,200];
      to wait_first
    []e?tmp[1]; /* wait second event, assign value to tmp[1] */
      to start_print
    end
  from start_print
    start t (tmp); /* start task t */
    to wait_print
  from wait_print
    sync t ret; /* wait end of task t */
    tmp := [{time=0,id=0}, {time=0,id=0}];
    to wait_first
```

```
process p_task_t [
    t_SyncGlobal : none,
    t_activate_1, t_activate_2, ..., t_activate_n   : tyIn,
    t_terminated_1, t_activate_2, ... t_activate_n  : tyOut
  ] is
  states waiting, running, synchronizing, terminating
  var param : tyIn, ret : tyOut
  from waiting
    select
       t_activate_1?param; to running
    [] t_activate_2?param; to running
    ...
    [] t_activate_n?param; to running
    end
  from running
    ret := c_foo(param); /* The computational function is called */
    wait[$bcrt, $wcrt];  /* simulate the WCRT */
    to synchronizing
  from synchronizing
    t_SyncGlobal; /* Synchronization with the global tick */
    to terminating
  from terminating
    select /* The return value are written */
       t_terminated_1 ! ret; to waiting
    [] t_terminated_2 ! ret; to waiting
    ...
    [] t_terminated_n ! ret; to waiting
    end
```

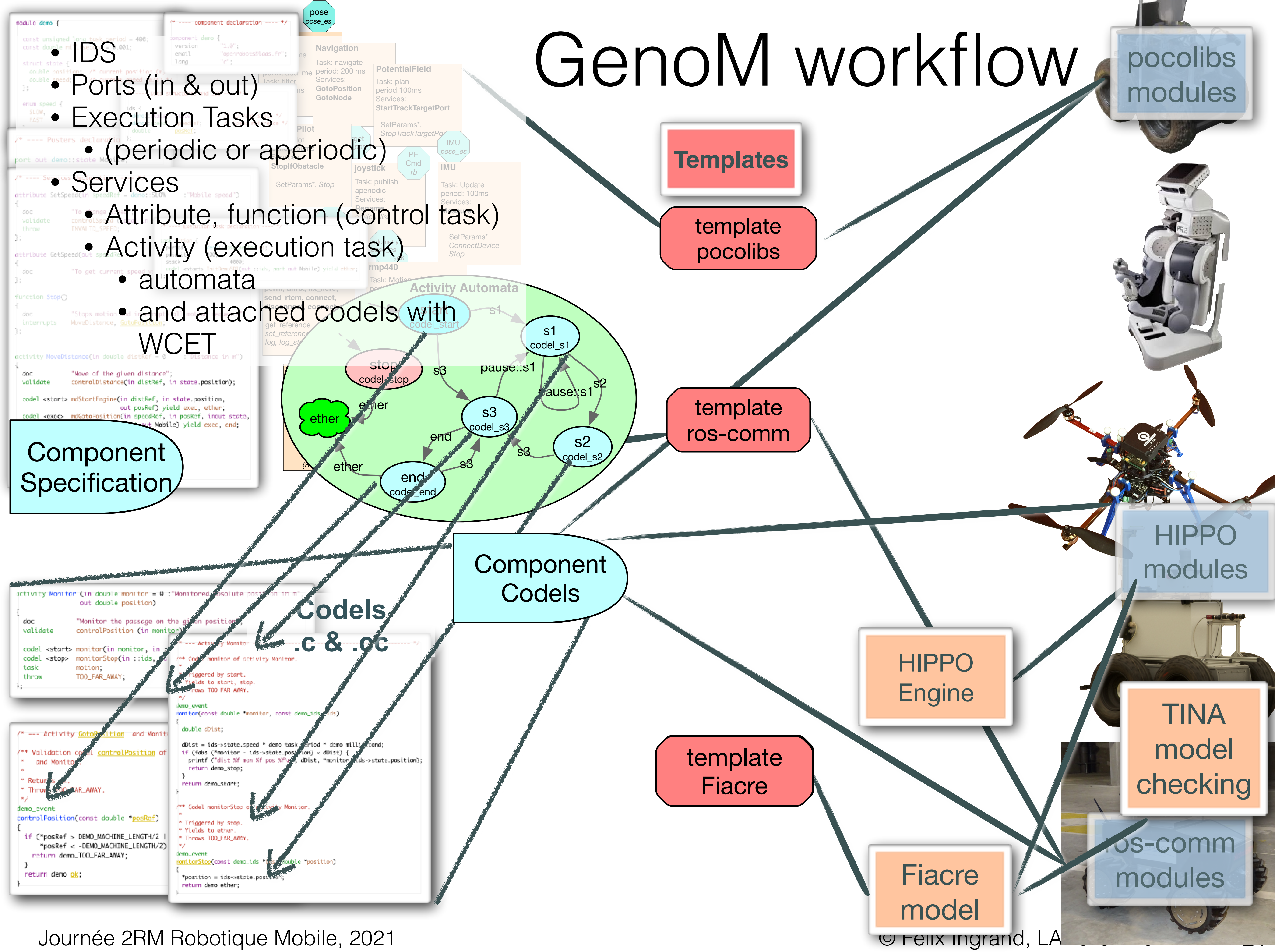
GenoM workflow

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

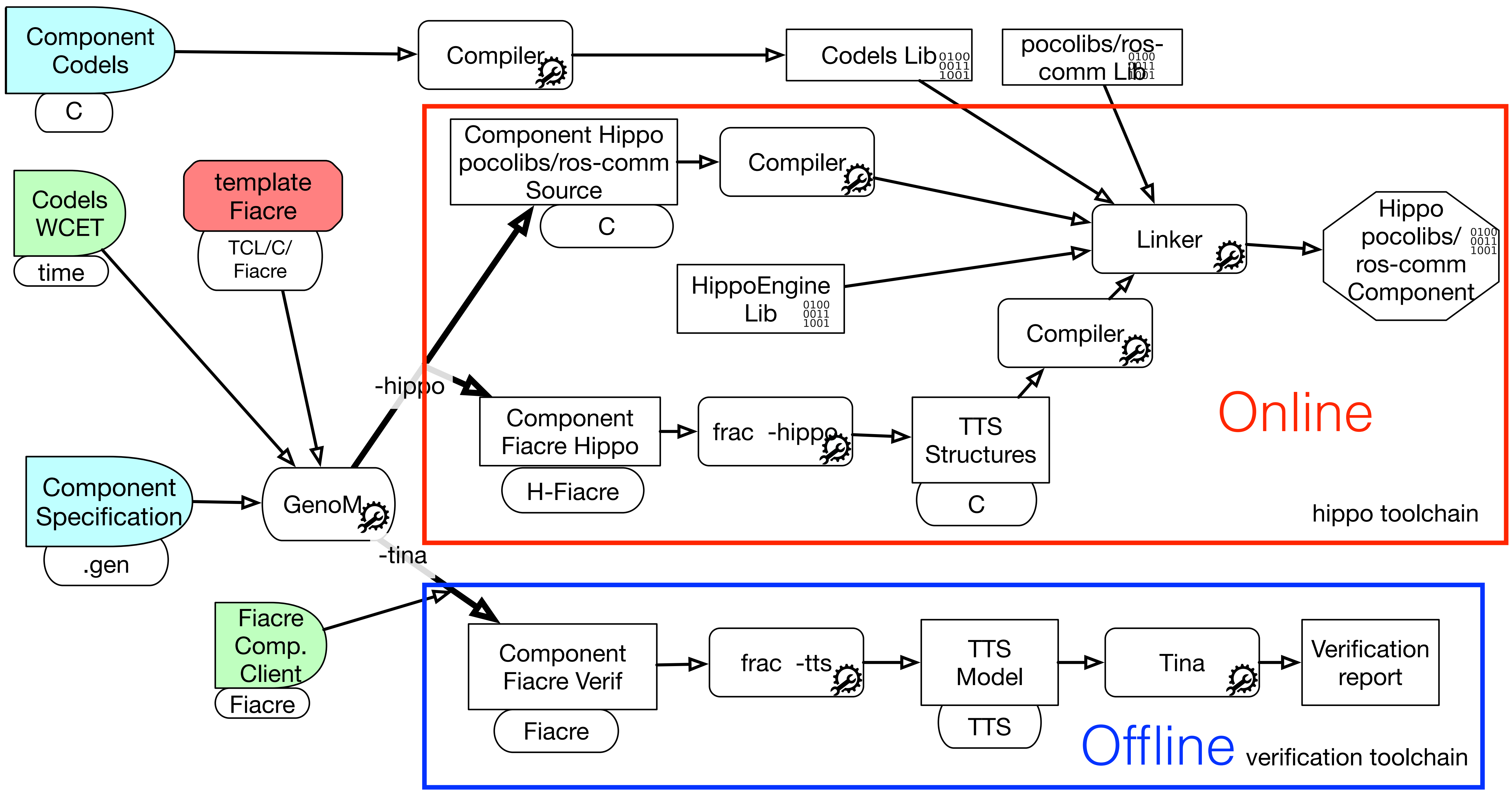


GenoM workflow

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

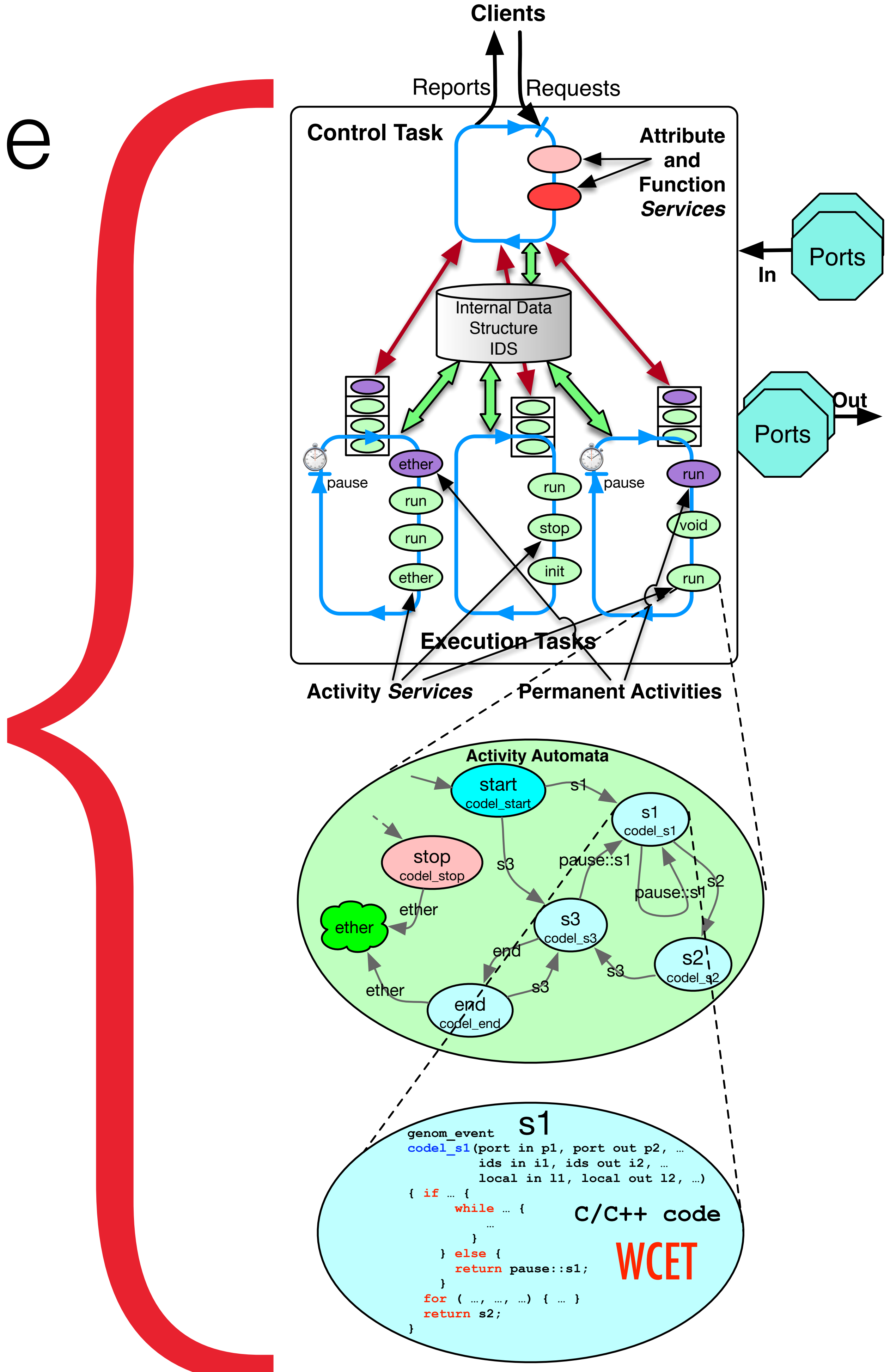


Template FIACRE



What is the content of the formal specifications

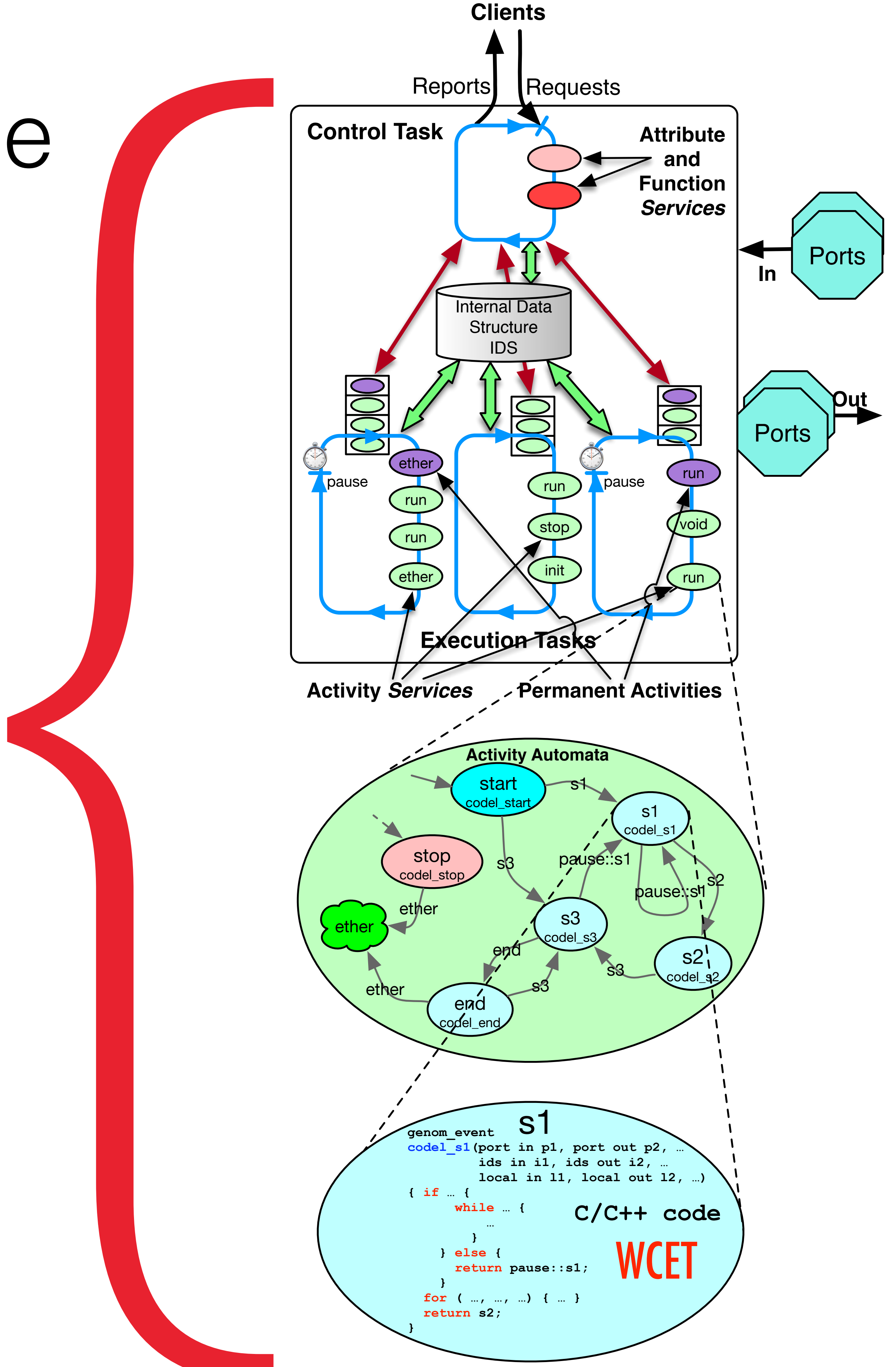
All the algorithms are programmed in FIACRE



```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
         …
        }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

What is the content of the formal specifications

All the algorithms are programmed in FIACRE

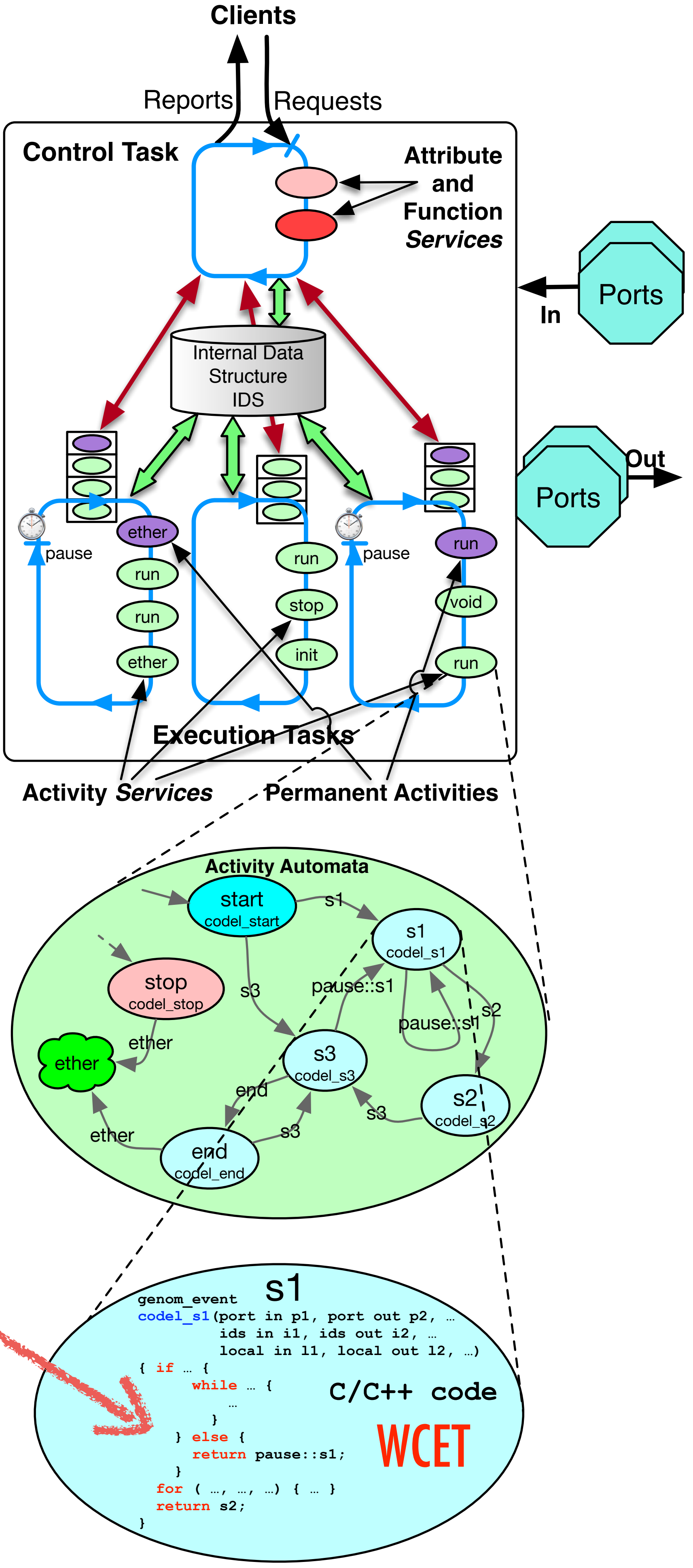


```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
            …
          }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

What is the content of the formal specifications

All the algorithms are programmed in FIACRE

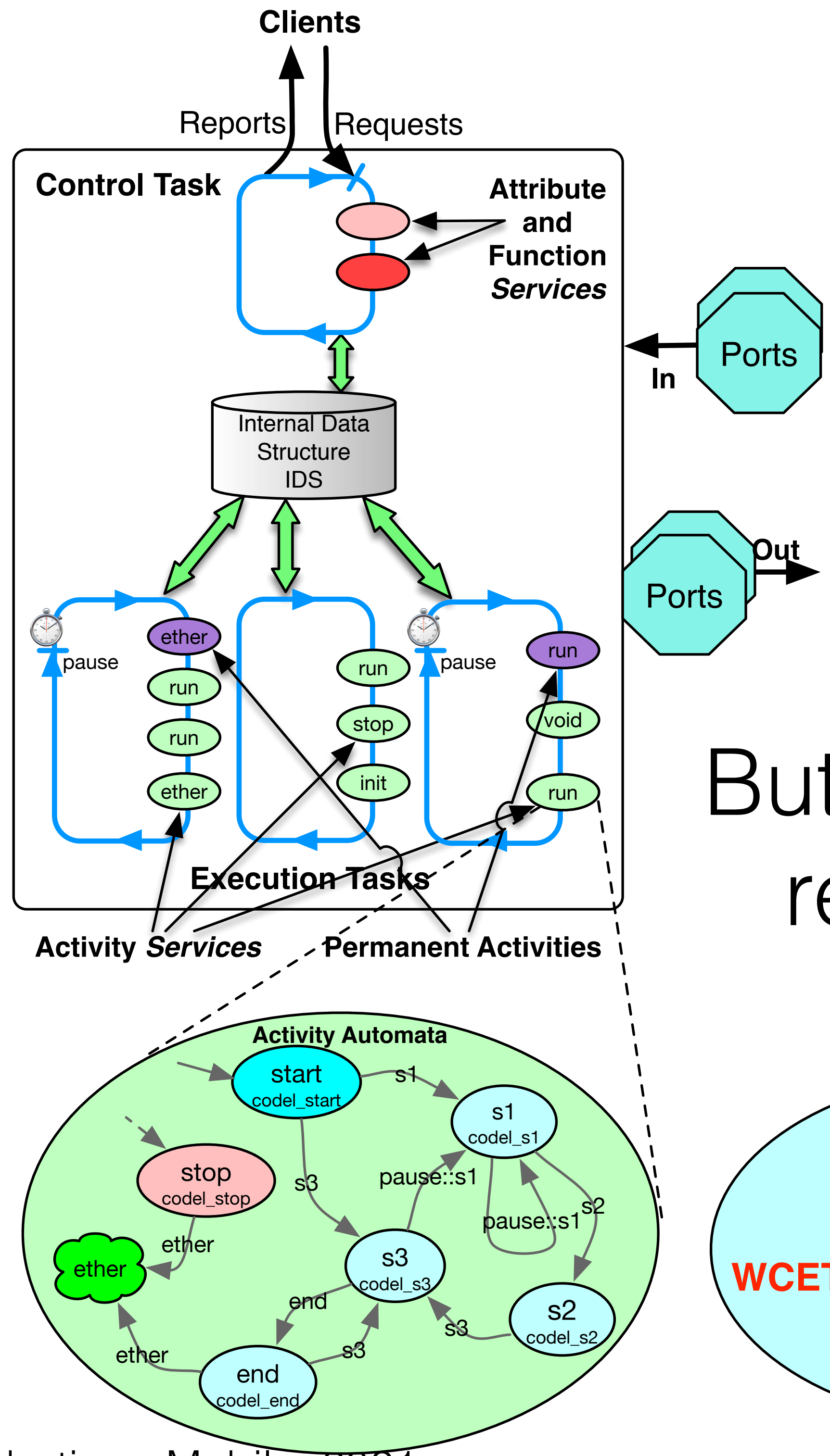
… except the specific user defined C/C++ code which is inside a codel (abstracted with a WCET (TINA) or really executed (Hippo))



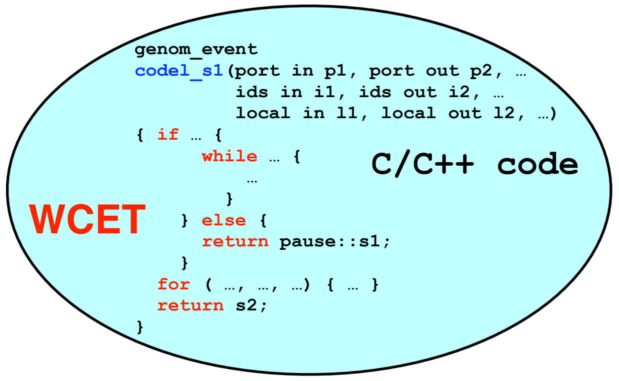
```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
          …
        }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

FIACRE implementation

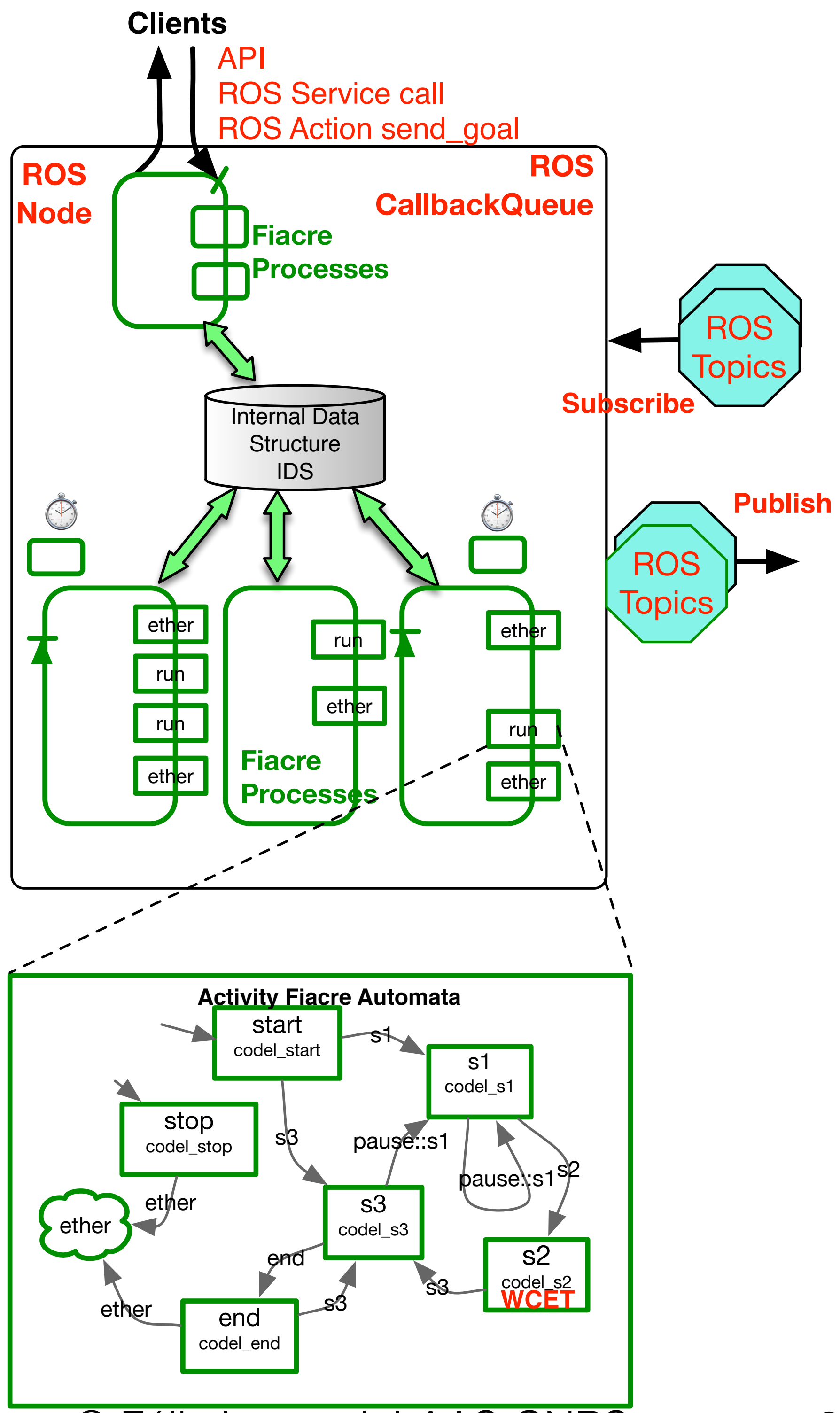
All internal algorithms are rewritten
as FIACRE/H-FIACRE processes



But the codels remain the same


WCET
C/C++ code

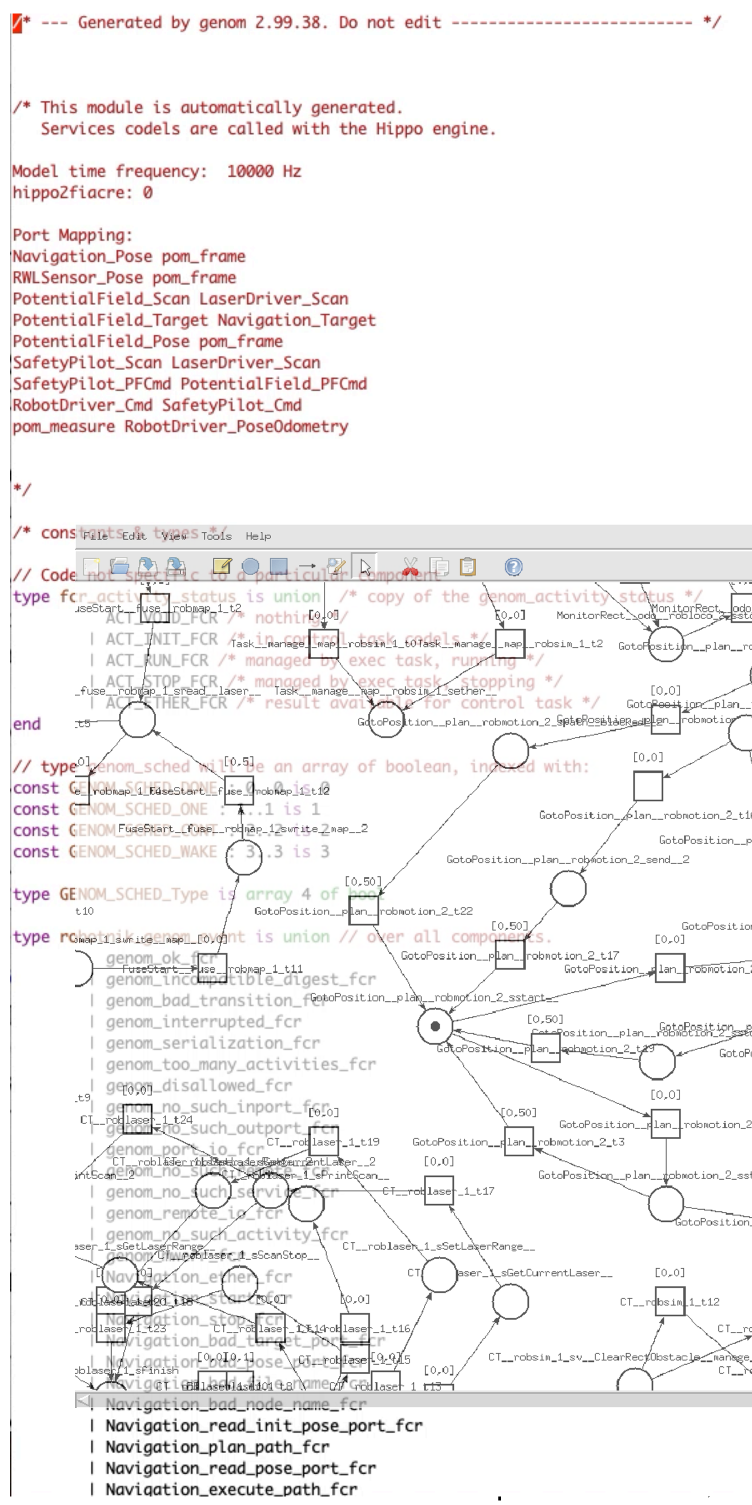
```
genom_event
codel_s1(port in p1, port out p2, …
         ids in i1, ids out i2, …
         local in l1, local out l2, …)
{ if … {
      while … {
          …
        }
    } else {
      return pause::s1;
    }
  for ( …, …, …) { … }
  return s2;
}
```

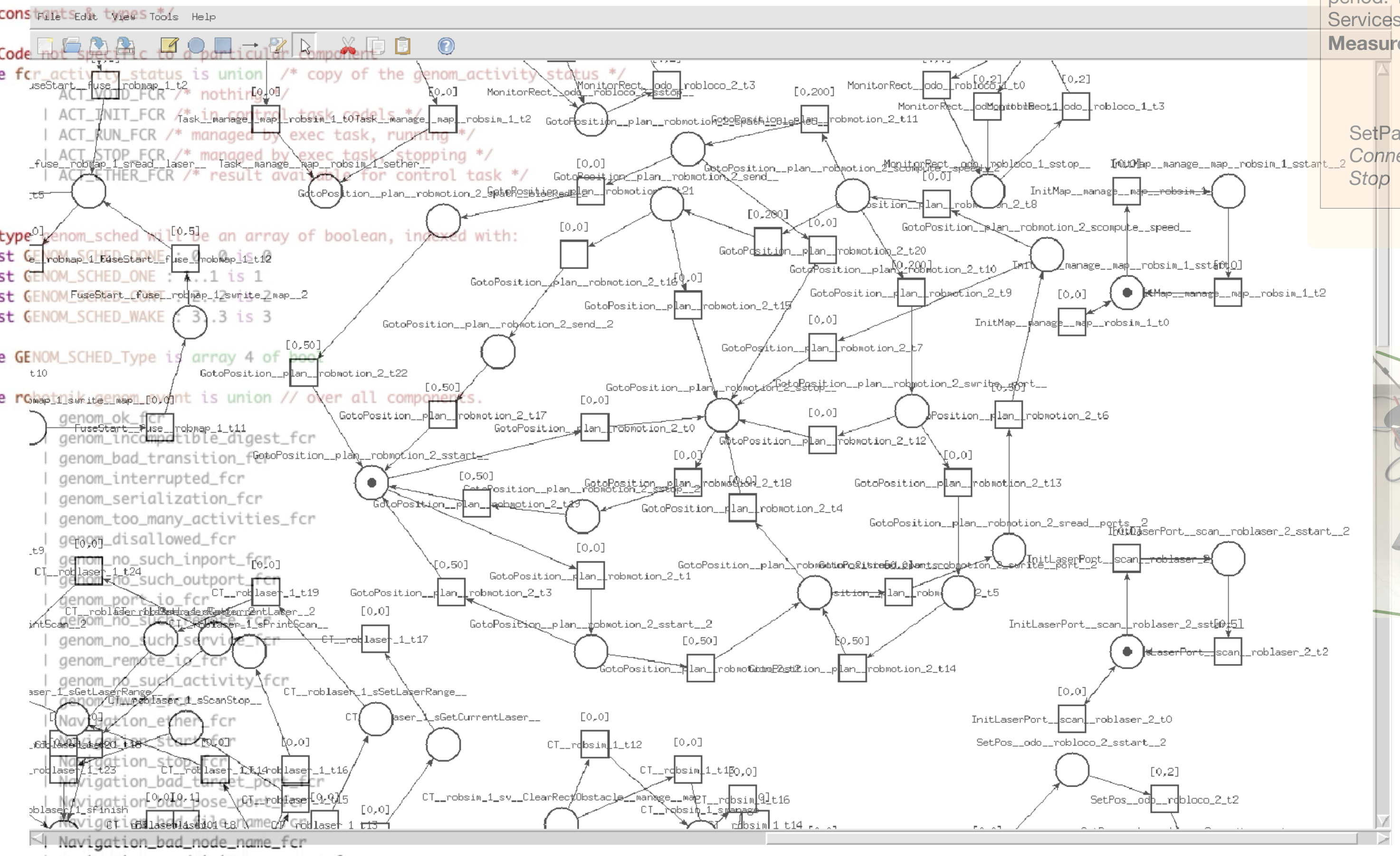


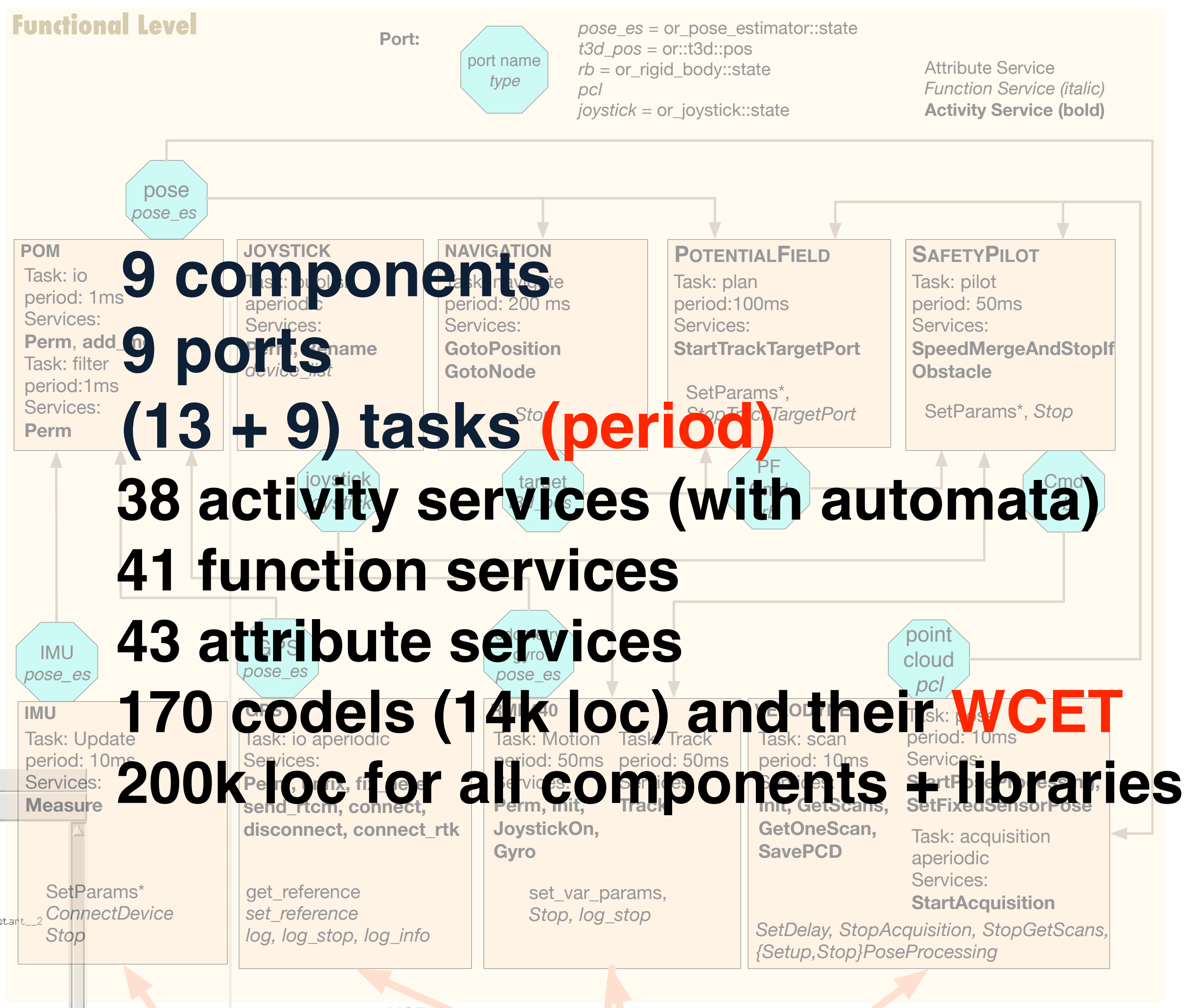
Minnie Fiacre model

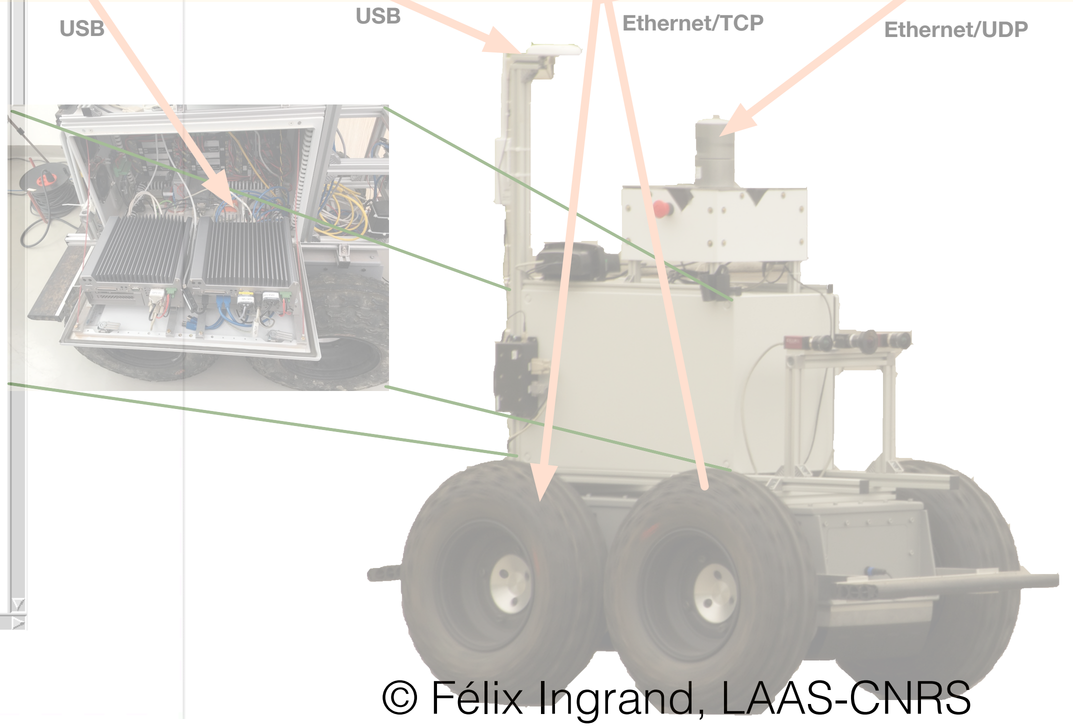
Fiacre model (42000 loc)
for all the components

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


```
/* --- Generated by genom 2.99.38. Do not edit --------------------------- */


/* This module is automatically generated.
   Services codels are called with the Hippo engine.

Model time frequency:  10000 Hz
hippo2fiacre: 0

Port Mapping:
Navigation_Pose pom_frame
RWLSensor_Pose pom_frame
PotentialField_Scan LaserDriver_Scan
PotentialField_Target Navigation_Target
PotentialField_Pose pom_frame
SafetyPilot_Scan LaserDriver_Scan
SafetyPilot_PFCmd PotentialField_PFCmd
RobotDriver_Cmd SafetyPilot_Cmd
pom_measure RobotDriver_PoseOdometry


*/

/* constants & types */

// Code not specific to a particular component
type fcr_activity_status is union  /* copy of the genom_activity_status */
   ACT_VOID_FCR /* nothing */
 | ACT_INIT_FCR /* in control task codels */
 | ACT_RUN_FCR /* managed by exec task, running */
 | ACT_STOP_FCR /* managed by exec task, stopping */
 | ACT_ETHER_FCR /* result available for control task */
end

// type genom_sched will be an array of boolean, indexed with:
const G[illegible]
const GENOM_SCHED_ONE : 1 is 1
const GENOM_[illegible]
const GENOM_SCHED_WAKE : 3 .3 is 3

type GENOM_SCHED_Type is array 4 of bool

type robotkit_genom_[illegible] is union // over all components.
   genom_ok_fcr
 | genom_incompatible_digest_fcr
 | genom_bad_transition_fcr
 | genom_interrupted_fcr
 | genom_serialization_fcr
 | genom_too_many_activities_fcr
 | genom_disallowed_fcr
 | genom_no_such_inport_fcr
 | genom_no_such_outport_fcr
 | genom_port_io_fcr
 | genom_no_such_service_fcr
 | genom_remote_io_fcr
 | genom_no_such_activity_fcr
 | Navigation_ether_fcr
 | Navigation_stop_fcr
 | Navigation_bad_target_port_fcr
 | Navigation_[illegible]
 | Navigation_bad_node_name_fcr
 | Navigation_read_init_pose_port_fcr
 | Navigation_plan_path_fcr
 | Navigation_read_pose_port_fcr
 | Navigation_execute_path_fcr
```

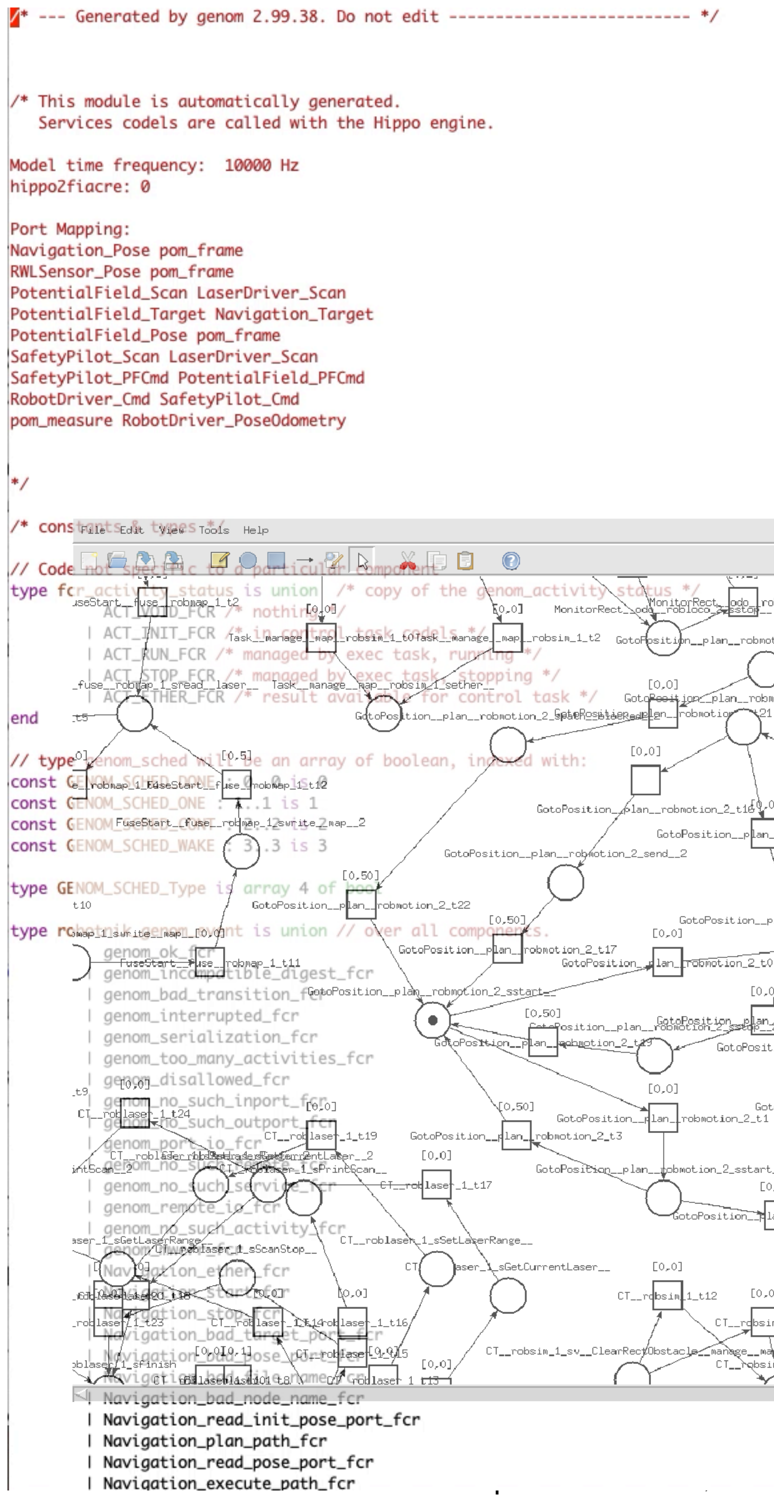




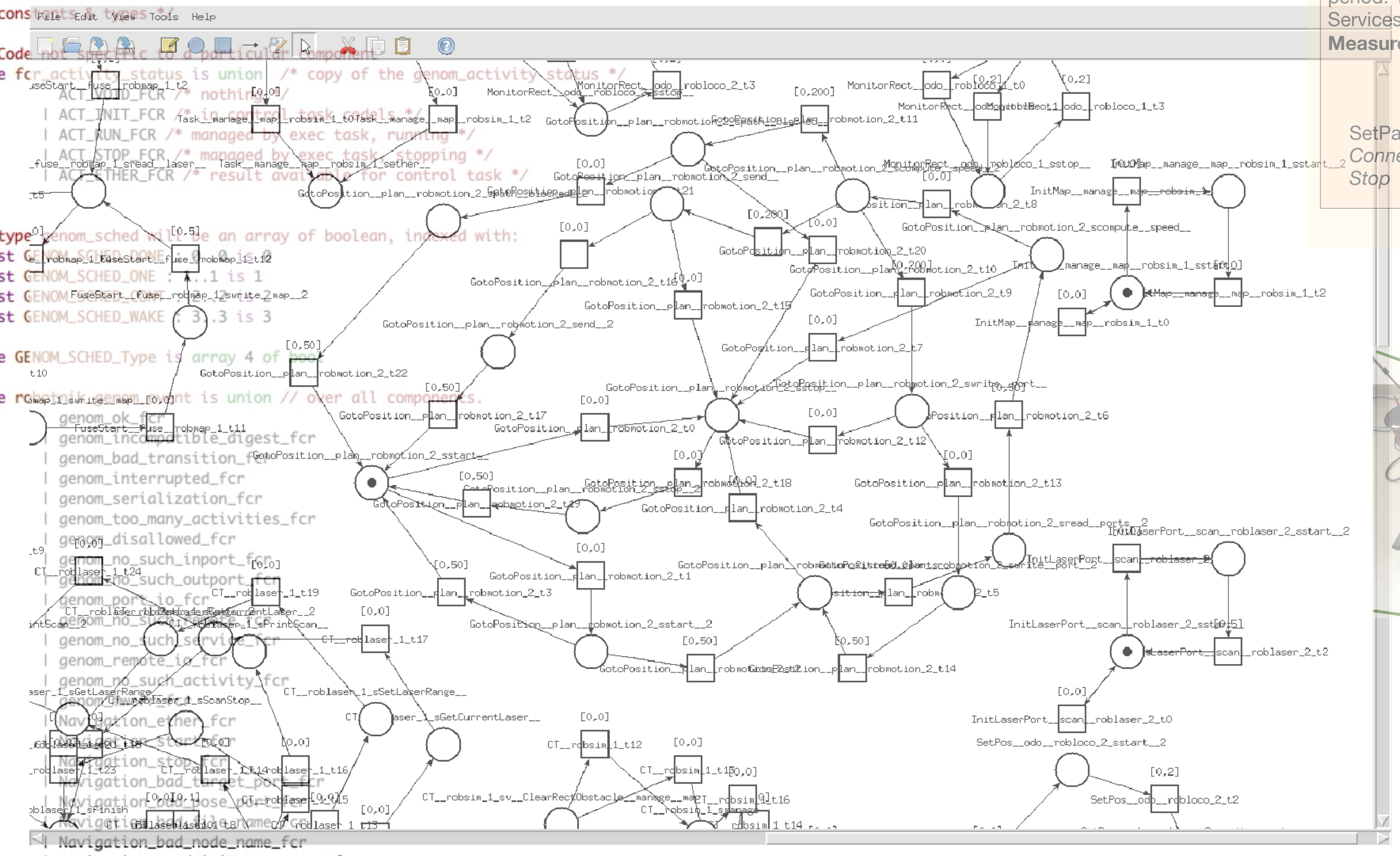


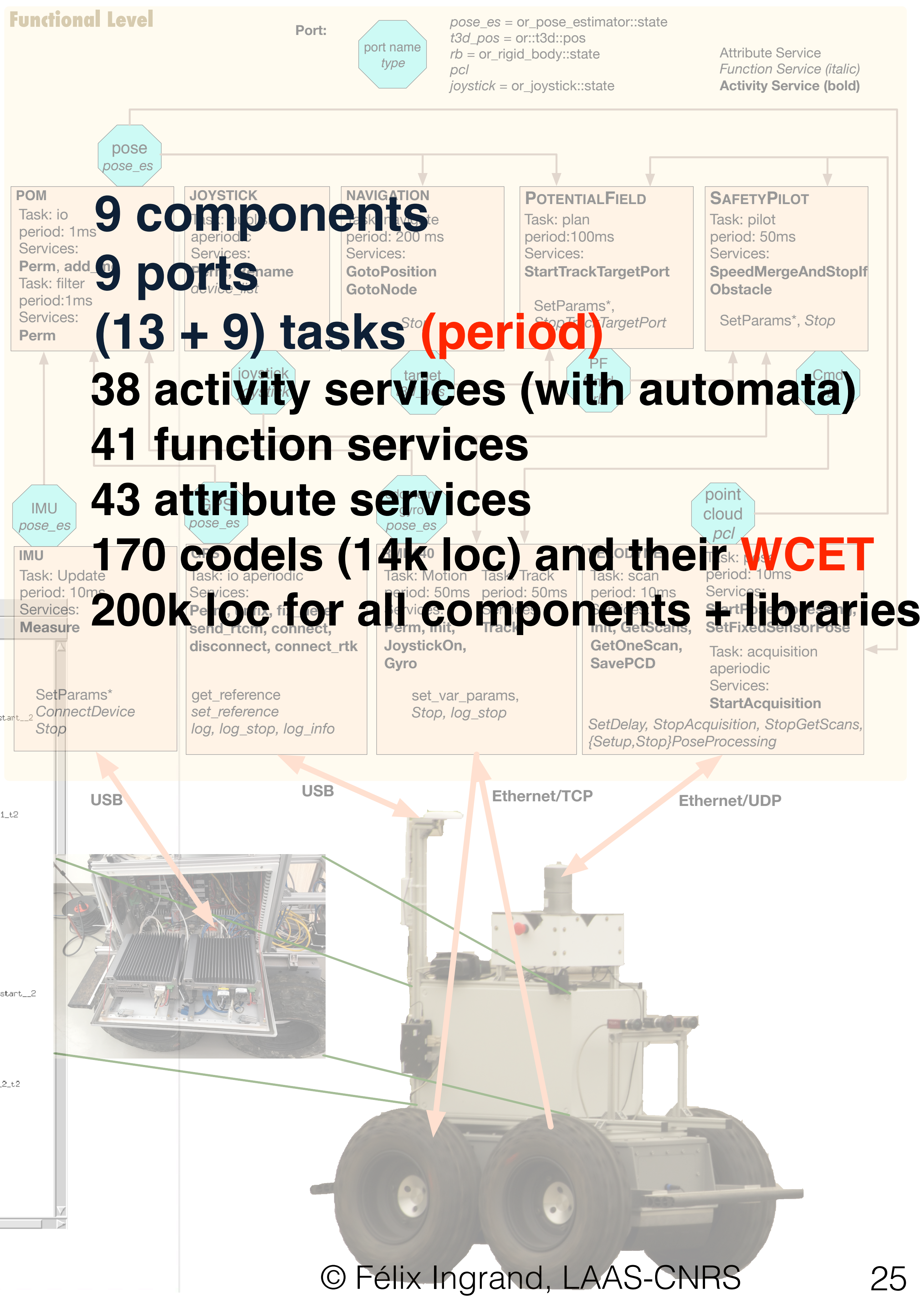
Minnie Fiacre model

Fiacre model (42000 loc)
for all the components



```
/* --- Generated by genom 2.99.38. Do not edit --------------------------- */


/* This module is automatically generated.
   Services codels are called with the Hippo engine.

Model time frequency:  10000 Hz
hippo2fiacre: 0

Port Mapping:
Navigation_Pose pom_frame
RWLSensor_Pose pom_frame
PotentialField_Scan LaserDriver_Scan
PotentialField_Target Navigation_Target
PotentialField_Pose pom_frame
SafetyPilot_Scan LaserDriver_Scan
SafetyPilot_PFCmd PotentialField_PFCmd
RobotDriver_Cmd SafetyPilot_Cmd
pom_measure RobotDriver_PoseOdometry

*/

/* constants & types */

// Code not specific to a particular component
type fcr_activity_status is union  /* copy of the genom_activity_status */
    ACT_VOID_FCR /* nothing */
  | ACT_INIT_FCR /* in control task codels */
  | ACT_RUN_FCR /* managed by exec task, running */
  | ACT_STOP_FCR /* managed by exec task, stopping */
  | ACT_ETHER_FCR /* result available for control task */
end

// type genom_sched will be an array of boolean, indexed with:
const GENOM_SCHED_ONE ... 1 is 1
const GENOM_SCHED_WAKE : 3 .3 is 3

type GENOM_SCHED_Type is array 4 of bool

type robot... is union // over all components.
    genom_ok_fcr
  | genom_incompatible_digest_fcr
  | genom_bad_transition_fcr
  | genom_interrupted_fcr
  | genom_serialization_fcr
  | genom_too_many_activities_fcr
  | genom_disallowed_fcr
  | genom_no_such_inport_fcr
  | genom_no_such_outport_fcr
  | genom_port_io_fcr
  | genom_no_such_service_fcr
  | genom_remote_io_fcr
  | genom_no_such_activity_fcr
  | Navigation_ether_fcr
  | Navigation_stop_fcr
  | Navigation_bad_target_port_fcr
  | Navigation_bad_pose_port_fcr
  | Navigation_bad_node_name_fcr
  | Navigation_read_init_pose_port_fcr
  | Navigation_plan_path_fcr
  | Navigation_read_pose_port_fcr
  | Navigation_execute_path_fcr
```





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

Verification offline with Fiacre/TINA

- ✓ Schedulability of execution tasks for each module

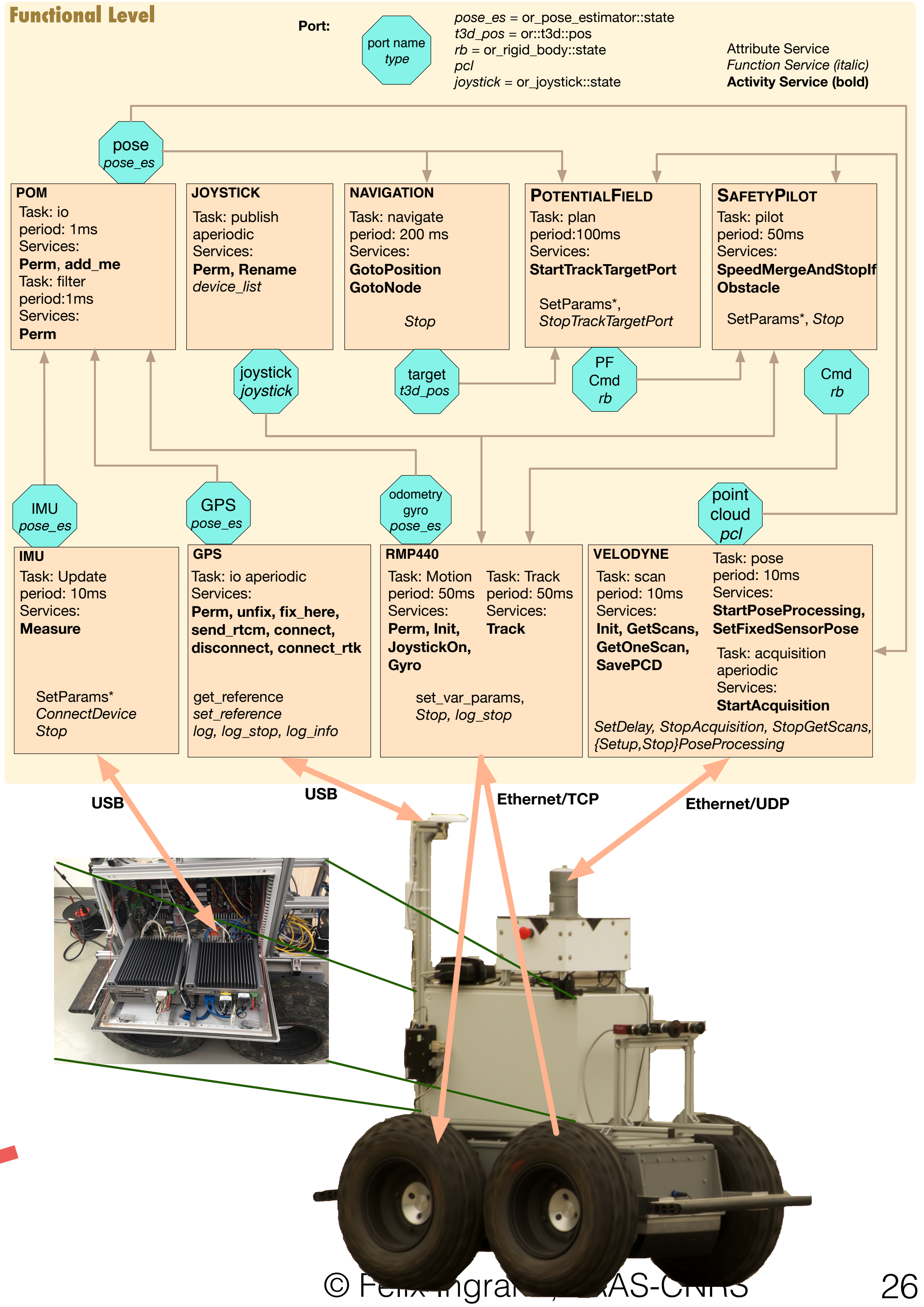
- We have in the model specific states to detect task overshoot
  e.g. ¬(velodyne_scan_overshoot ∨ velodyne_pose_overshoot)

- ✓ Within **rmp440**, exclusion of *JoystickOn* and *Track*

Scenario	*JoystickOn* then *Track*	*Track* then *JoystickOn*
Time	16 min	10 h
#classes	42,714,945	832,778,752
#markings	5,817,082	44,533,432

- ✓ Worst- Case Response Time (WCRT) to stop the robot: 141ms

```
process rmp440_Track_Stopper(&track_started:bool, &track_stopped:bool,
        &TrackTask_activities: Activities_rmp440_TrackTask_Array,
        Track_index: act_inst_rmp440_TrackTask_index_type) is

states wait_started, wait_stop, wait_delay, finished, robot_stopped, robot_NOT_stopped

from wait_started
     wait [0,0];
     on (track_started); // wait the Track service has started
     to wait_stop

from wait_stop  // (no wait) can stop anytime
     TrackTask_activities[Track_index].stop := true;
     to wait_delay

from wait_delay
    wait [141,141]; //<--- This is the response time value we want to measure
    to finished

from finished
    wait [0,0];
    if (track_stopped)  then
       to robot_stopped  //The robot has been stopped before the delay
    else
       to robot_NOT_stopped //The robot has not been fully stopped yet
    end
```



Run Time Verification with H-FIACRE and HIPPO Engine

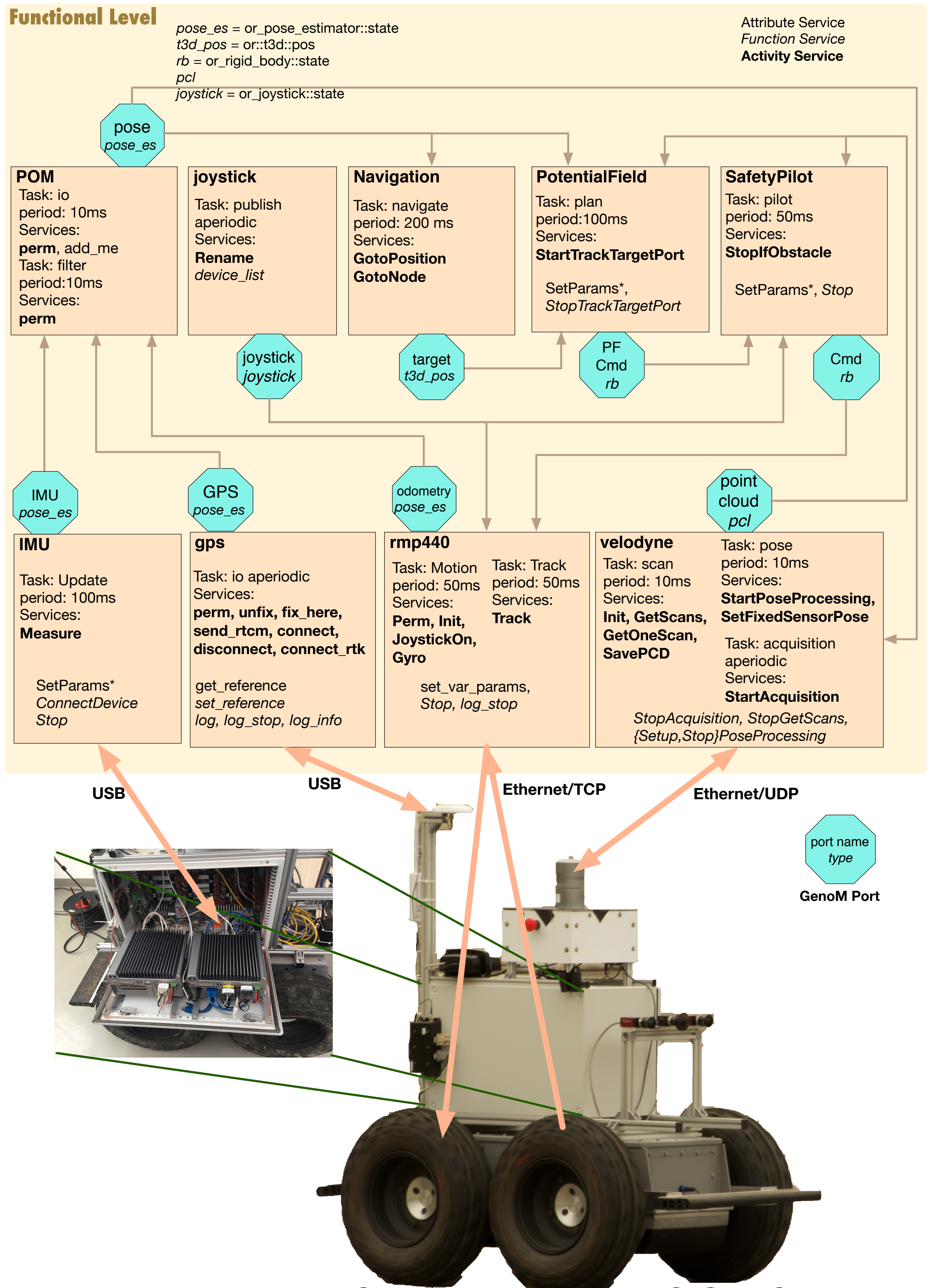
- The H-FIACRE model of any GenoM component can thus be executed by HIPPO
    - codels are called in H-Fiacre *tasks*
    - external requests (from ROS CallbackQueue or pocolibs Mbox) are handled with *event ports*
- The GenoM code and algorithms are now handled by the H-FIACRE model

Regular On Line Verification with the HIPPO Engine

- The regular H-FIACRE model includes some basic verification:
    - Schedulability (period overshoot)
    - WCET overshoot
    - UPR Uninitialized Port Read

Run Time Verification with HIPPO Engine

- Stop if Velodyne point cloud is not refreshed for more than 200ms

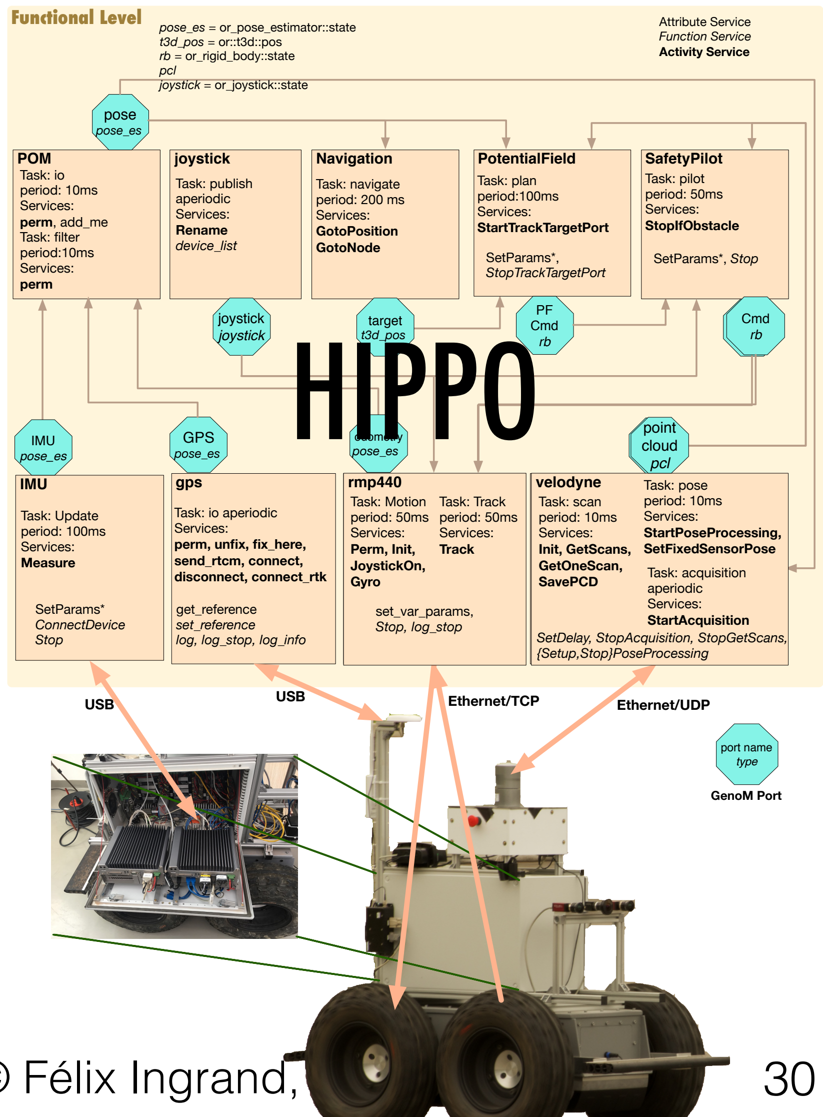


Functional Level
pose_es = or_pose_estimator::state
t3d_pos = or::t3d::pos
rb = or_rigid_body::state
pcl
joystick = or_joystick::state
Attribute Service
Function Service
Activity Service
pose pose_es
POM Task: io period: 10ms Services: perm, add_me Task: filter period:10ms Services: perm
joystick Task: 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

Run Time Verification with HIPPO

```
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;
};
```

```
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;

};
```

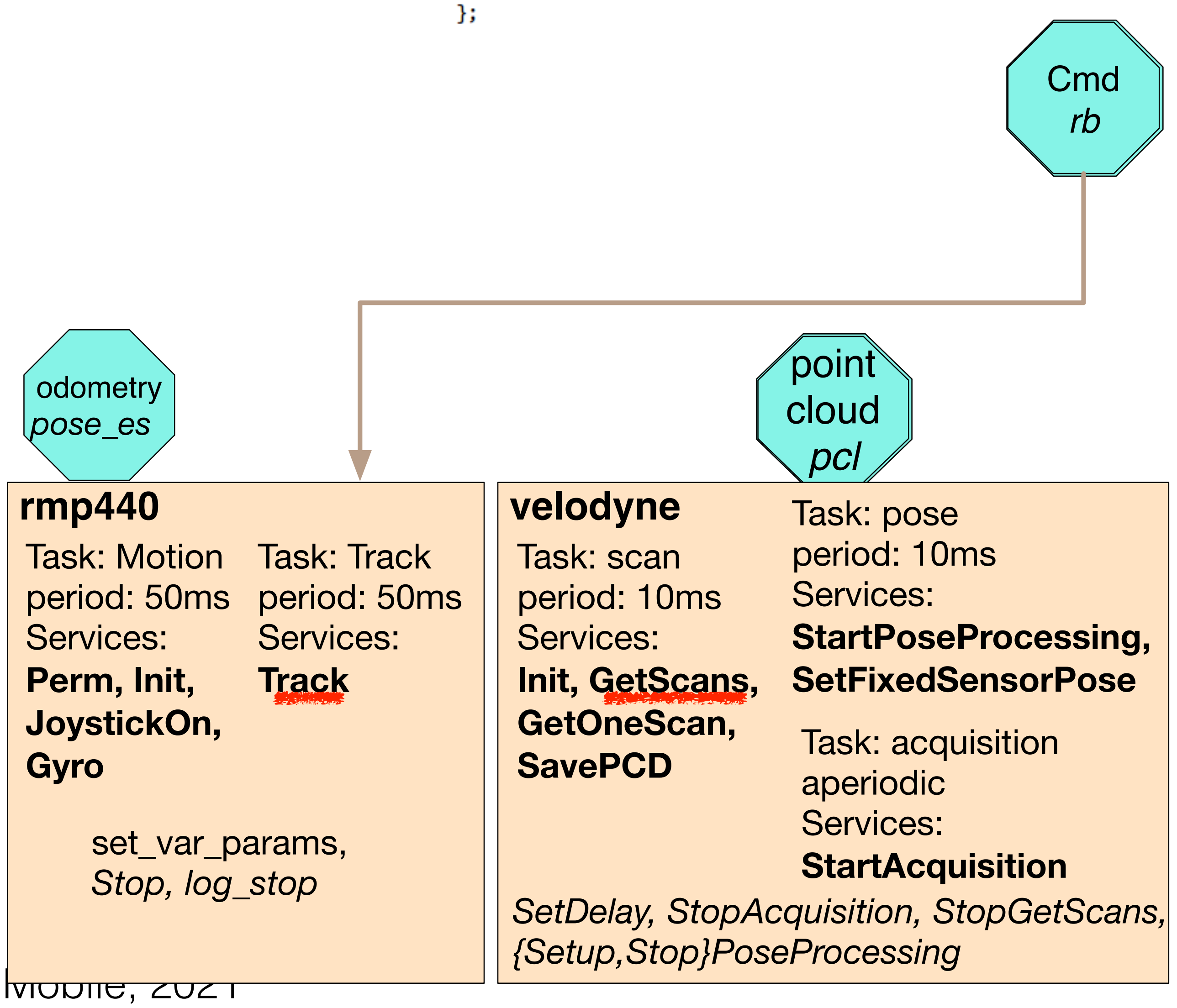


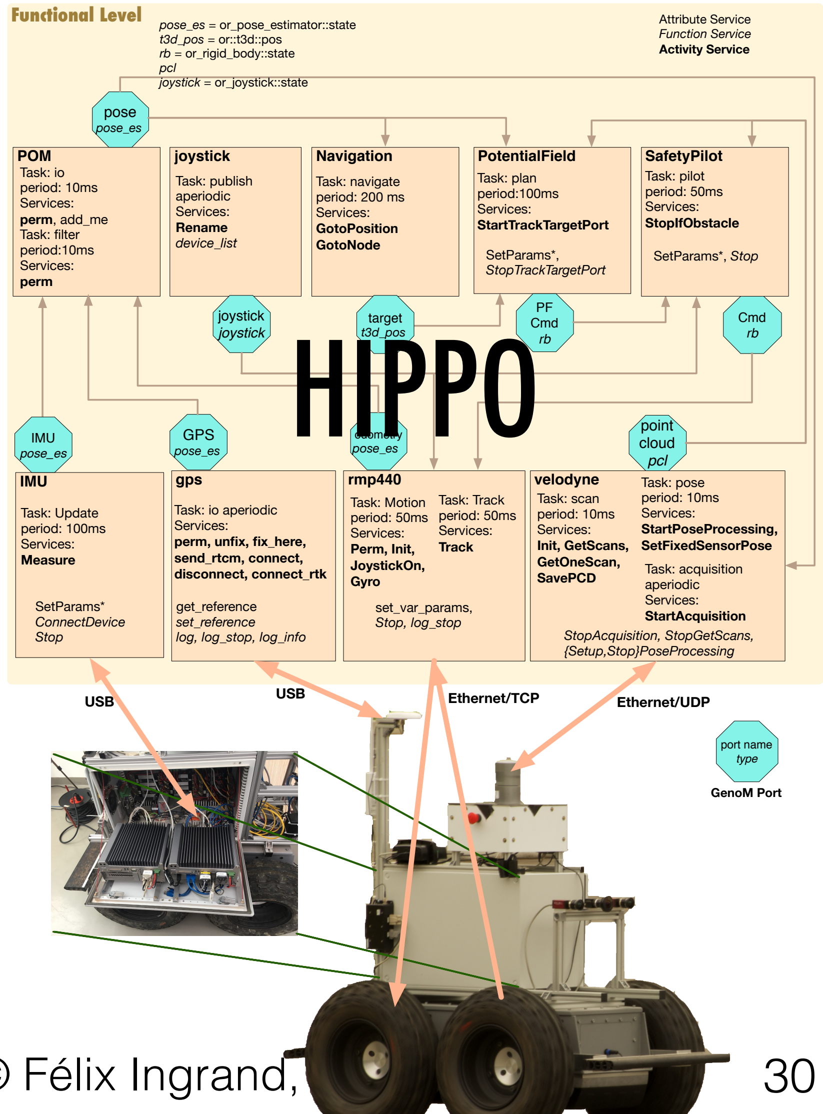
Functional Level
pose_es = or_pose_estimator::state
t3d_pos = or::t3d::pos
rb = or_rigid_body::state
pcl
joystick = or_joystick::state
Attribute Service
Function Service
Activity Service
pose pose_es
POM Task: io period: 10ms Services: perm, add_me Task: filter period:10ms Services: perm
joystick Task: 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
HIPPO
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 SetDelay, StopAcquisition, StopGetScans, {Setup,Stop}PoseProcessing
USB
USB
Ethernet/TCP
Ethernet/UDP
port name type
GenoM Port

Run Time Verification with HIPPO

```
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;
};
```

```
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;

};
```

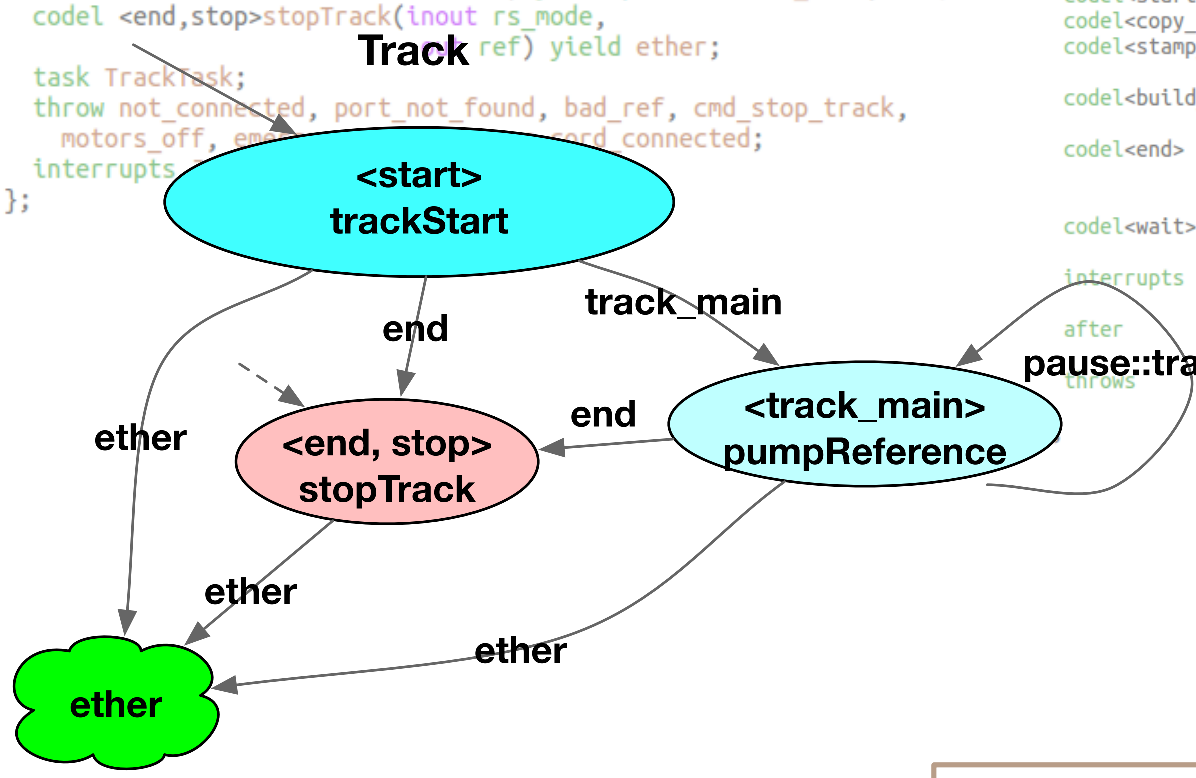


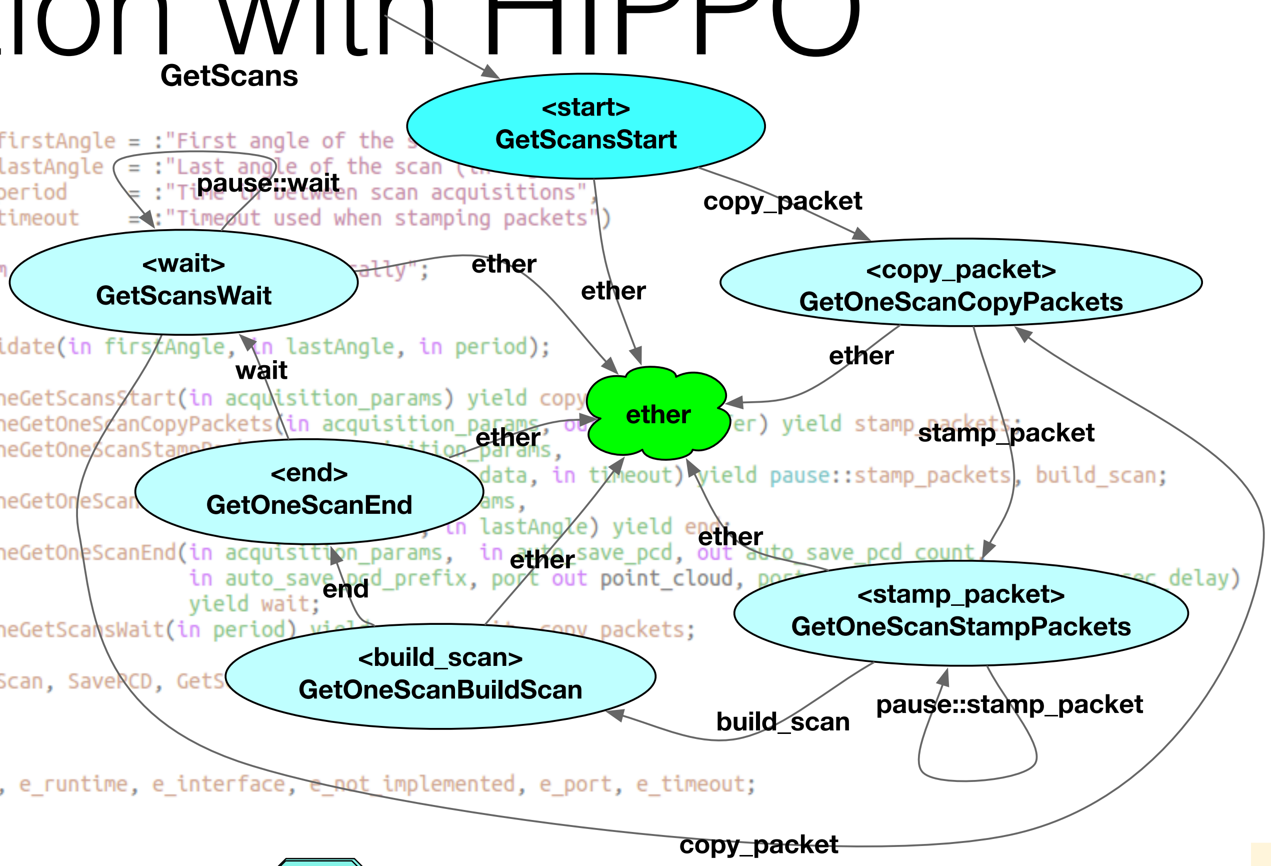


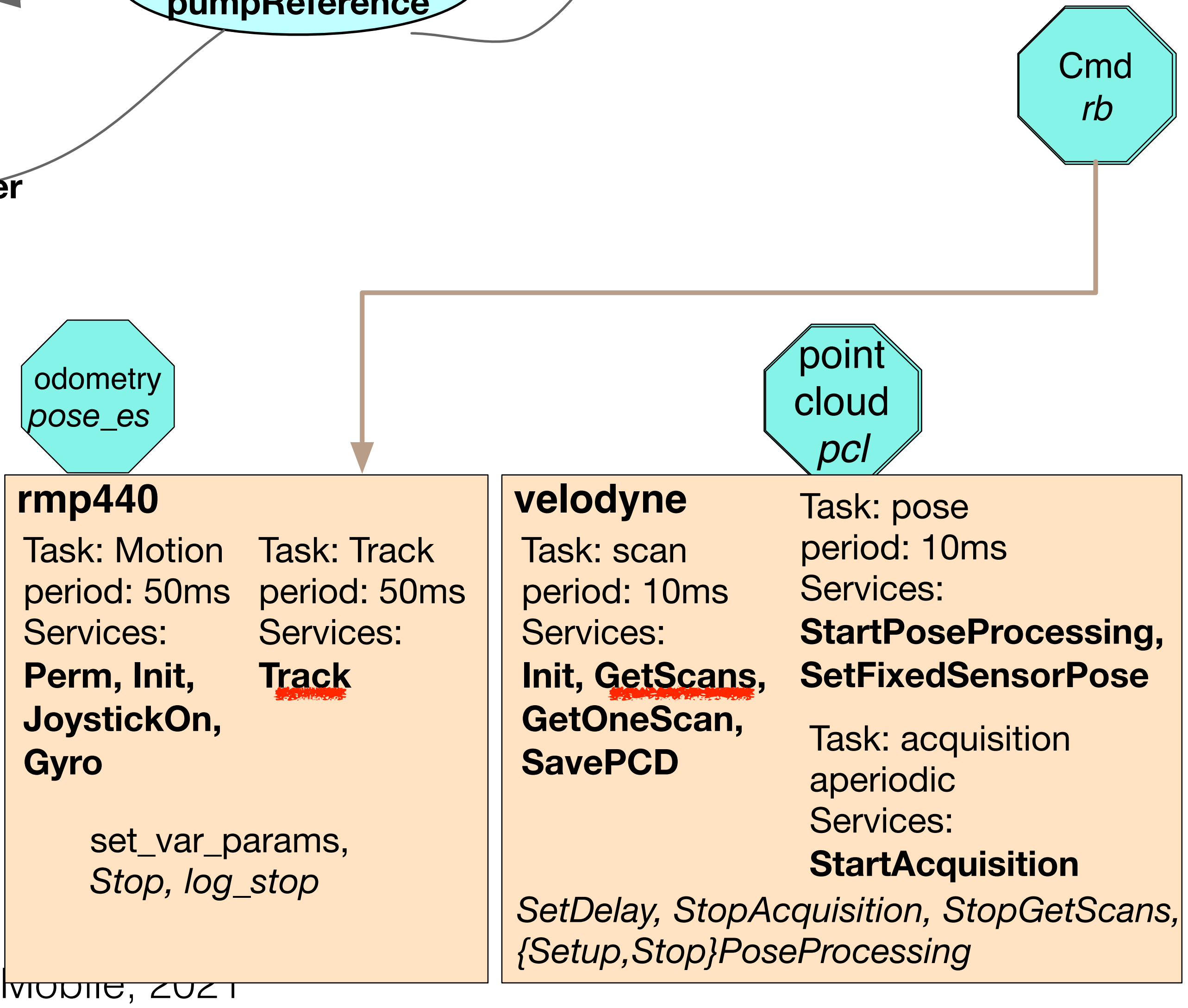
Run Time Verification with HIPPO

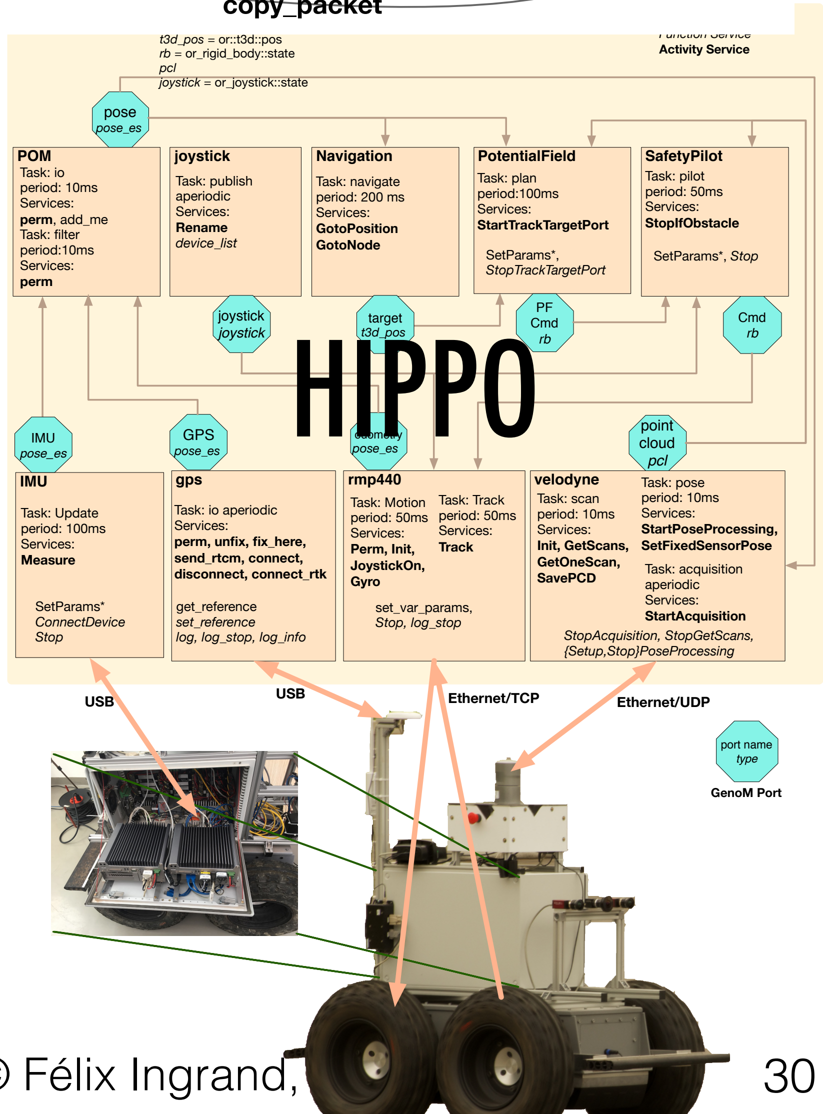
```
activity Track() {
  doc  "Start tracking a reference port";
  validate trackControl(in rmp);
  codel <start>trackStart(inout rs_mode,
                          out max_accel,
                          port in cmd_vel) yield track_main, end;
  codel <track_main>pumpReference(in rs_mode,
                           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, eme                        ord_connected;
  interrupts
};
```

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

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

  codel<start>          velodyneGetScansStart(in acquisition_params) yield copy
  codel<copy_packets>   velodyneGetOneScanCopyPackets(in acquisition_params, ou             er) yield stamp_packets;
  codel<stamp_packets>  velodyneGetOneScanStamp               ition_params,
                                                            data, in timeout) yield pause::stamp_packets, build_scan;
  codel<build_scan>     velodyneGetOneScan                  ams,
                                                          , 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, po                    c_delay)
                                              yield wait;
  codel<wait>           velodyneGetScansWait(in period) yiel              copy_packets;

  interrupts            GetOneScan, SavePCD, GetS

  after               Init;

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





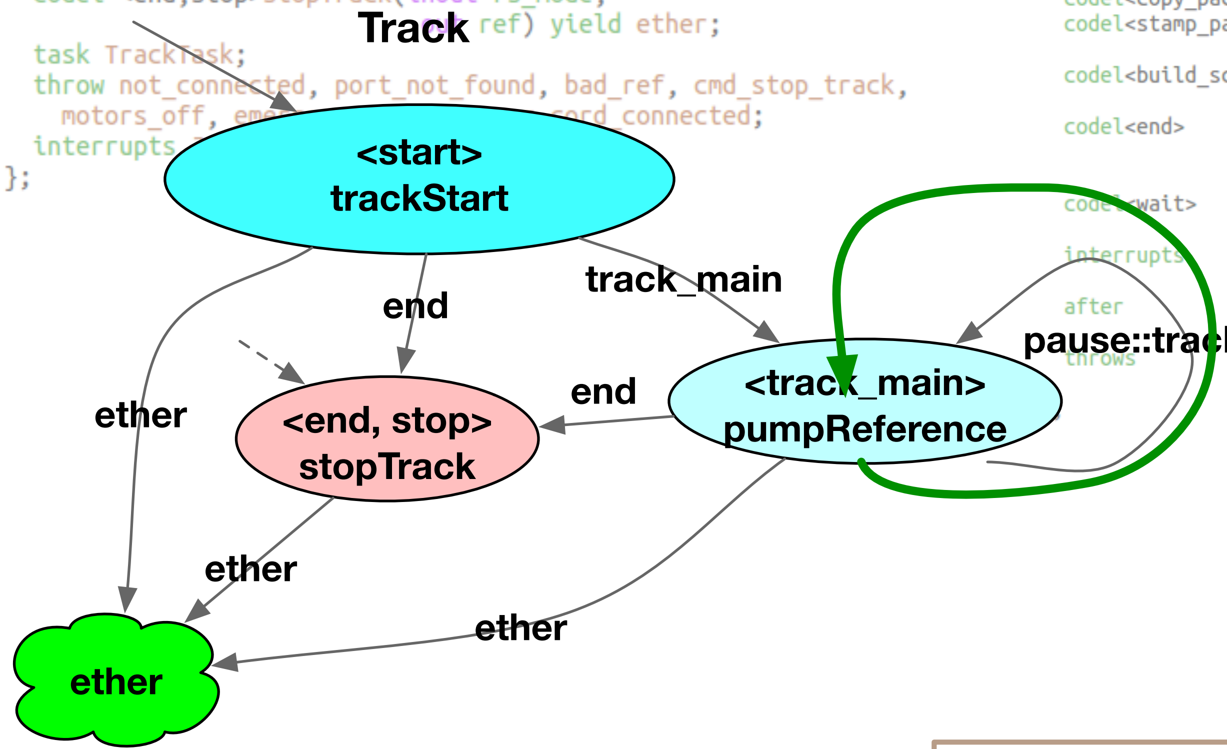


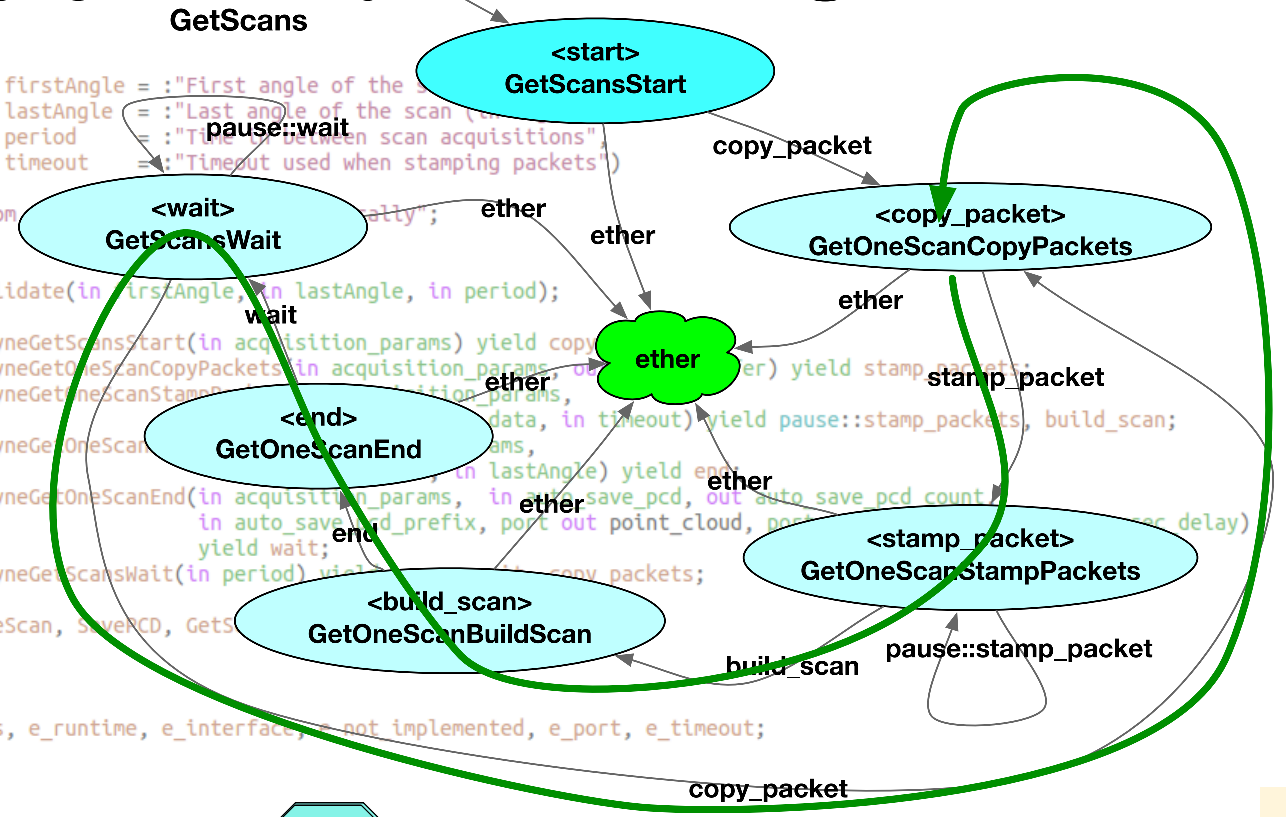


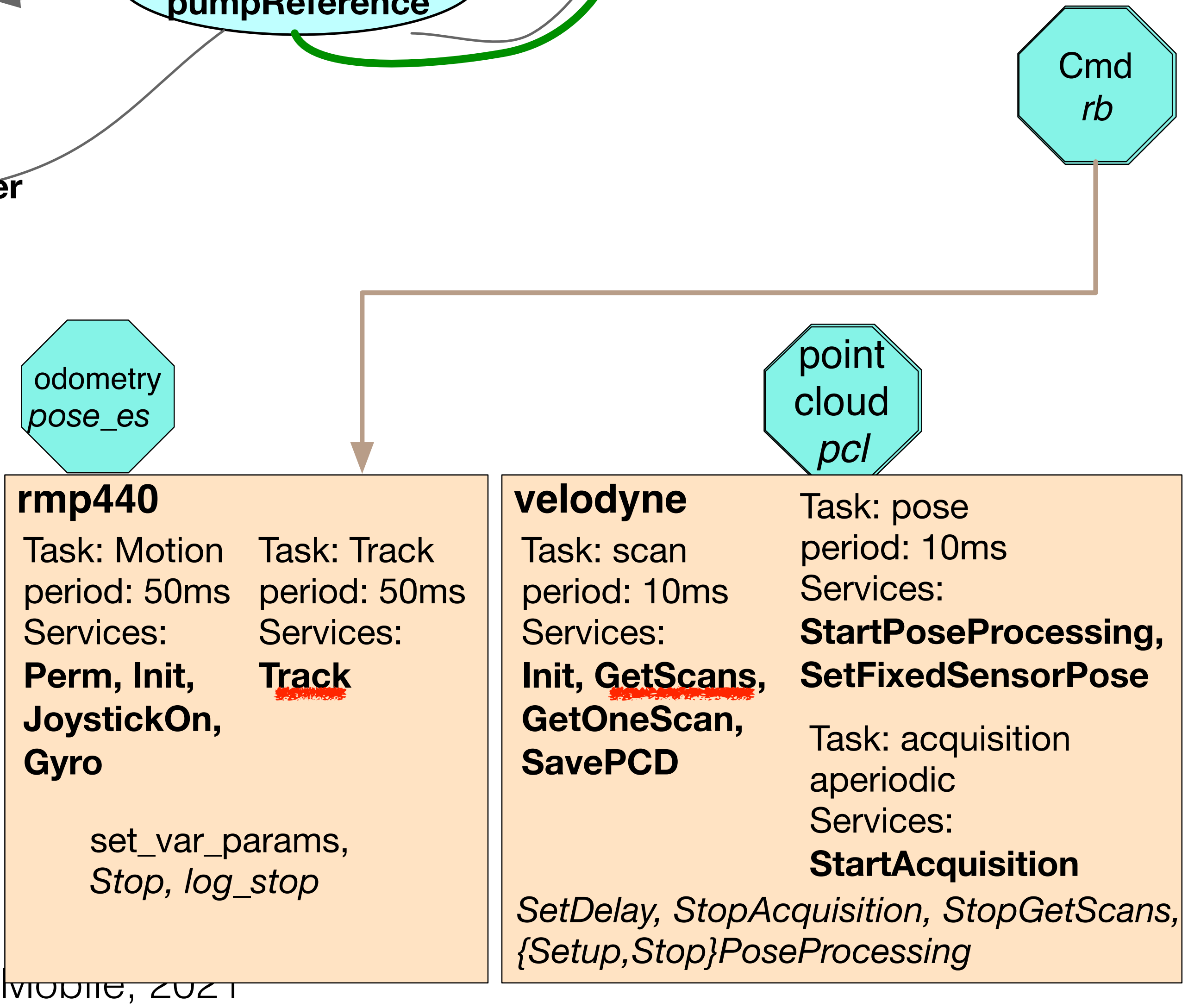
Run Time Verification with HIPPO

```
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, eme                cord_connected;
  interrupts
};
```

```
activity GetScans ( in double firstAngle = :"First angle of the
                    in double lastAngle  = :"Last angle of the scan (
                    in double period     = :"T                n scan acquisitions",
                    in double timeout    = :"Timeout used when stamping packets")
{
  doc  "Acquire full scans from                          ally";
  task scan;

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

  codel<start>         velodyneGetScansStart(in acquisition_params) yield copy
  codel<copy_packets>  velodyneGetOneScanCopyPackets(in acquisition_params, ou          er) yield stamp_packets;
  codel<stamp_packets> velodyneGetOneScanStam                       ition_params,
                                                             data, in timeout) yield pause::stamp_packets, build_scan;
  codel<build_scan>    velodyneGetOneScan                      ams,
                                                            , 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, por                    sec_delay)
                                    yield wait;
  codel<wait>          velodyneGetScansWait(in period) yi                   copy_packets;

  interrupts           GetOneScan, SavePCD, GetS

  after                Init;

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







rmp440

Task: Motion
period: 50ms
Services:
Perm, Init, JoystickOn, Gyro

Task: Track
period: 50ms
Services:
~~Track~~

set_var_params,
Stop, log_stop

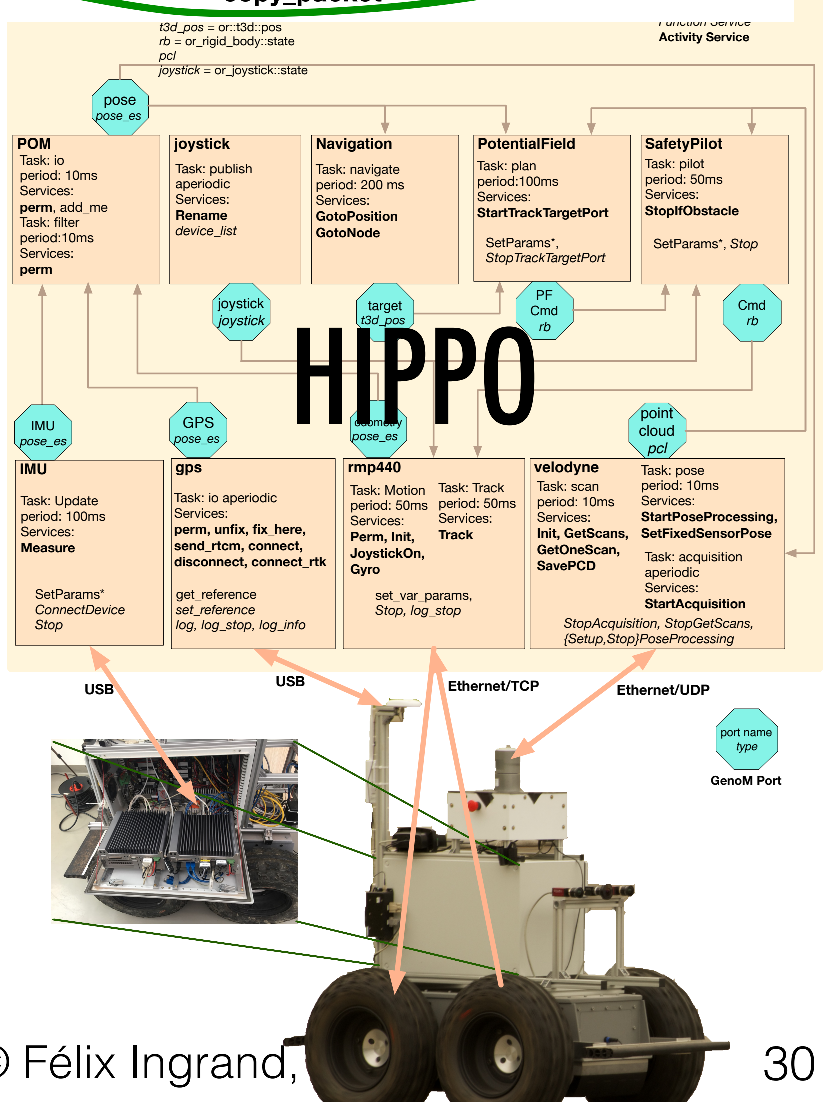
velodyne

Task: scan
period: 10ms
Services:
Init, GetScans, GetOneScan, SavePCD

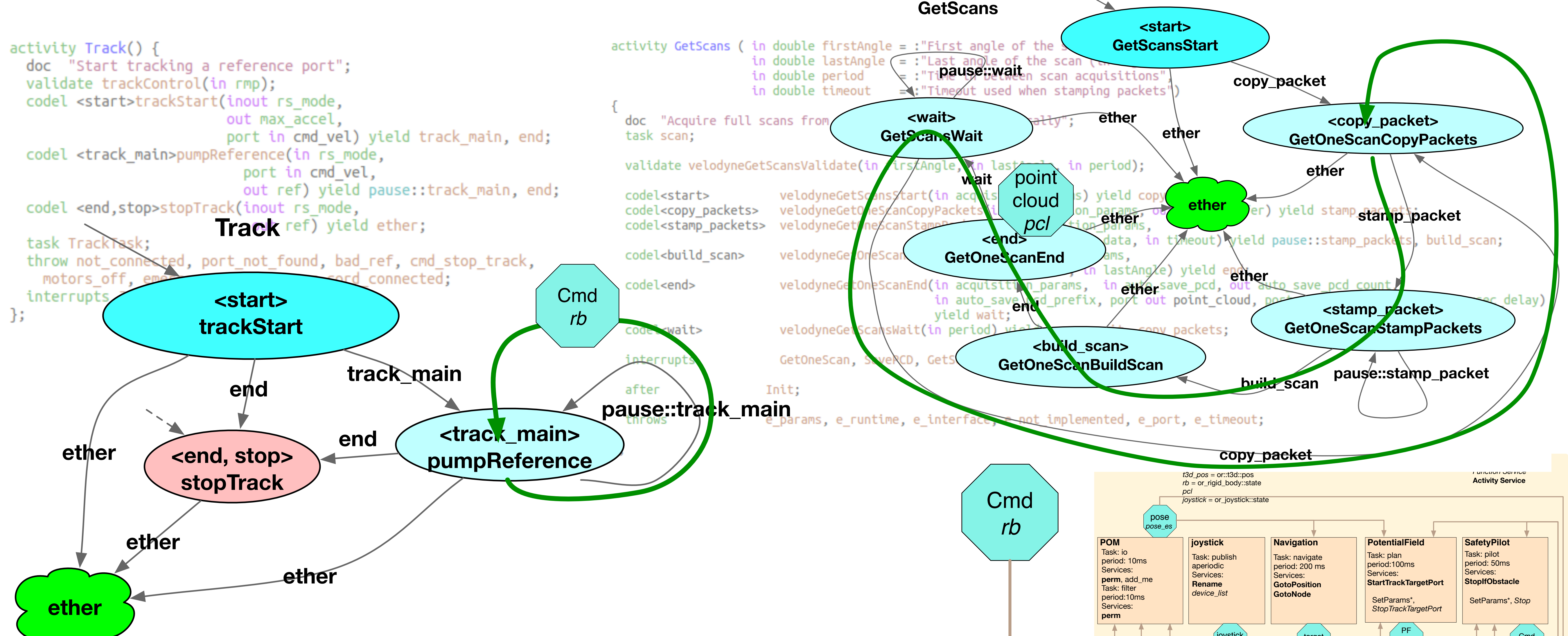
Task: pose
period: 10ms
Services:
StartPoseProcessing, SetFixedSensorPose

Task: acquisition
aperiodic
Services:
StartAcquisition

SetDelay, StopAcquisition, StopGetScans, {Setup,Stop}PoseProcessing



Run Time Verification with HIPPO



```
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, eme                        cord_connected;
  interrupts
};
```

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

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

  codel<start>         velodyneGetScansStart(in acquis            s) yield copy
  codel<copy_packets>  velodyneGetOneScanCopyPackets(i        on_params, ou         er) yield stamp_packets;
  codel<stamp_packets> velodyneGetOneScanStampP            tion_params,
                                                        data, in timeout) yield pause::stamp_packets, build_scan;
  codel<build_scan>    velodyneGetOneScan                 ams,
                                                        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                  ec delay)
                                             yield wait;
  codel<wait>          velodyneGetScansWait(in period) yiel                copy_packets;

  interrupts           GetOneScan, SavePCD, GetS

  after                Init;

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

Cmd
rb

odometry
pose_es

point
cloud
pcl

rmp440

Task: Motion
period: 50ms
Services:
Perm, Init,
JoystickOn,
Gyro

Task: Track
period: 50ms
Services:
Track

set_var_params,
Stop, log_stop

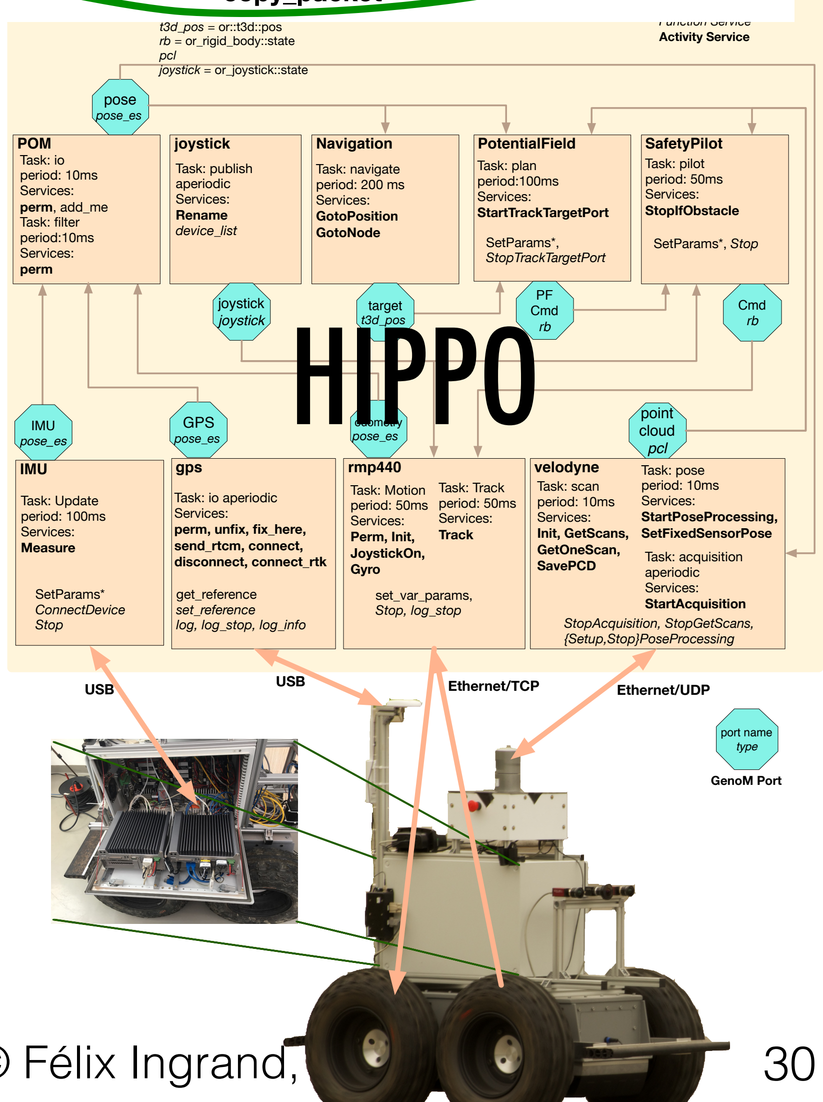
velodyne

Task: scan
period: 10ms
Services:
Init, GetScans,
GetOneScan,
SavePCD

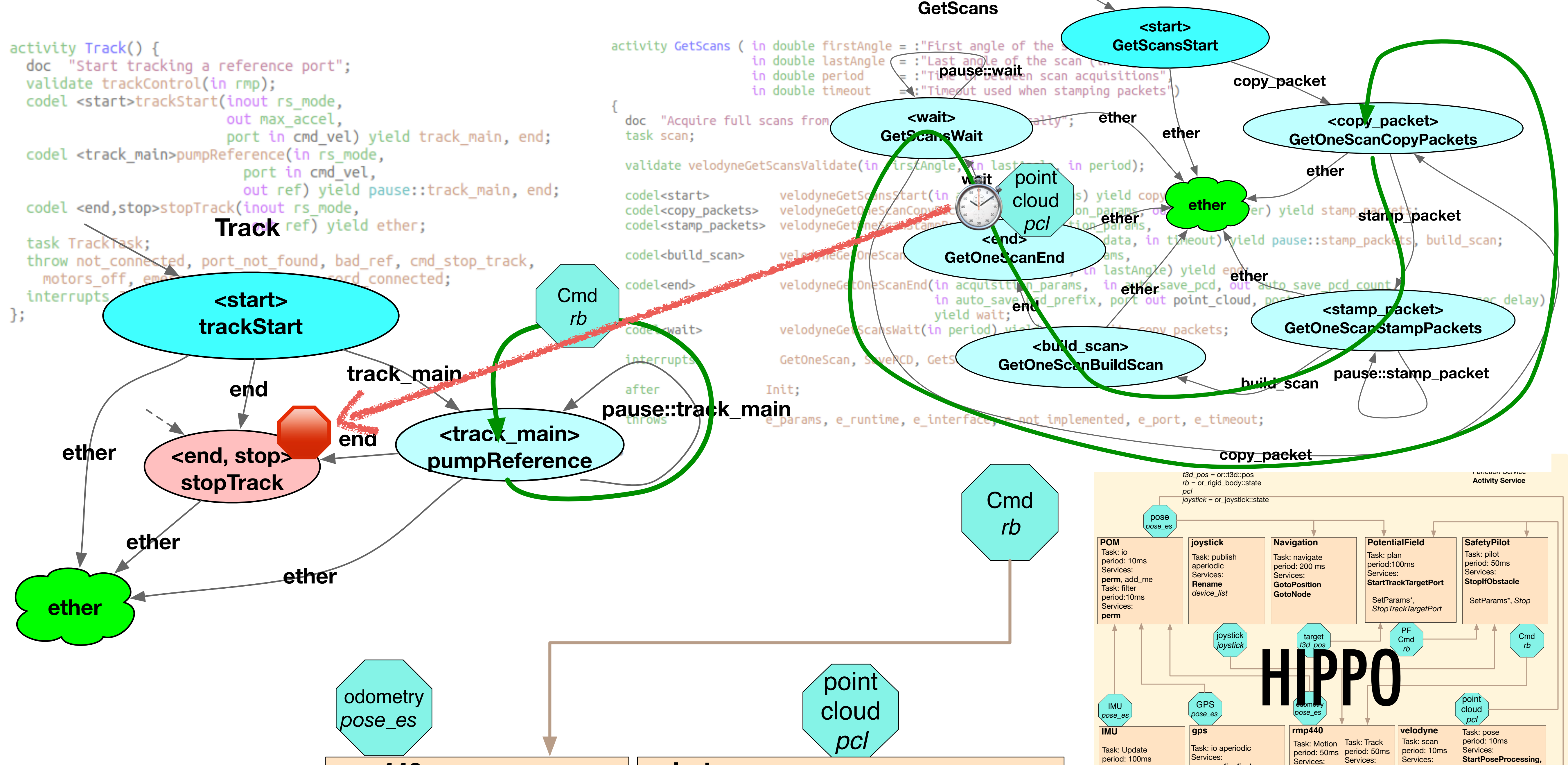
Task: pose
period: 10ms
Services:
StartPoseProcessing,
SetFixedSensorPose

Task: acquisition
aperiodic
Services:
StartAcquisition

SetDelay, StopAcquisition, StopGetScans,
{Setup,Stop}PoseProcessing



Run Time Verification with HIPPO



```
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,
                              ref) yield ether;
  task TrackTask;
  throw not_connected, port_not_found, bad_ref, cmd_stop_track,
    motors_off, eme                    cord_connected;
  interrupts
};
```

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

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

  codel<start>          velodyneGetScansStart(in                s) yield copy
  codel<copy_packets>   velodyneGetOneScanCop                  on_params, ou            er) yield stamp_packets;
  codel<stamp_packets>  velodyneGetOneScanStamp               tion_params,
                                                               data, in timeout) yield pause::stamp_packets, build_scan;
  codel<build_scan>     velodyneGetOneScan                     ams,
                                                              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, por                  sec delay)
                                    yield wait;
  codel<wait>           velodyneGetScansWait(in period) yi                copy_packets;

  interrupts            GetOneScan, SavePCD, GetS

  after                 Init;

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

rmp440

Task: Motion
period: 50ms
Services:
Perm, Init,
JoystickOn,
Gyro

Task: Track
period: 50ms
Services:
Track

set_var_params,
Stop, log_stop

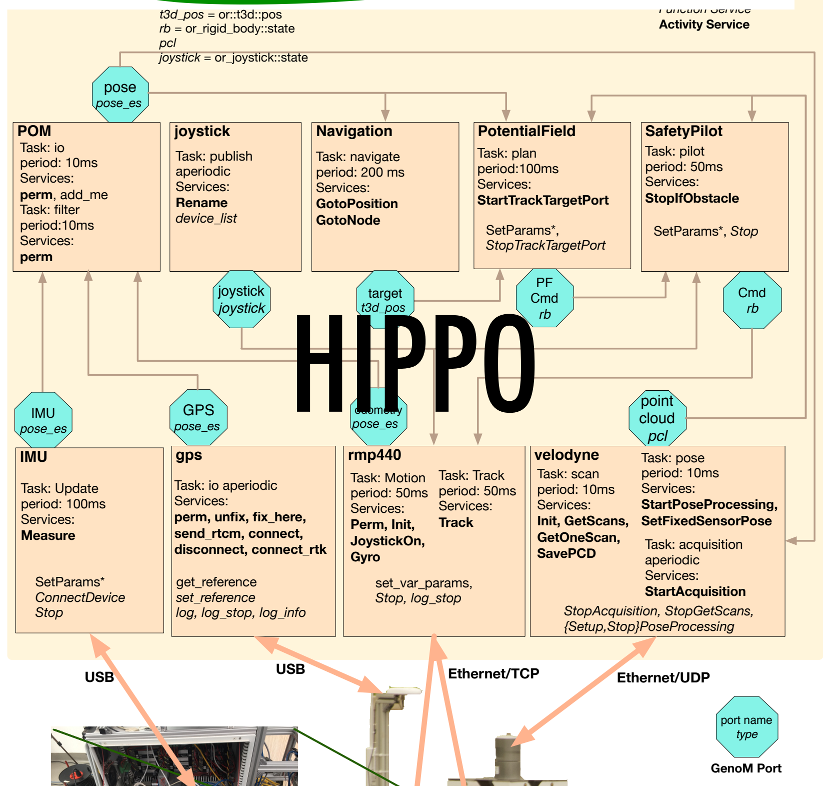
velodyne

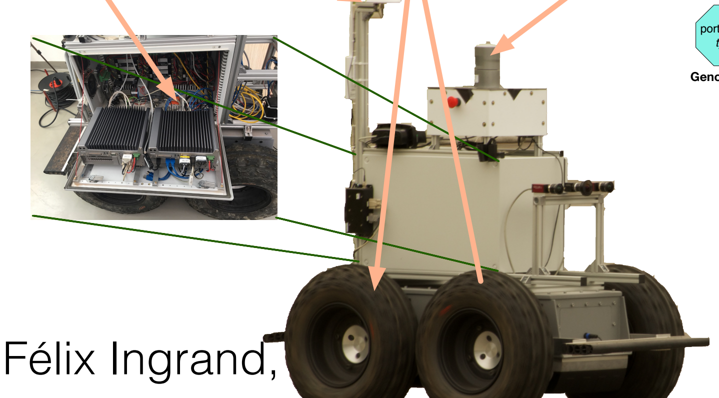
Task: scan
period: 10ms
Services:
Init, GetScans,
GetOneScan,
SavePCD

Task: pose
period: 10ms
Services:
StartPoseProcessing,
SetFixedSensorPose

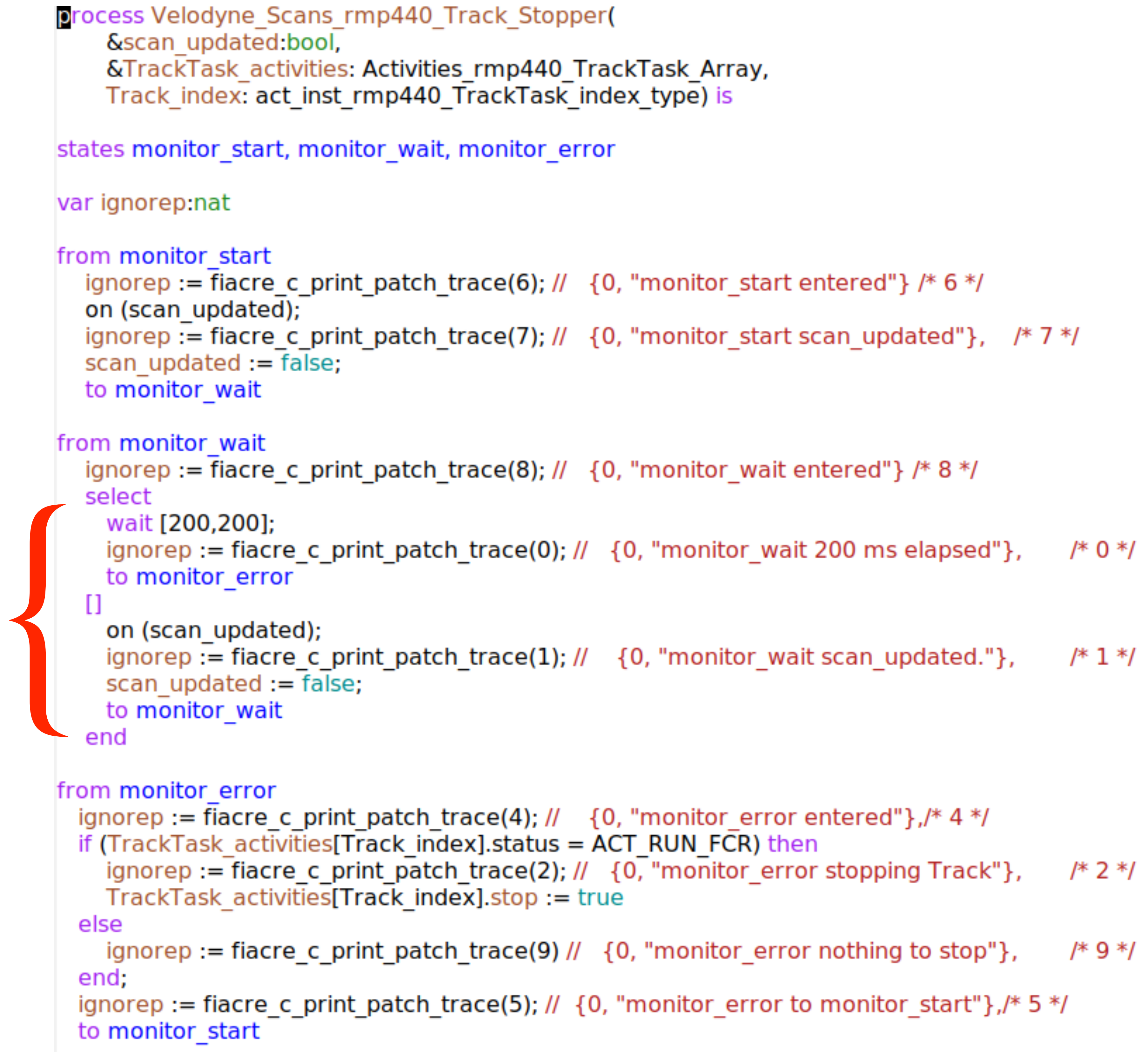
Task: acquisition
aperiodic
Services:
StartAcquisition

SetDelay, StopAcquisition, StopGetScans,
{Setup,Stop}PoseProcessing



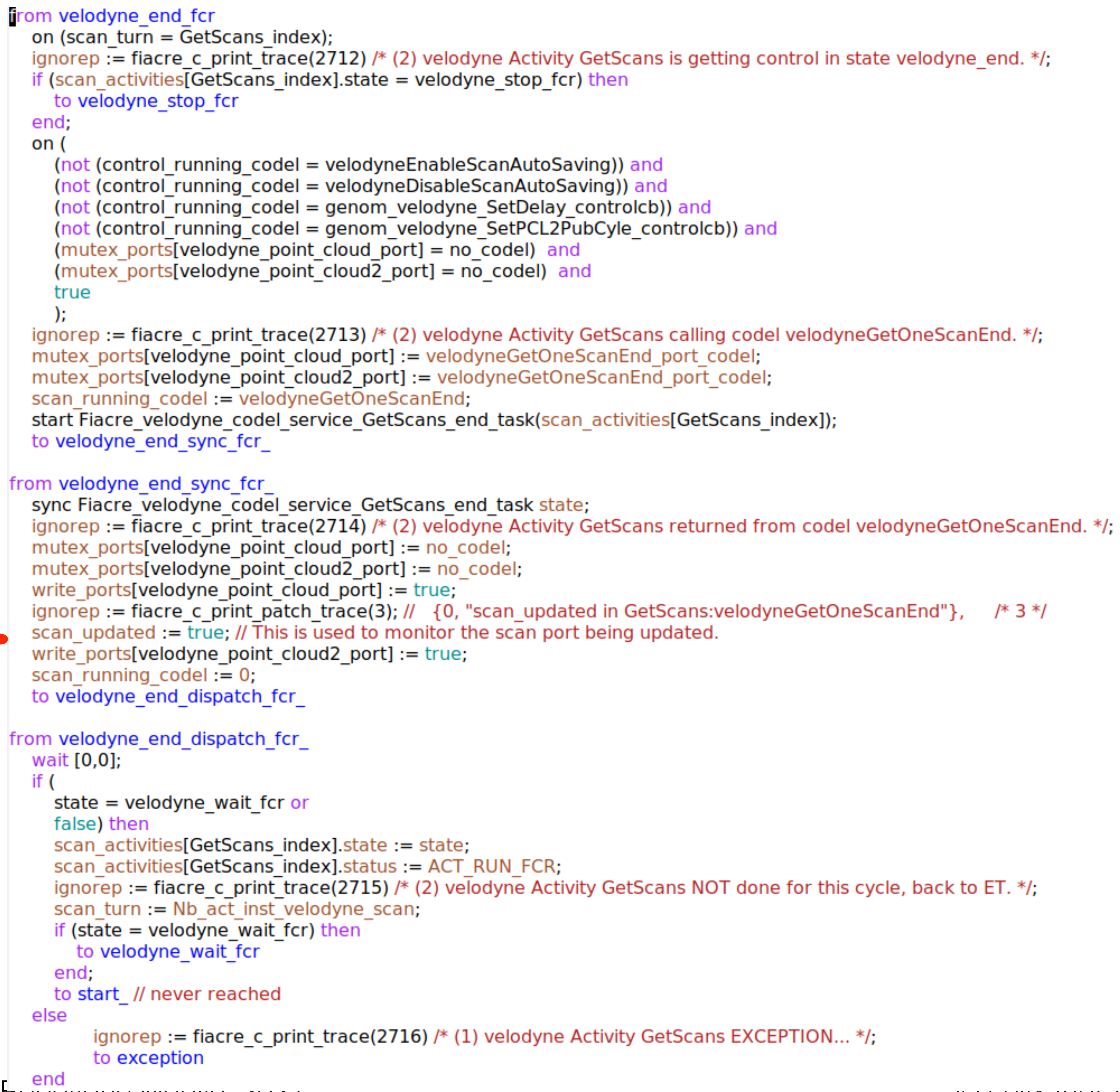


Run Time Verification with Hippo (Minnie)



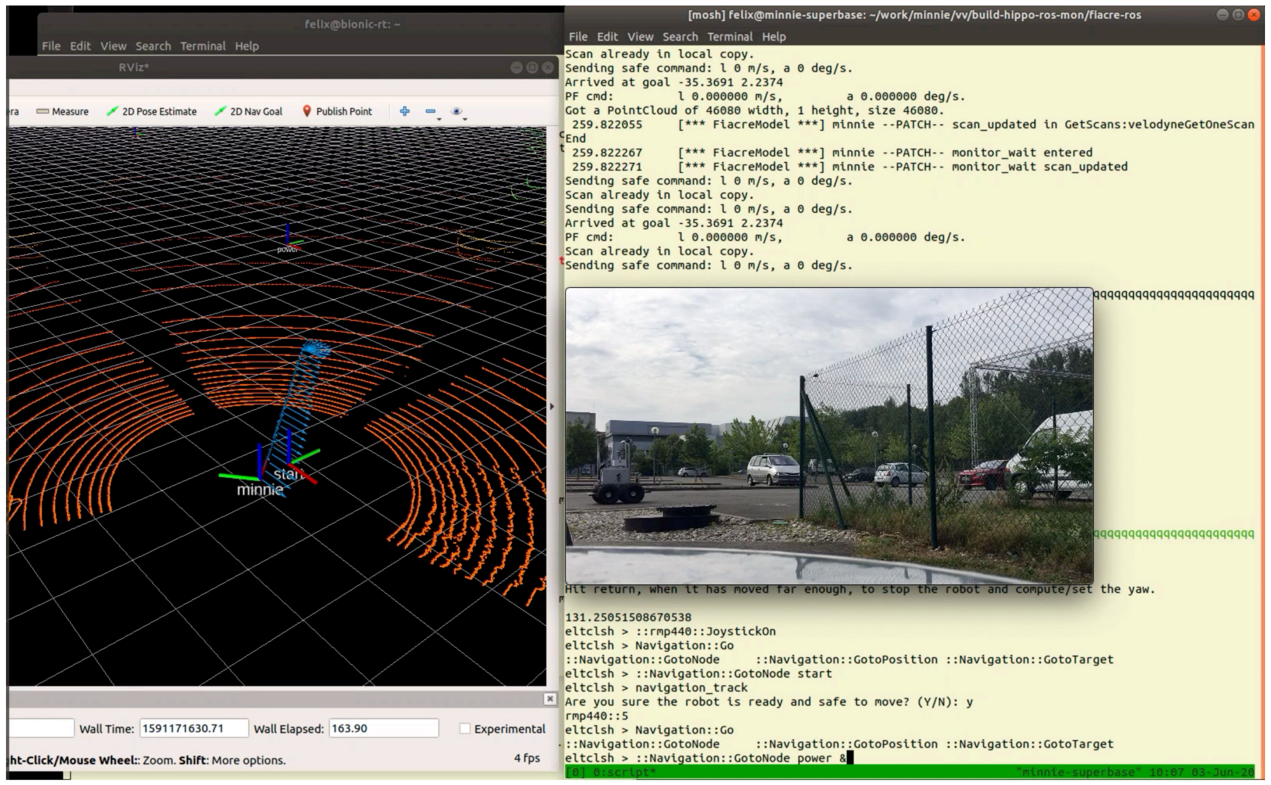
```
process Velodyne_Scans_rmp440_Track_Stopper(
      &scan_updated:bool,
      &TrackTask_activities: Activities_rmp440_TrackTask_Array,
      Track_index: act_inst_rmp440_TrackTask_index_type) is

states monitor_start, monitor_wait, monitor_error

var ignorep:nat

from monitor_start
   ignorep := fiacre_c_print_patch_trace(6); //   {0, "monitor_start entered"} /* 6 */
   on (scan_updated);
   ignorep := fiacre_c_print_patch_trace(7); //   {0, "monitor_start scan_updated"},    /* 7 */
   scan_updated := false;
   to monitor_wait

from monitor_wait
   ignorep := fiacre_c_print_patch_trace(8); //   {0, "monitor_wait entered"} /* 8 */
   select
     wait [200,200];
     ignorep := fiacre_c_print_patch_trace(0); //   {0, "monitor_wait 200 ms elapsed"},      /* 0 */
     to monitor_error
   []
     on (scan_updated);
     ignorep := fiacre_c_print_patch_trace(1); //    {0, "monitor_wait scan_updated."},       /* 1 */
     scan_updated := false;
     to monitor_wait
   end

from monitor_error
  ignorep := fiacre_c_print_patch_trace(4); //    {0, "monitor_error entered"},/* 4 */
  if (TrackTask_activities[Track_index].status = ACT_RUN_FCR) then
     ignorep := fiacre_c_print_patch_trace(2); //  {0, "monitor_error stopping Track"},      /* 2 */
     TrackTask_activities[Track_index].stop := true
  else
     ignorep := fiacre_c_print_patch_trace(9) //  {0, "monitor_error nothing to stop"},      /* 9 */
  end;
  ignorep := fiacre_c_print_patch_trace(5); //  {0, "monitor_error to monitor_start"},/* 5 */
  to monitor_start
```

Run Time Verification with Hippo (Minnie)



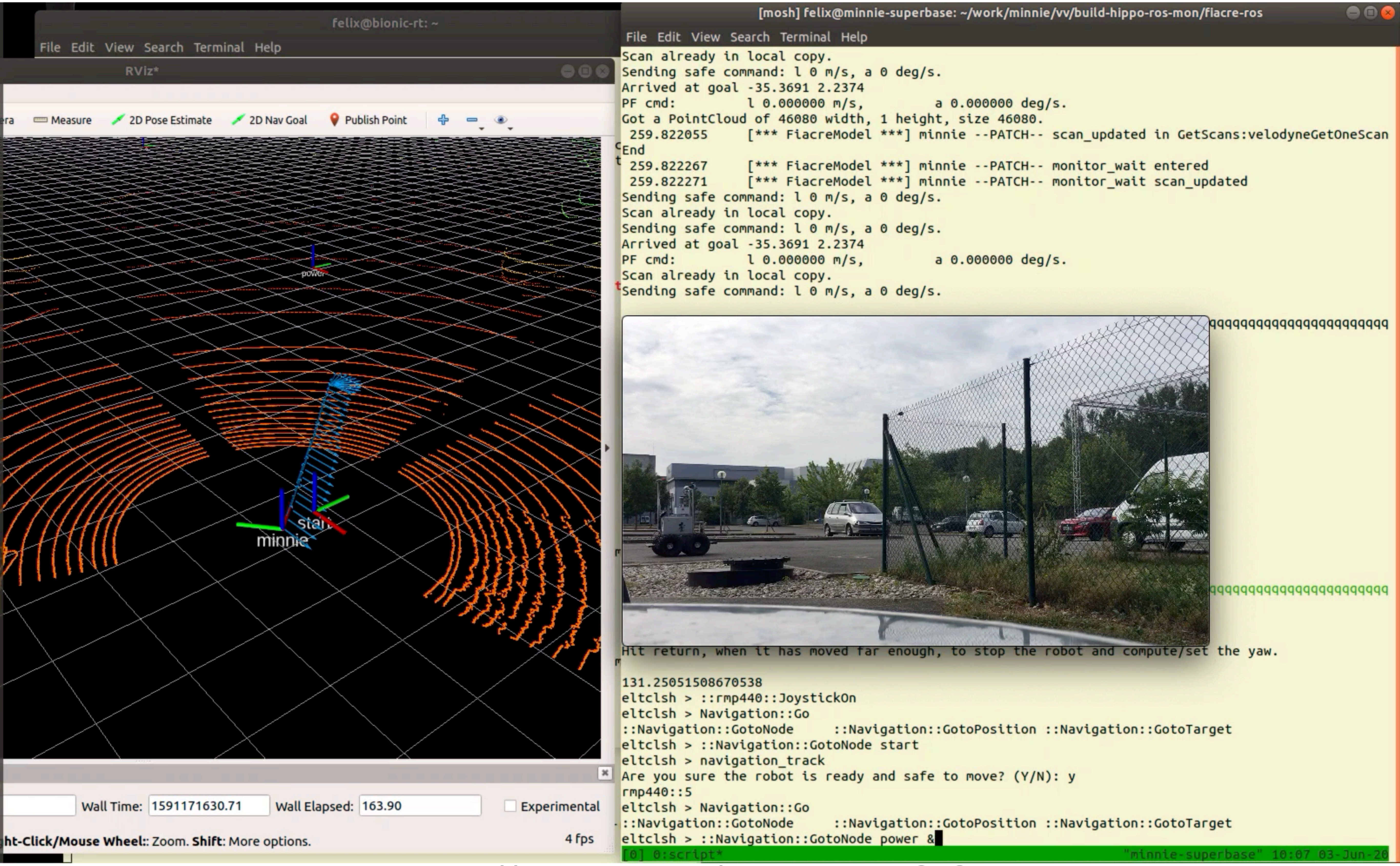
```
from velodyne_end_fcr
    on (scan_turn = GetScans_index);
    ignorep := fiacre_c_print_trace(2712) /* (2) velodyne Activity GetScans is getting control in state velodyne_end. */;
    if (scan_activities[GetScans_index].state = velodyne_stop_fcr) then
        to velodyne_stop_fcr
    end;
    on (
        (not (control_running_codel = velodyneEnableScanAutoSaving)) and
        (not (control_running_codel = velodyneDisableScanAutoSaving)) and
        (not (control_running_codel = genom_velodyne_SetDelay_controlcb)) and
        (not (control_running_codel = genom_velodyne_SetPCL2PubCyle_controlcb)) and
        (mutex_ports[velodyne_point_cloud_port] = no_codel)  and
        (mutex_ports[velodyne_point_cloud2_port] = no_codel)  and
        true
        );
    ignorep := fiacre_c_print_trace(2713) /* (2) velodyne Activity GetScans calling codel velodyneGetOneScanEnd. */;
    mutex_ports[velodyne_point_cloud_port] := velodyneGetOneScanEnd_port_codel;
    mutex_ports[velodyne_point_cloud2_port] := velodyneGetOneScanEnd_port_codel;
    scan_running_codel := velodyneGetOneScanEnd;
    start Fiacre_velodyne_codel_service_GetScans_end_task(scan_activities[GetScans_index]);
    to velodyne_end_sync_fcr_

from velodyne_end_sync_fcr_
    sync Fiacre_velodyne_codel_service_GetScans_end_task state;
    ignorep := fiacre_c_print_trace(2714) /* (2) velodyne Activity GetScans returned from codel velodyneGetOneScanEnd. */;
    mutex_ports[velodyne_point_cloud_port] := no_codel;
    mutex_ports[velodyne_point_cloud2_port] := no_codel;
    write_ports[velodyne_point_cloud_port] := true;
    ignorep := fiacre_c_print_patch_trace(3); //   {0, "scan_updated in GetScans:velodyneGetOneScanEnd"},     /* 3 */
    scan_updated := true; // This is used to monitor the scan port being updated.
    write_ports[velodyne_point_cloud2_port] := true;
    scan_running_codel := 0;
    to velodyne_end_dispatch_fcr_

from velodyne_end_dispatch_fcr_
    wait [0,0];
    if (
        state = velodyne_wait_fcr or
        false) then
        scan_activities[GetScans_index].state := state;
        scan_activities[GetScans_index].status := ACT_RUN_FCR;
        ignorep := fiacre_c_print_trace(2715) /* (2) velodyne Activity GetScans NOT done for this cycle, back to ET. */;
        scan_turn := Nb_act_inst_velodyne_scan;
        if (state = velodyne_wait_fcr) then
            to velodyne_wait_fcr
        end;
        to start_ // never reached
    else
            ignorep := fiacre_c_print_trace(2716) /* (1) velodyne Activity GetScans EXCEPTION... */;
            to exception
    end
```

Video


felix@bionic-rt: ~
File Edit View Search Terminal Help
RViz*
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
start
minnie
Wall Time: 1591171630.71
Wall Elapsed: 163.90
Experimental
ht-Click/Mouse Wheel:: Zoom. Shift: More options.
4 fps
[mosh] felix@minnie-superbase: ~/work/minnie/vv/build-hippo-ros-mon/fiacre-ros
File Edit View Search Terminal Help

```
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.
Arrived at goal -35.3691 2.2374
PF cmd:         l 0.000000 m/s,         a 0.000000 deg/s.
Got a PointCloud of 46080 width, 1 height, size 46080.
 259.822055     [*** FiacreModel ***] minnie --PATCH-- scan_updated in GetScans:velodyneGetOneScan
End
 259.822267     [*** FiacreModel ***] minnie --PATCH-- monitor_wait entered
 259.822271     [*** FiacreModel ***] minnie --PATCH-- monitor_wait scan_updated
Sending safe command: l 0 m/s, a 0 deg/s.
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.
Arrived at goal -35.3691 2.2374
PF cmd:         l 0.000000 m/s,         a 0.000000 deg/s.
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.

                                                              qqqqqqqqqqqqqqqqqqqqqq

                                                              qqqqqqqqqqqqqqqqqqqqqq

Hit return, when it has moved far enough, to stop the robot and compute/set the yaw.

131.25051508670538
eltclsh > ::rmp440::JoystickOn
eltclsh > Navigation::Go
::Navigation::GotoNode     ::Navigation::GotoPosition ::Navigation::GotoTarget
eltclsh > ::Navigation::GotoNode start
eltclsh > navigation_track
Are you sure the robot is ready and safe to move? (Y/N): y
rmp440::5
eltclsh > Navigation::Go
::Navigation::GotoNode     ::Navigation::GotoPosition ::Navigation::GotoTarget
eltclsh > ::Navigation::GotoNode power &
[0] 0:script*                                          "minnie-superbase" 10:07 03-Jun-20
```

https://youtu.be/vXZiW5tOG54


felix@bionic-rt: ~
File Edit View Search Terminal Help
RViz*
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
power
start
minnie
Wall Time: 1591171630.71
Wall Elapsed: 163.90
Experimental
Right-Click/Mouse Wheel:: Zoom. Shift: More options.
4 fps
[mosh] felix@minnie-superbase: ~/work/minnie/vv/build-hippo-ros-mon/fiacre-ros
File Edit View Search Terminal Help

```
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.
Arrived at goal -35.3691 2.2374
PF cmd:         l 0.000000 m/s,        a 0.000000 deg/s.
Got a PointCloud of 46080 width, 1 height, size 46080.
 259.822055     [*** FiacreModel ***] minnie --PATCH-- scan_updated in GetScans:velodyneGetOneScan
End
 259.822267     [*** FiacreModel ***] minnie --PATCH-- monitor_wait entered
 259.822271     [*** FiacreModel ***] minnie --PATCH-- monitor_wait scan_updated
Sending safe command: l 0 m/s, a 0 deg/s.
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.
Arrived at goal -35.3691 2.2374
PF cmd:         l 0.000000 m/s,        a 0.000000 deg/s.
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.

Hit return, when it has moved far enough, to stop the robot and compute/set the yaw.

131.25051508670538
eltclsh > ::rmp440::JoystickOn
eltclsh > Navigation::Go
::Navigation::GotoNode     ::Navigation::GotoPosition ::Navigation::GotoTarget
eltclsh > ::Navigation::GotoNode start
eltclsh > navigation_track
Are you sure the robot is ready and safe to move? (Y/N): y
rmp440::5
eltclsh > Navigation::Go
::Navigation::GotoNode     ::Navigation::GotoPosition ::Navigation::GotoTarget
eltclsh > ::Navigation::GotoNode power &
[0] 0:script*                                            "minnie-superbase" 10:07 03-Jun-20
```

https://youtu.be/vXZiW5tOG54

Take home messages

- GenoM offers a high level specification language…
- … along a template mechanism
- to automatically synthesize:
  - components for various MW (e.g. ros-comm, pocolibs)
  - but also their equivalent formal models (e.g. FIACRE)
- which can be used for offline verification (TINA)
- and online runtime verification (Hippo) with added formal monitor
- Can be used for others robotic application (e.g. drones)

Software and Papers

Useful links:

GenoM3 https://git.openrobots.org/projects/genom3

Template GenoM3-Pocolibs https://git.openrobots.org/projects/genom3-pocolibs

Template GenoM3-ROS https://git.openrobots.org/projects/genom3-ros

Fiacre http://projects.laas.fr/fiacre/

Tina http://projects.laas.fr/tina/

Hippo https://redmine.laas.fr/projects/genom3-fiacre-template/gollum/hippo

Template GenoM3 Fiacre (ROS et pocolibs) https://redmine.laas.fr/projects/genom3-fiacre-template/gollum/index

Drone experiment https://redmine.laas.fr/projects/drone-v-v/gollum/index
containerized (requires gazebo client on the host): https://hub.docker.com/repository/docker/felixfi/hippodrone

Expérimentation sur Minnie RMP440 https://redmine.laas.fr/projects/minnie/gollum/fiacre

Paper on V&V in robotic https://hal.laas.fr/hal-02927311

Paper on Fiacre/Hippo/GenoM3 https://hal.laas.fr/hal-03017661