

# Hippo and GenoM

*On the Advantages of Using a **Formal-Model Execution Engine**  
to Control and Verify **Critical Robotic System***

**Silvano DAL ZILIO**, Pierre-Emmanuel HLADIK, Félix INGRAND  
and also Anthony MALLET, Reyyan TEKKIN, Mohamed FOUGHALI, Robin VINCELLE, ...



# Robotic Software

# Formal Methods

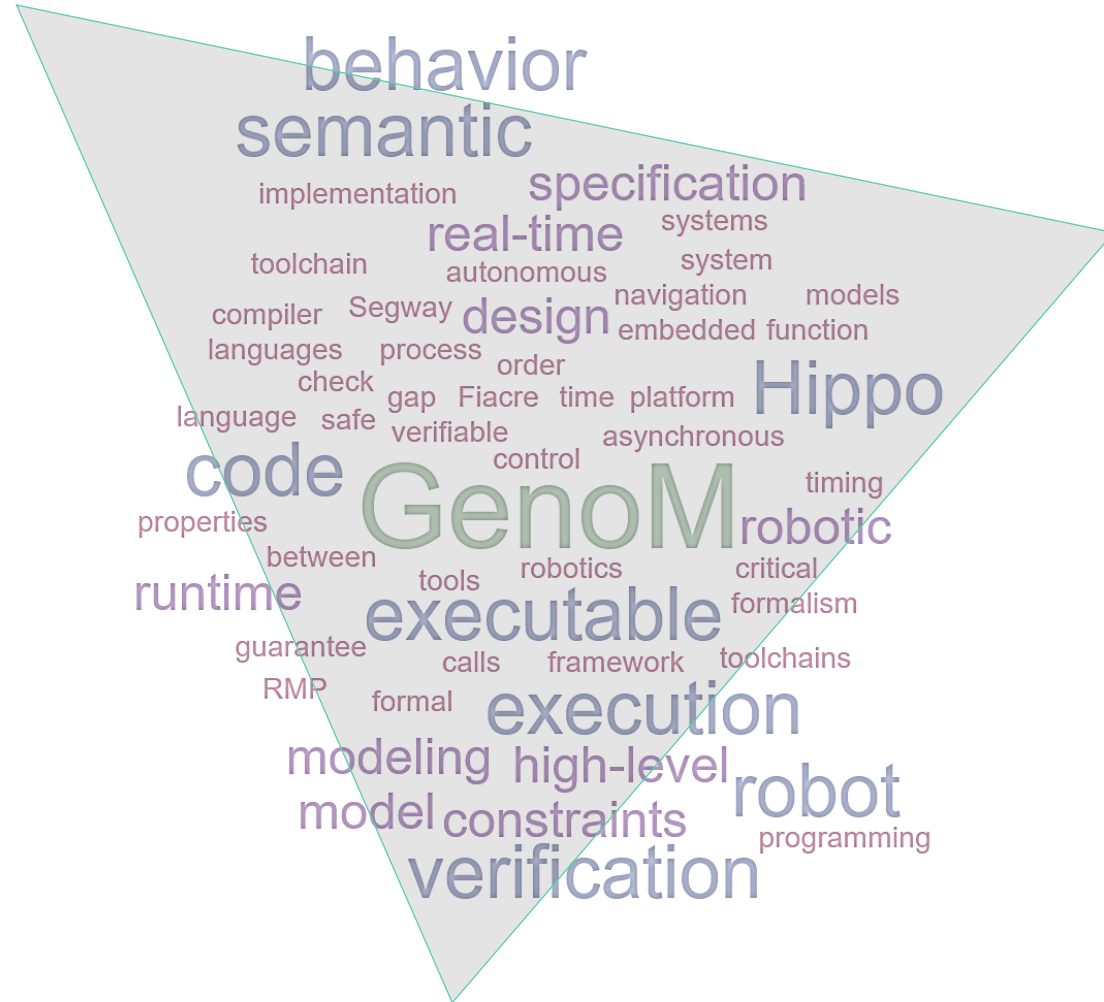
ROS

**GenoM**

Pocolibs

**Programming  
Languages**

**Functional Architecture; Runtime Systems**



Time Petri nets  
Model-checking  
Temporal logic (LTL, CTL)

Tina

Process Calculi  
Data and Types

Fiacre

**Hippo**

**Schedulability**

# A High-Level Overview of GenoM

Formal Methods 101

Formal-Model Execution Engine

Experimentations

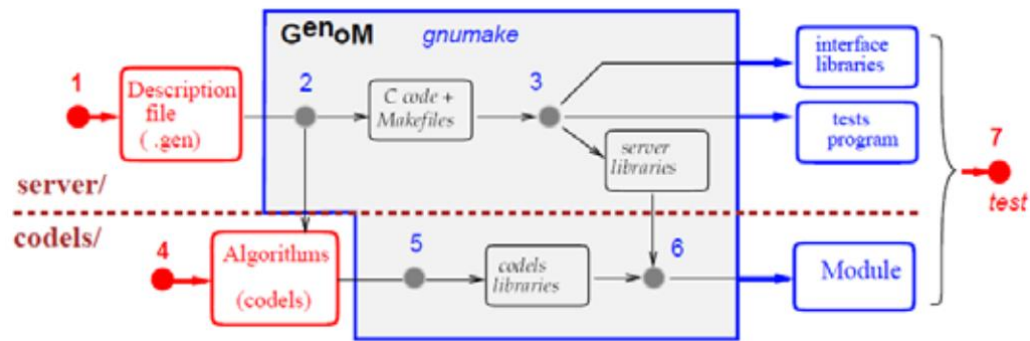
# GenoM: Generator of Modules

Answer the question: how do you program these ?

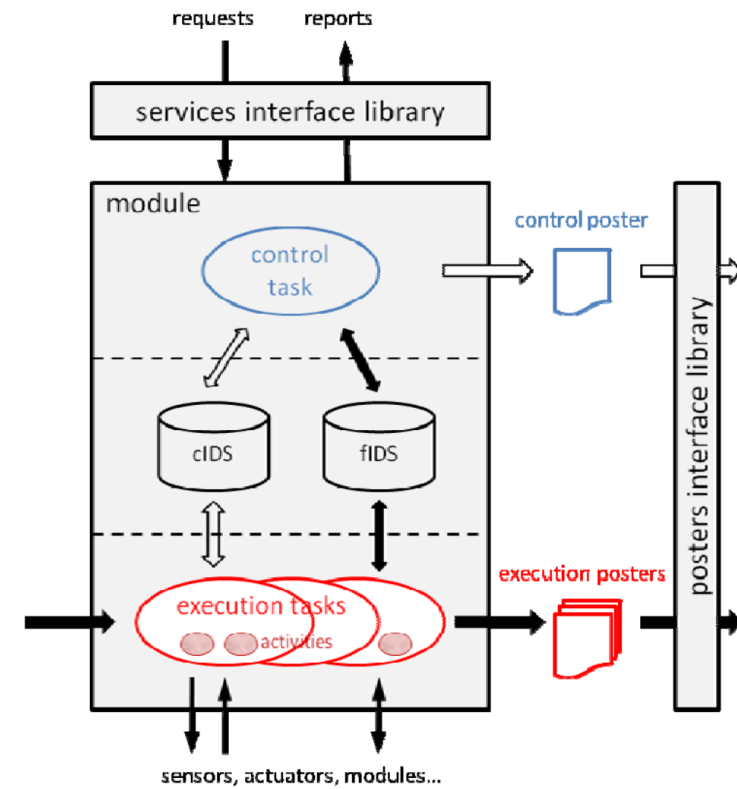
Used and developed by the RIS team, at LAAS, for the last 25 years



# GenoM (architecture)

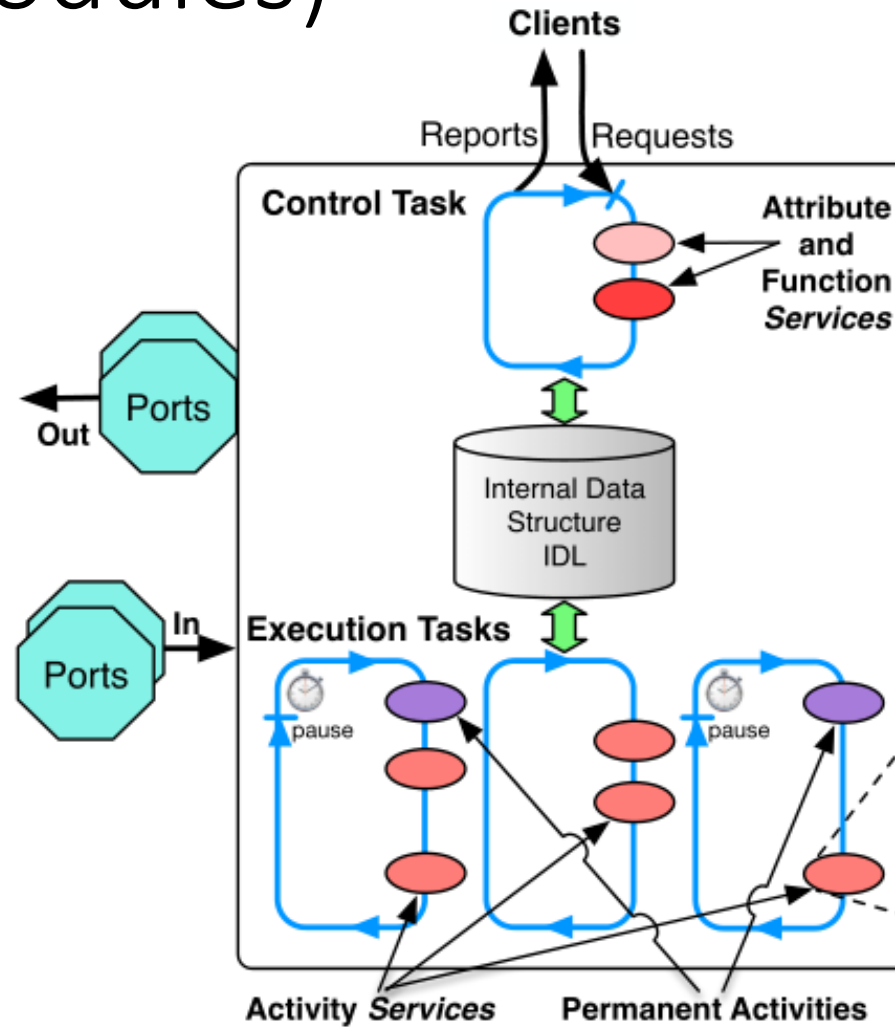
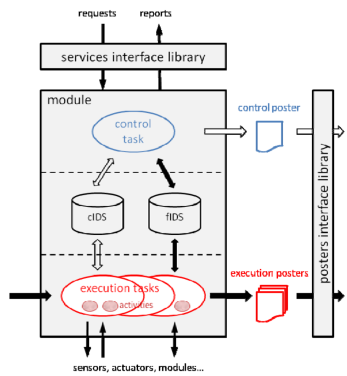


Development cycle



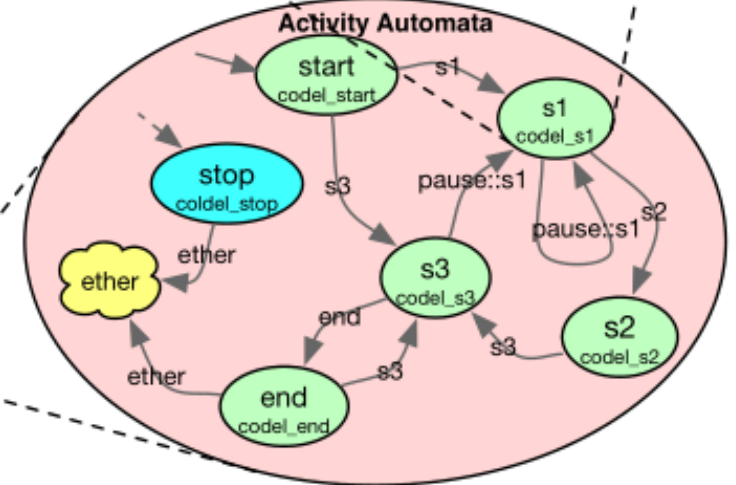
Module structure

# GenoM (modules)



```

genom_event s1
code1_s1(port in p1, port out p2, ...
         ids in i1, ids out i2, ...
         local in l1, local out l2, ...)
{
  if ... {
    while - { C/C++ code
    }
  } else {
    return pause::s1; WCET
  }
  for ( ..., ..., ... ) { - }
  return s2;
}
    
```



# GenoM (code)



<b>velodyne</b>	Task: pose
Task: scan	period: 10ms
period: 10ms	Services:
Services:	<b>StartPoseProcessing,</b>
<b>Init, GetScans,</b>	<b>SetFixedSensorPose</b>
<b>GetOneScan,</b>	Task: acquisition
<b>SavePCD</b>	aperiodic
	Services:
	<b>StartAcquisition</b>
	<i>SetDelay, StopAcquisition, StopGetScans,</i>
	<i>{Setup, Stop}PoseProcessing</i>

```
#pragma require "openrobots2-idl >= 2.0"
#pragma require "minnie-idl"

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

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

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

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

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

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

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

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

    throws e_mem, e_grabber;
  };

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

# GenoM (code)

```

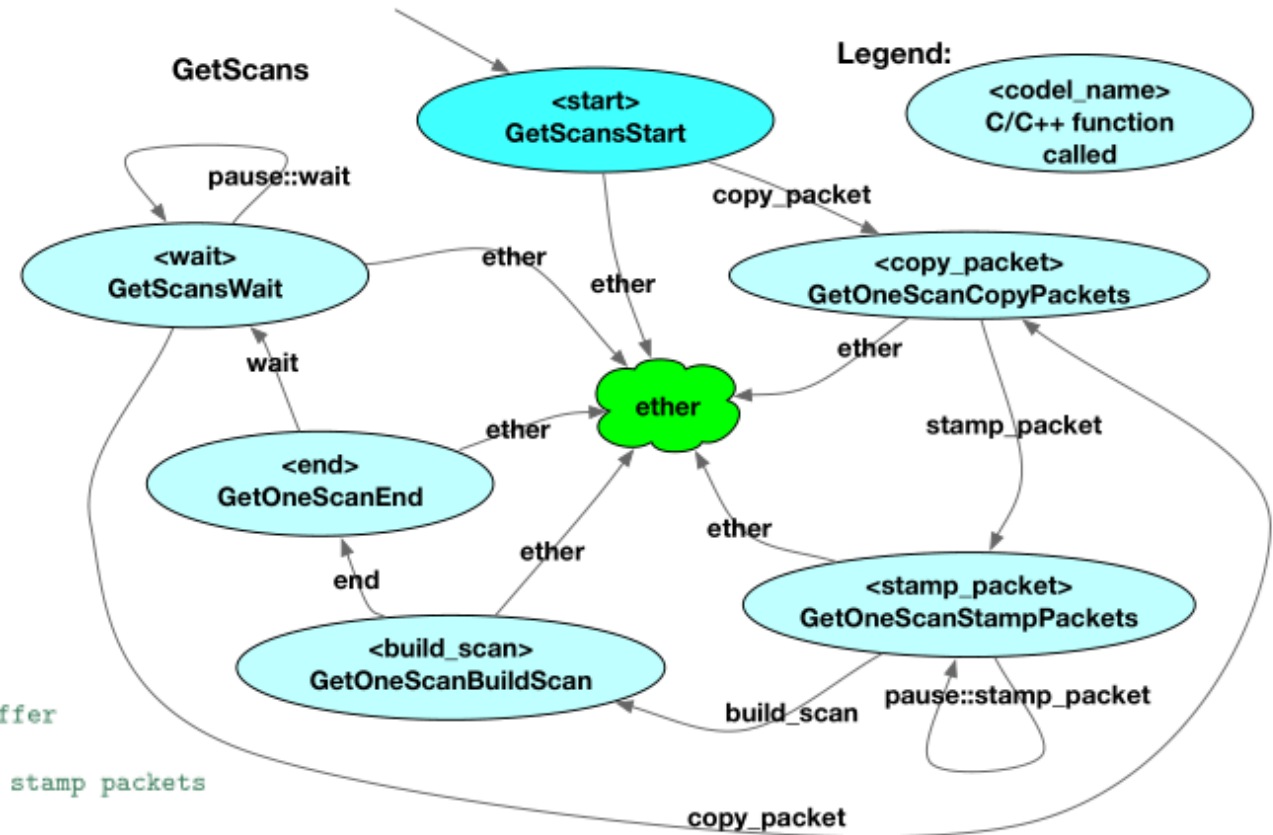
activity GetScans(
    in double firstAngle = : "First angle of the scan (in degrees)",
    in double lastAngle = : "Last angle of the scan (in degrees)",
    in double period = : "Time in between two scans",
    in double timeout = : "Timeout used when stamping packets")
{
    doc "Acquire full scans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

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

    codel <start> GetScansStart(in acquisition_params)
        yield copy_packets;
    codel <copy_packets> GetOneScanCopyPackets(in acquisition_params,
        inout scan_buffer) // get packets from acquisition buffer
        yield stamp_packets;
    codel <stamp_packets> GetOneScanStampPackets(in acquisition_params, // stamp packets
        inout pose_data, in timeout) // with the proper pose
        yield pause::stamp_packets, build_scan; // pause:: if pose not available
    codel <build_scan> GetOneScanBuildScan(in acquisition_params,
        in firstAngle, in lastAngle) // build scan repositioning
        yield end; // individual packet in the first pose.
    codel <end> GetOneScanEnd(in acquisition_params,
        port out point_cloud, inout usec_delay) //publish the scan in the
        yield wait; // point_cloud port. usec_delay is for fault injection.
    codel <wait> GetScansWait(in period) // wait next user defined scan period
        yield pause::wait, copy_packets; // then loop back.

    interrupts GetOneScan, SavePCD, GetScans;
};

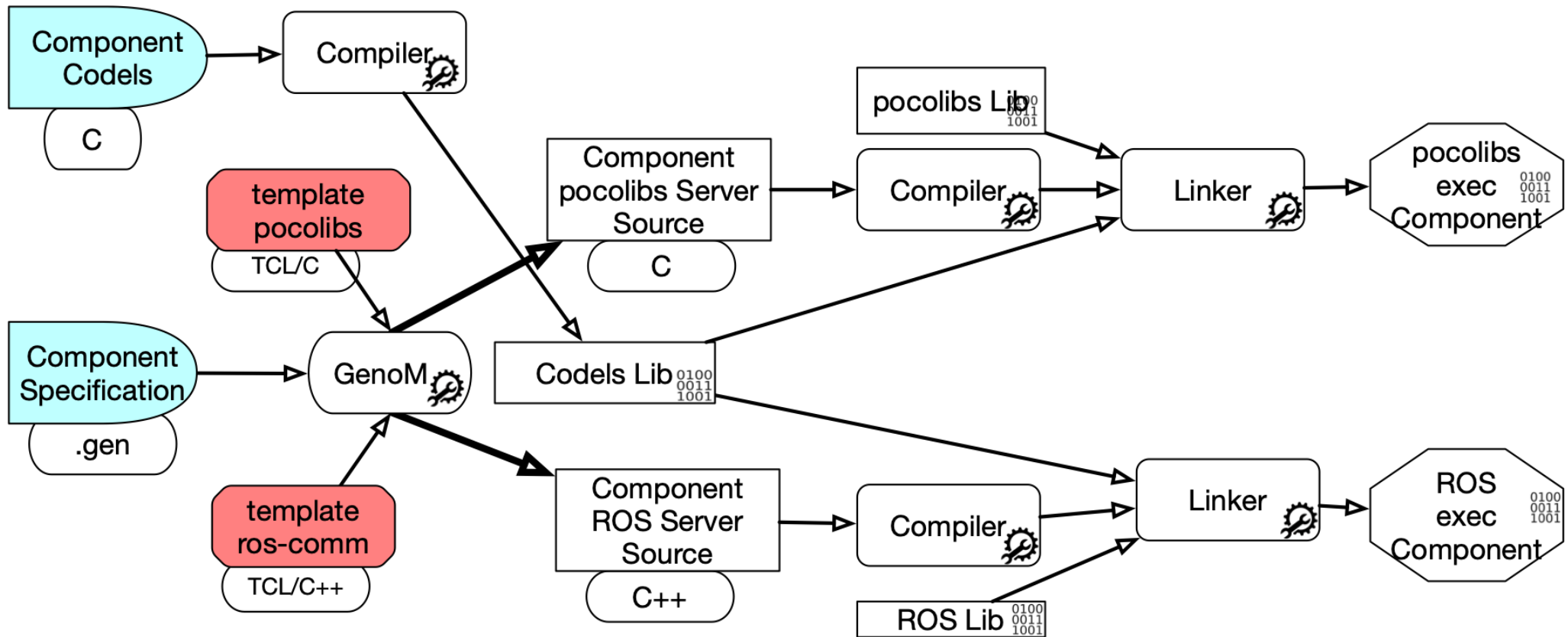
```



scan task of the velodyne module

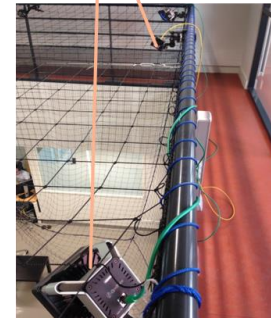
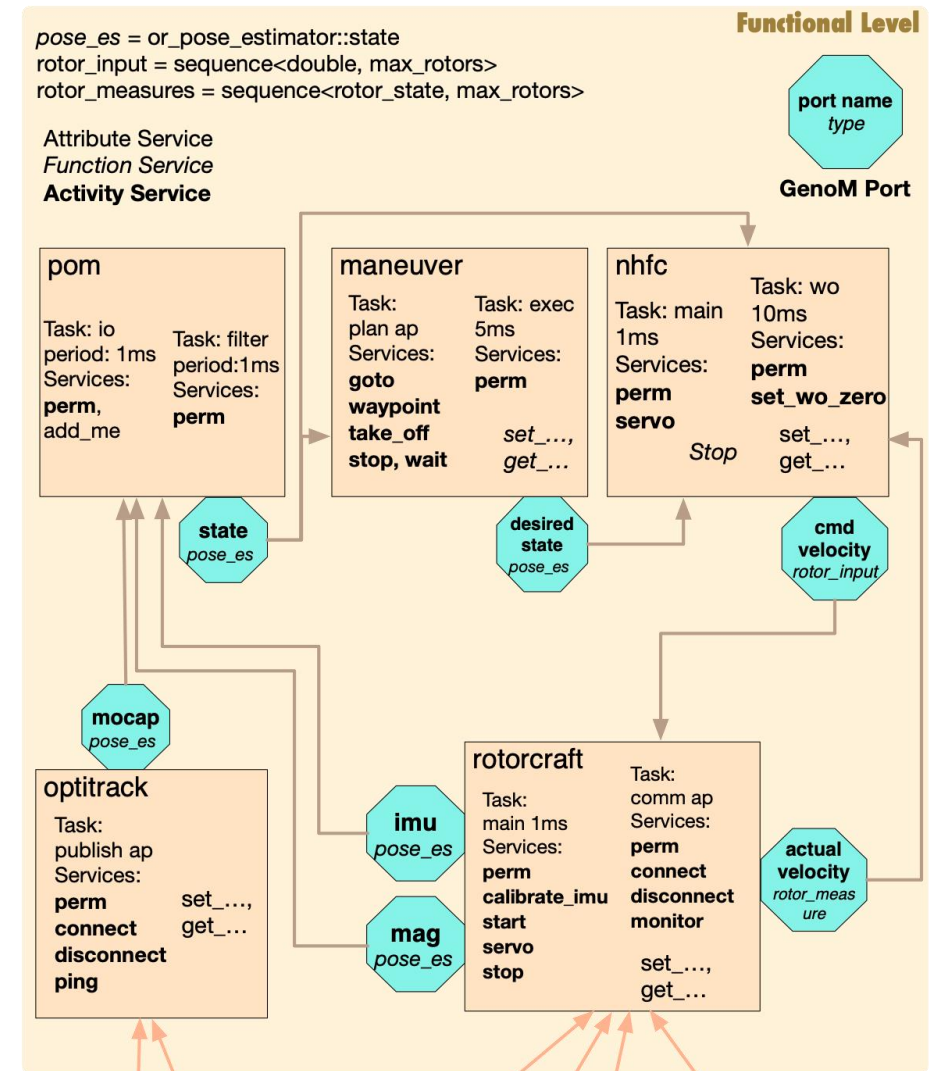


# GenoM (templates)



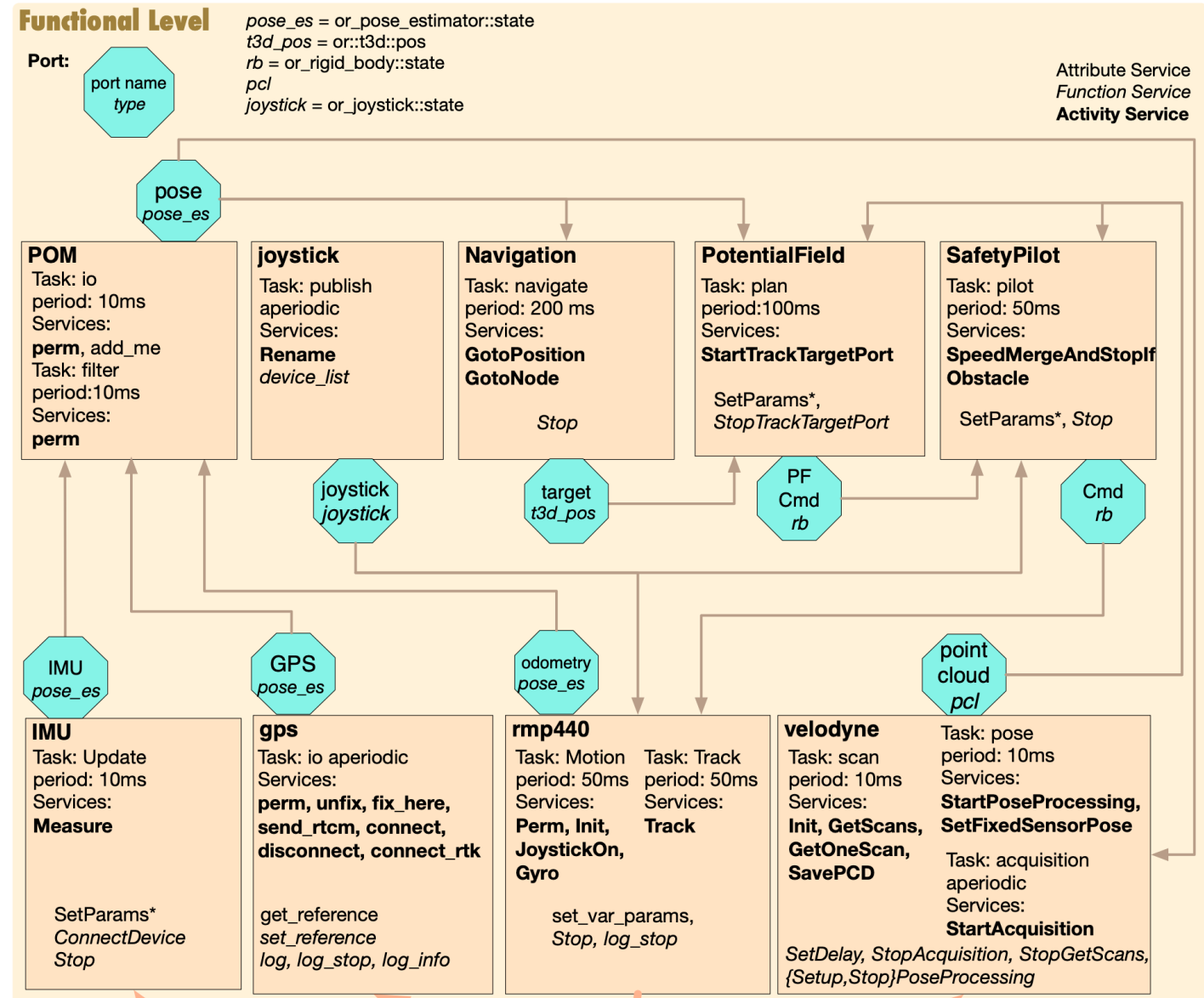
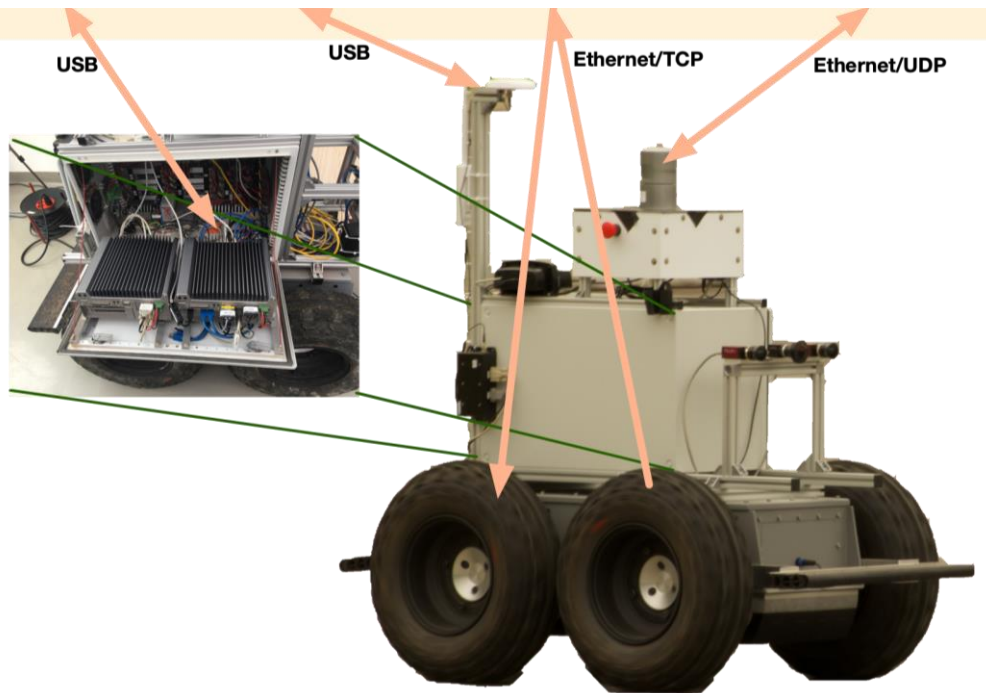
# Drone

- Motion Capture localization
- IMU (angular velocities and accelerations)
- Control each propeller velocity separately
- Only 1 CPU
- Update frequency is  $\sim 1\text{kHz}$



# RMP440: Minnie

- Segway RMP 440
- Fast (up to 8 m/s)
- GPS ; Gyro ; IMU ; Velodyne LIDAR
- 2 recent CPUs



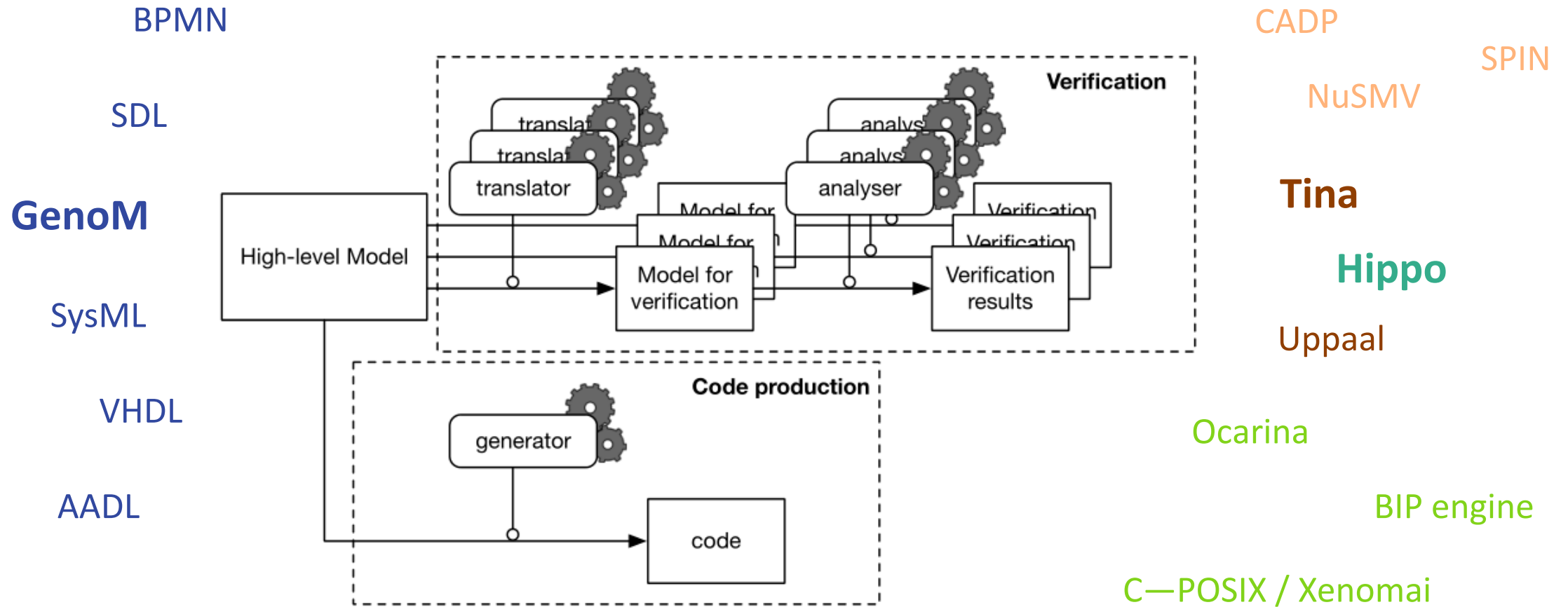
A High-Level Overview of GenoM

Formal Methods 101





Formal-Model Execution Engine

Experimentations

# Generic verification: AnyADL ↔ AnyTool

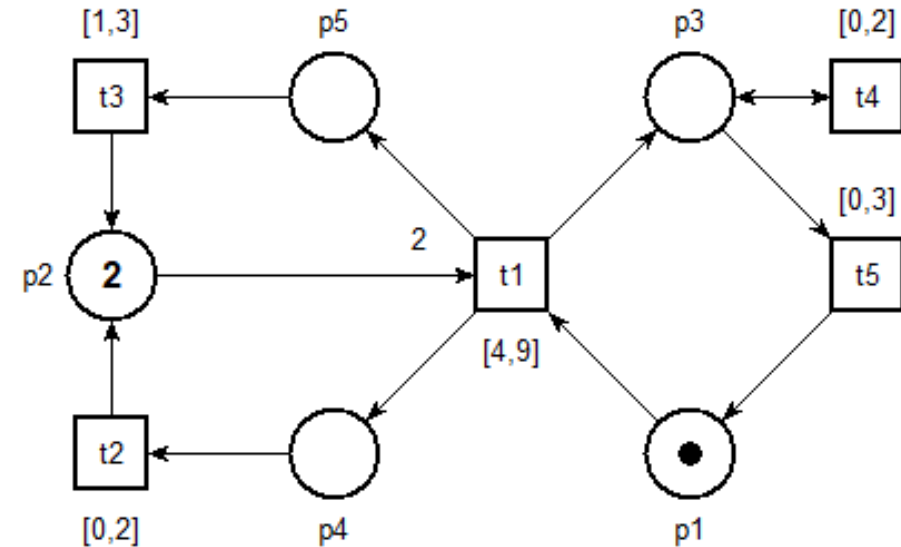


# Some characteristics of GenoM

- It is **opinionated**  
Impose a rather “rigid/strict” way to define components, with little room for fooling/messing around. “Everything is here”, as little magic as possible
- It relies on a **templating mechanism** to generate all the artifacts 
- It is explicit about **error handling** and possible **failure scenarios** 
- It is middleware independent
- It uses explicit constructs to express realtime constraints and requirements: tasks; periods; WCET; ... 
- Behavioral description based on state machines and synchronization on ports 

# Tina

- Modelling based on a Time extension of Petri nets (TPN), with priorities, ...
- Historically: checking protocols ; hardware (now SoC) ; architecture exploration ; etc.
- Toolbox with multiple abstraction and verification methods
  - Reachability analysis
  - Simulation
  - **Model-checking** using different temporal logics



```
Selt version 3.6.0 -- 07/07/20 -- LAAS/CNRS
ktz loaded, 12 states, 29 transitions
0.000s
```

```
- [] (t1 => <> t5);
FALSE
state 0: p1 p2*2
-t1 ... (preserving - t5 /\ t1)->
state 12: p3 p4 p5
-t4 ... (preserving - t5)->
* [accepting] state 15: p2*2 p3
-t4 ... (preserving - t5)->
state 15: p2*2 p3
0.000s
```

# Fiacre and H(ippo)-Fiacre

Think of Fiacre as TPN  
with **datatypes** (arrays,  
structs, fifo queues, ...)  
and **components**  $\Rightarrow$  it  
generates TTS

Hippo adds operators for  
“runtime” **tasks** and  
**events**  $\Rightarrow$  generates  
executable code

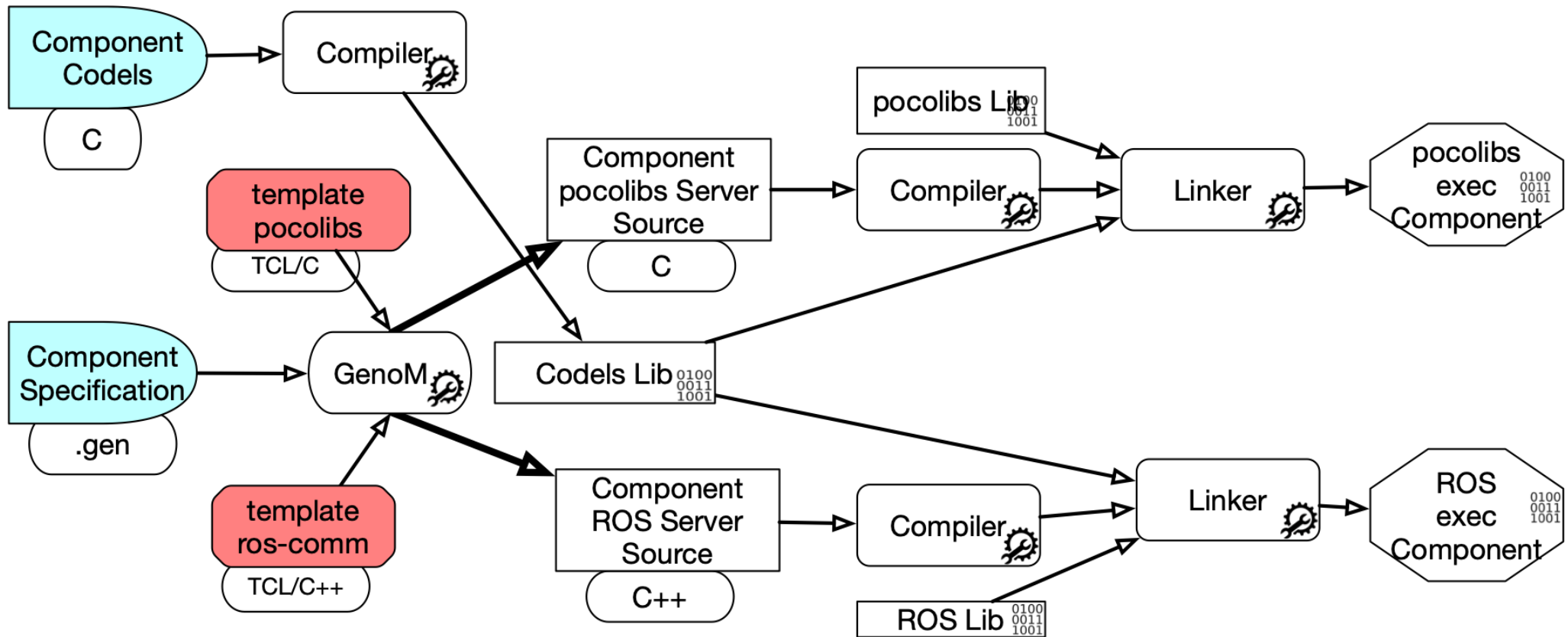
```
type tyEvt is record time : int, id : nat end
type tyDblEvt is array 2 of tyEvt

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

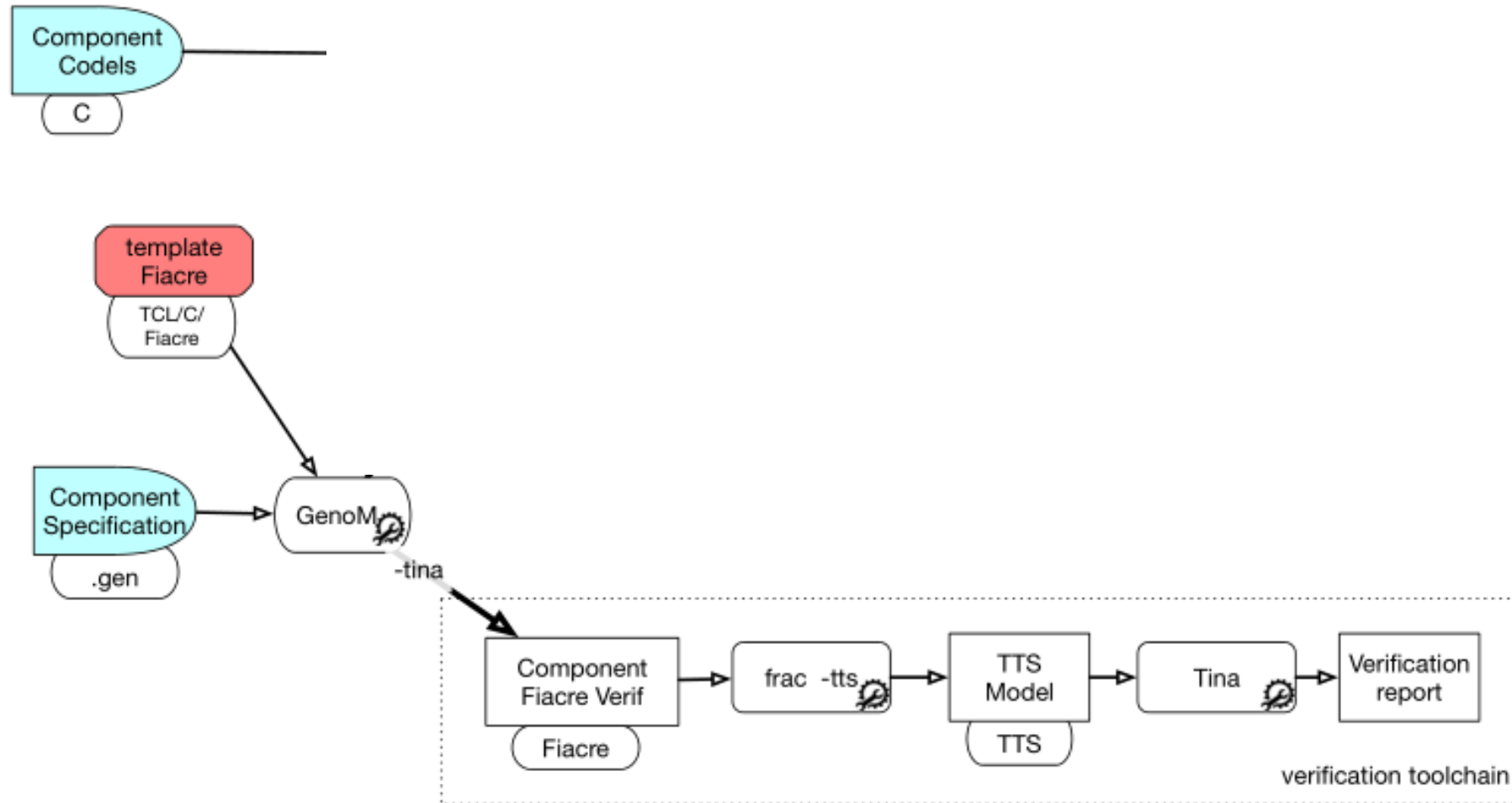
process double_event is
  states wait_first, wait_second, start_print, wait_print
  var tmp : tyDblEvt := [{time=0,id=0}, {time=0,id=0}], ret : nat := 0
  from wait_first
    e?tmp[0]; /* wait first event, assign value to tmp[0] */
    to wait_second
  from wait_second
    select
      wait [200,200];
      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
```



# GenoM → executable toolchain



# GenoM → Fiacre (verif) toolchain



# Short Demo

```
dalzilio@alf MINGW64 ~/Documents/Now/20210709_SHARCV_GENOM/CT_robot
$ sift -M cc.tts cc.ktz
132842 classe(s), 132842 marking(s), 192 domain(s)
4.000s
```

```
dalzilio@alf MINGW64 ~/Documents/Now/20210709_SHARCV_GENOM/CT_robot
$ selt cc.ktz -f '[] - dead'
Selt version 3.6.0 -- 07/07/20 -- LAAS/CNRS
ktz loaded, 132842 states, 420232 transitions
0.969s
TRUE
0.000s
```

```
dalzilio@alf MINGW64 ~/Documents/Now/20210709_SHARCV_GENOM/CT_robot
$
```



## CT\_robot

### IDS:

speed, patrol\_speed  
threshold  
x, y, width, height

### Task:

track 10ms

### Services:

SetThresholdf  
SetPatrolSpeed  
StopTracking  
ColorTrack

CmdPort

Twist

ImagePort

Image

# Why mix Robotics and Formal Methods

- Formal verification is one approach, among others, to increase the trust we have in robotic systems
- Already used in many critical domains
  - with safety standards:** transport, energy, ... **and without:** space, military, ...
- It does not solve “all the problems”
  - but it is a step in the right direction, and it is very good at challenging preconceived ideas
- It can be integrated in existing frameworks

A High-Level Overview of GenoM

Formal Methods 101

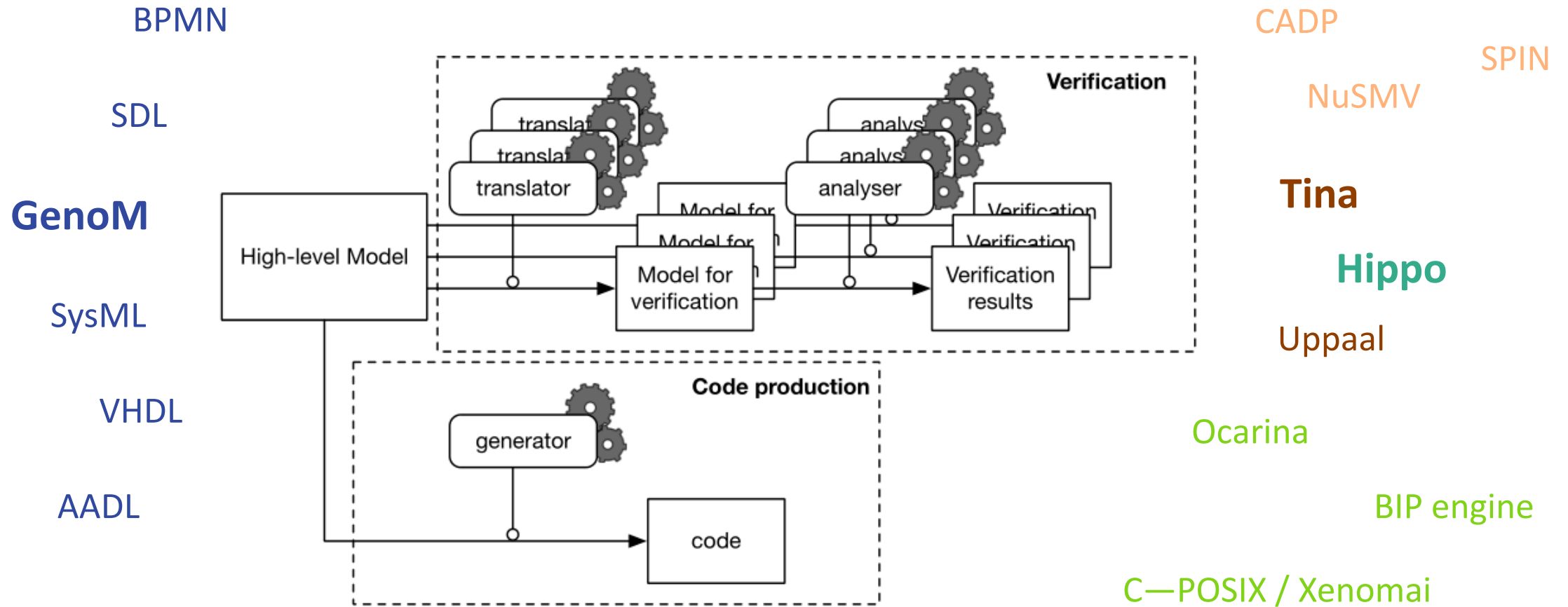
**Formal-Model Execution Engine**

Experimentations

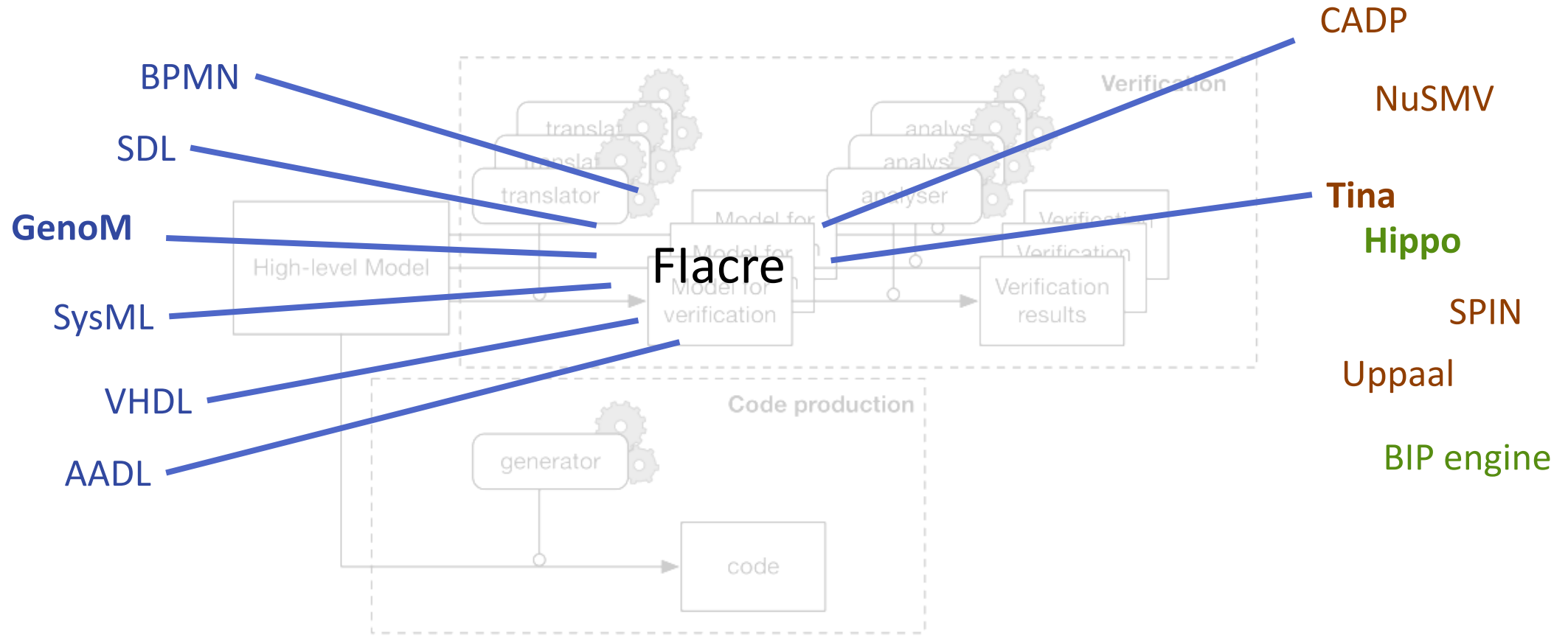
# Hippo: a Faithfull Execution Engine for H-Fiacre

- The interpretation of GenoM into Fiacre is quite precise; it only lacks knowledge about the behavior of codels
    - We get the state of each execution task and activities
    - We know what messages are exchanged/stored in the IDS
    - We track every timing constraints (timeouts, periods, activations, ...)
- ⇒ We can execute a GenoM specification using a **Hippo engine**
- ⇒ This engine is (time-) **faithful** and **predictable**

# Generic verification: AnyADL $\leftrightarrow$ AnyTool

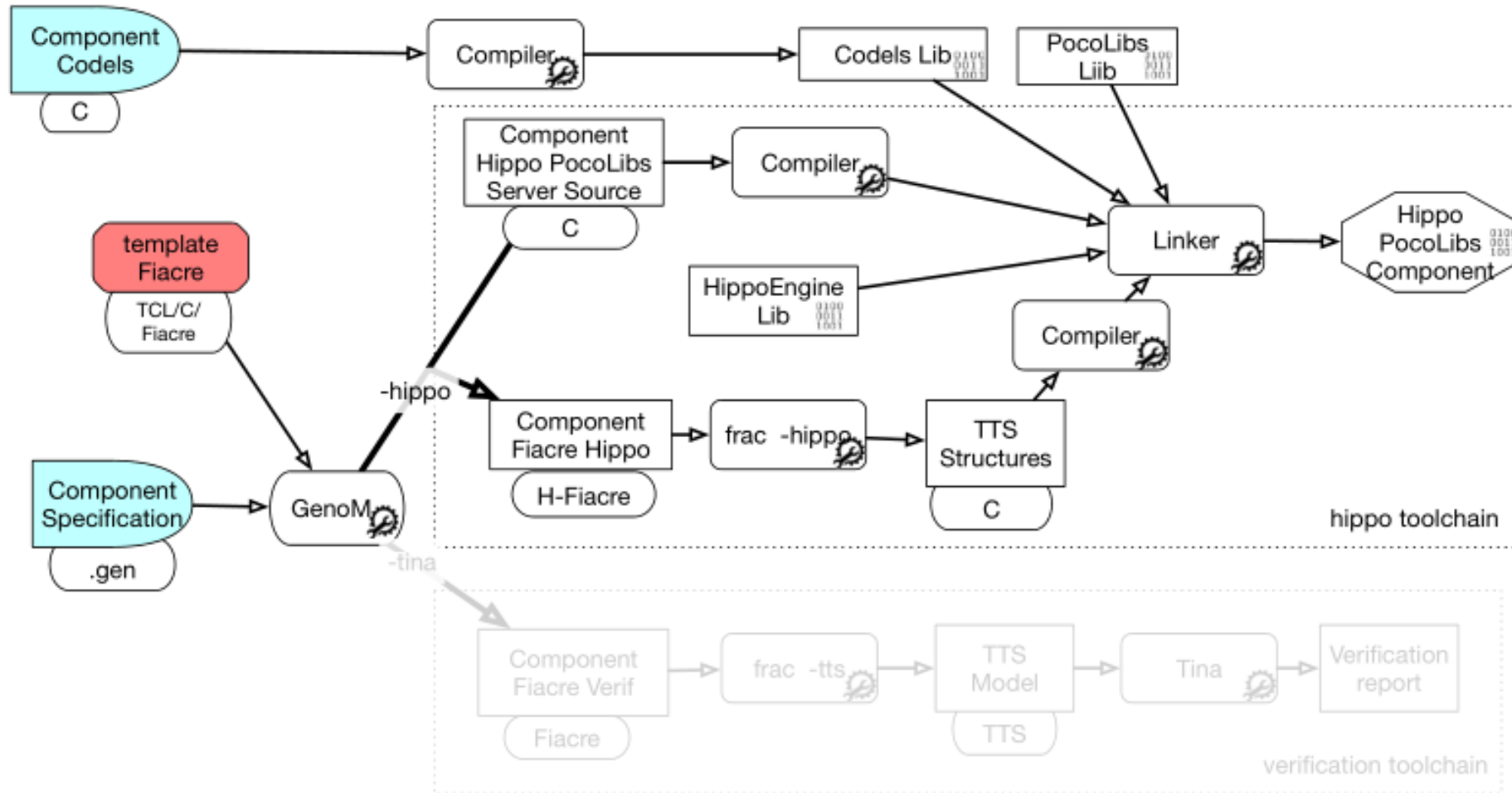


# Generic verification: AnyADL $\leftrightarrow$ AnyTool





# GenoM → Hippo toolchain



A High-Level Overview of GenoM

Formal Methods 101

Formal-Model Execution Engine

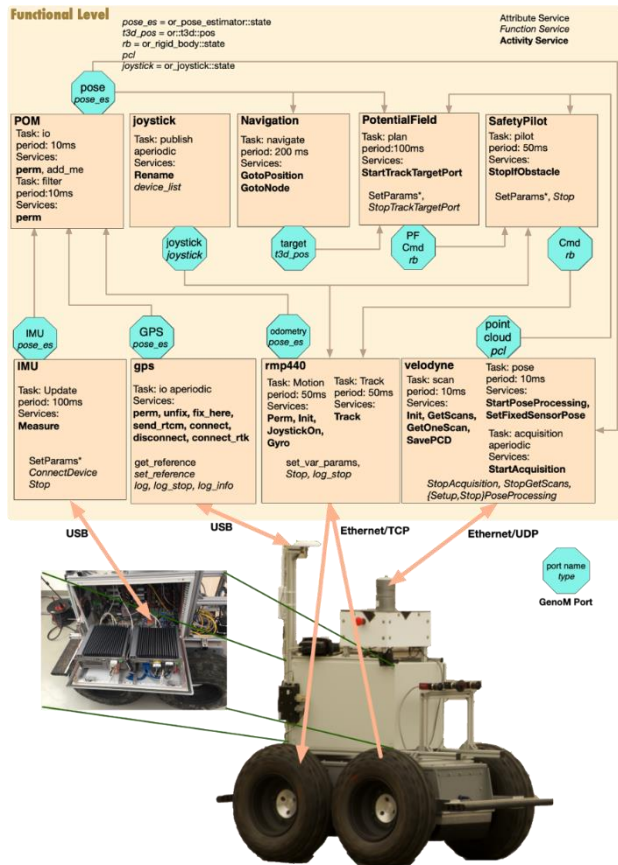
Experimentations

# What did we do with all this ?

- We used it to check and “run” GenoM specifications (does it run ?)
- We checked properties, “offline” (is it true that ... ?)
- A validation of the Hippo Engine (can we trust it ?)  
*We check that  $runtime\ executions \subseteq traces\ in\ the\ formal\ model$*
- An empirical analysis of the Hippo Engine (does it scale well ?)
- We checked properties, online (can we monitor it ?)

# Minnie

<https://youtu.be/vXZiW5tOG54>



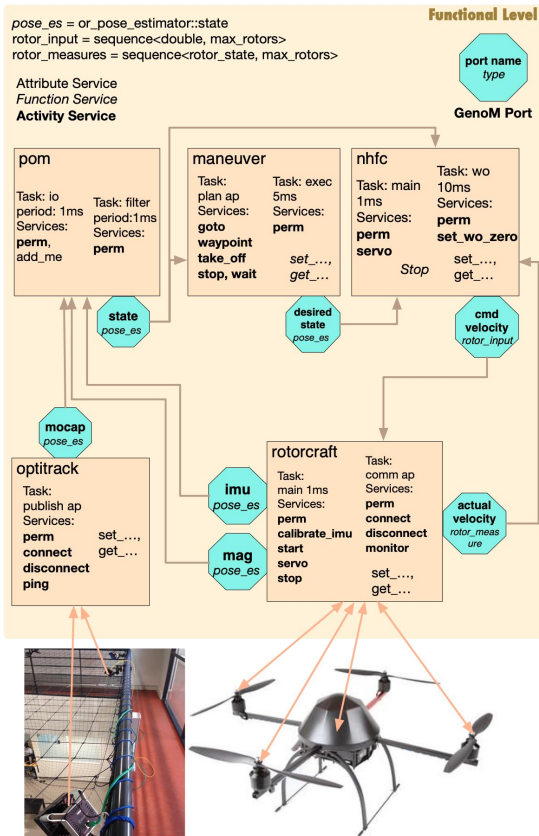
```
[mosh] felix@minnie-superbase: ~/work/minnie/vv/build-hippo-ros-mon/flacre-ros
File Edit View Search Terminal Help
Scan already in local copy.
Sending safe command: l 0 m/s, a 0 deg/s.
Arrived at goal -35.3691 2.2374
PF cmd: l 0.000000 m/s, a 0.000000 deg/s.
Got a PointCloud of 46080 width, 1 height, size 46080.
259.822055 [*** FlacreModel ***] minnie --PATCH-- scan_updated in GetScans:velodyneGetOneScan
End
259.822267 [*** FlacreModel ***] minnie --PATCH-- monitor_wait entered
259.822271 [*** FlacreModel ***] 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*
```

# Drone

[https://youtu.be/30k\\_c-ATY8I](https://youtu.be/30k_c-ATY8I)



The top part of the image shows a drone flying in a room with a window. The middle part is a terminal window displaying flight logs with timestamps and status messages for various components like 'quad-hippo-pocolibs', 'FiacreModel', 'maneuver', 'rotorcraft', and 'nhfc'. The bottom part is a 3D simulation of the drone's path on a grid, with a red line indicating the flight trajectory and a blue line for the current path. The simulation is running at 31 fps.

```

    File Edit View Search Terminal Help
    quad-hippo-pocolibs: emergency descent due to uncertain angular velocity estimation
    quad-hippo-pocolibs: recovered from emergency
    1258.792568 12584111 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1261.117659 12607355 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1261.117690 12607355 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1261.117800 12607355 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1263.587085 12632844 [*** FiacreModel ***] quad rotorcraft: CT start request processing
    1264.809554 12644226 [*** FiacreModel ***] quad rotorcraft: Activity start DONE (ether), back to ET.
    1264.809560 12644226 [*** FiacreModel ***] quad rotorcraft: ET main activity returned ACT_ETHER. 3
    1264.805732 12644226 [*** FiacreModel ***] quad rotorcraft: CT processes main activities, activity report.
    1264.806675 12644235 [*** FiacreModel ***] quad maneuver: CT set current state request processing
    1264.806930 12644236 [*** FiacreModel ***] quad maneuver: Activity set current state DONE (ether), back to ET.
    1264.806950 12644236 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 1
    1264.806979 12644236 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1264.807773 12644242 [*** FiacreModel ***] quad nhfc: CT set current position request processing
    1264.807836 12644242 [*** FiacreModel ***] quad nhfc: CT set current position interrupts a running activity in main.
    1264.808896 12644253 [*** FiacreModel ***] quad nhfc: Activity servo in default stop.
    1264.808906 12644253 [*** FiacreModel ***] quad nhfc: Activity servo throw an exception, back to ET.
    1264.808915 12644253 [*** FiacreModel ***] quad nhfc: ET main activity returned ACT_ETHER. 1
    1264.809016 12644253 [*** FiacreModel ***] quad nhfc: CT processes main activities, handle exception.
    1264.809029 12644253 [*** FiacreModel ***] quad nhfc: CT processes main activities, activity report.
    1264.809954 12644264 [*** FiacreModel ***] quad nhfc: Activity set current position DONE (ether), back to ET.
    1264.809965 12644264 [*** FiacreModel ***] quad nhfc: ET main activity returned ACT_ETHER. 2
    1264.810009 12644264 [*** FiacreModel ***] quad nhfc: CT processes main activities, activity report.
    1264.810789 12644272 [*** FiacreModel ***] quad rotorcraft: CT servo request processing
    1264.811377 12644275 [*** FiacreModel ***] quad nhfc: CT servo request processing
    1295.072682 12946867 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1296.306559 12959205 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1296.306582 12959205 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1296.306689 12959205 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1306.977445 13065901 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1310.193159 13098055 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1310.193183 13098055 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1310.193276 13098055 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1310.194974 13098069 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1312.213688 13118255 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1312.213716 13118255 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1312.213810 13118255 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1312.214797 13118262 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1313.714158 13133255 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1313.714185 13133255 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1313.714280 13133255 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1313.715406 13133263 [*** FiacreModel ***] quad maneuver: CT goto request processing
    1315.214750 13148255 [*** FiacreModel ***] quad maneuver: Activity goto DONE (ether), back to ET.
    1315.214781 13148255 [*** FiacreModel ***] quad maneuver: ET plan activity returned ACT_ETHER. 5
    1315.214875 13148255 [*** FiacreModel ***] quad maneuver: CT processes plan activities, activity report.
    1315.215800 13148262 [*** FiacreModel ***] quad maneuver: CT goto request processing

    maneuver::3
    etlclsh > setup
    nhfc::2
    etlclsh > draw
    draw H draw I draw 0 draw P draw hippo
    etlclsh > draw hippo -2 0 0.3 0.4 0.8
    etlclsh > setup
    nhfc::3
    etlclsh > carre
    etlclsh > land
    etlclsh >
    etlclsh >
    etlclsh >
    etlclsh > set
    set setup
    etlclsh > ::maneuver::goto 0 0 0.3 0 0 &
    maneuver::4
    etlclsh > setup
    nhfc::4
    etlclsh > ::maneuver::goto 0 0 0.5 0 0 &
    maneuver::5
    etlclsh > draw hippo -2 0 0.5 0.4 0.8
  
```

# Controlling Minnie with Hippo

- The GenoM spec for Minnie compiles into a Hippo model with 197 tasks, 9 event ports, 441 extern functions, 1780 (Petri) transitions
- Hippo runs the whole experiment at 10 kHz in one process.
- The load is  $\approx$  5-10% above normal GenoM usage, without noticeable slowdown
  - We report task period overshoots
  - We detect possibly uninitialized port reads
- The runtime is implemented on Linux (better with PREEMPT\_RT) and uses POSIX services with SCHED\_FIFO (~ fixed priority sched.)

# Offline Verification for Minnie

- **Schedulability:** it is an invariant, `[] – task_overshoot`  
We can take into account the number of cores (we found scheduling errors when using the Velodyne component with less than 3 cores)
- **Mutual Exclusion:** (also a safety property)

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

- **Delay to Stop:** example of quantitative property  
We found a WCRT of 141 ms (85 cm before we brake),  
To be compared with a WCET of 43 ms for the slowest code!

# Conclusion

GenoM3

<https://git.openrobots.org/projects/genom3>

Template GenoM3 Fiacre (ROS et pocolib)

<https://redmine.laas.fr/projects/genom3-fiacre-template/gollum/index>

Expérimentation sur Minnie RMP440

<https://redmine.laas.fr/projects/minnie/gollum/fiacre>

Expérimentation sur un drone

<https://redmine.laas.fr/projects/drone-v-v/gollum/index>

Papier sur la V&V en robotique

[hal-02927311](https://hal.archives-ouvertes.fr/hal-02927311)

Papier sur Fiacre/Hippo/GenoM3

[hal-03017661](https://hal.archives-ouvertes.fr/hal-03017661)



# Cl\_robot.gen

```

/* ----- Activity Services Definition ----- */
activity ColorTrack (in color tracked_color, // the color we want to track in the image
                    out long lost) { // we return how many time we lost the target

doc    "Produce a twist so the robot follow a color in an image";
local  boolean found; // to help counting found/lost status
task   track;         // The task in which ColorTrack instance will execute
validate ValidateColor(local in tracked_color); // validate the values passed as argument

// Automata syntax
// codel <state> c_function({{ids|port|local}? {in|out|inout} arg_k,*})
//                               yield {pause:::?<state_i> {, {pause:::?<state_i>};
// - 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 leave

codel <start>  InitColorTrack(local out lost, local out found) yield Find;
codel <Find>   GetImageFindCenter(port in ImagePort, local in tracked_color,
                                ids in threshold, local out found, ids out width,
                                ids out height)
                yield pause:::Find, // no new image, wait next cycle or
                Lost,                // lost the color
                CompCmd,             // found the color, compute the speed
                ether;               // in case of error.
codel <Lost>  ComputeSpeedWhenLost(ids out cmd, ids in patrol_speed,
                                   local out found, local out lost)
                yield PubCmd;
codel <CompCmd> ComputeSpeed(ids in x, ids in y, ids in width, ids in height,
                             yield PubCmd;
codel <PubCmd> PublishSpeed(ids in cmd, port out CmdPort)
                yield pause:::Find, // Loop back at the search in the next cycle
                ether;               // in case of error.
codel <stop>  StopRobot(ids out cmd, port out CmdPort) // stop is a predefined state
                yield ether; // ColorTrack execution will jump to this state
                //service is interrupted

after  SetPatrolSpeed; // we can only call this service after SetPatrolSpeed
throw  bad_cmd_port, bad_image_port, // Possible errors in the codel's arguments
       opencv_error, bad_color;     // Any will force execution to end

interrupts ColorTrack; // Only one ColorTrack service running at a time
};
};

```

