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

- ROS
- GenoM
- Pocolibs
- Programming Languages

Formal Methods

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

Functional Architecture; Runtime Systems

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

- Process Calculi
- Data and Types
- GenoM
- Hippo
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 (code)

```plaintext
#pragma require "openrobots2-ldl >= 2.0"
#pragma require "minnie-ldl"

#include "or/pose/pose_estimator.gen"
#include "ni/sensor/pcl.ldl"

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

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

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

  /* --- lds --------------------------------------------------------------- */
  lds f
  AcquisitionParams acquisition_params; // Acquisition parameters
  PacketBuffer packet_buffer; // Buffer to store time stamped raw packets
  PoseBuffer pose_buffer; // Buffer to store time stamped pose.
  long fd; // file descriptor to get the raw packets (UDP)
  long usec_delay; // for fault injection purpose to delay port scan writting
};

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

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

  throws e_mem, e_grabber;
};

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

```c
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
Fiacre and H(ippo)-Fiacre

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

Hippo adds operators for “runtime” tasks and events ⇒ generates executable code
GenoM → executable toolchain
GenoM → Fiacre (verif) toolchain
Short Demo

dalzilio@alf  ~/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  ~/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  ~/Documents/Now/20210709_SHARCv_GENOM/CT_robot
$
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 if (time-) **faithful** and **predictable**
Generic verification: AnyADL ↔ AnyTool
Generic verification: AnyADL ↔ AnyTool

- BPMN
- SDL
- GenoM
- SysML
- VHDL
- AADL
- Flacre
- CADP
- NuSMV
- Tina
- Hippo
- SPIN
- Uppaal
- BIP engine
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
Drone

https://youtu.be/3Ok_c-ATY8I
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 \(\approx 5\text{-}10\%\) 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 (\(\sim\) fixed priority sched.).
Offline Verification for Minnie

• Schedulabity: 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)

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

• 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 codel
Conclusion

GenoM3

Template GenoM3 Fiacre (ROS et pocolibs)

Expérimentation sur Minnie RMP440

Expérimentation sur un drone

Papier sur la V&V en robotique

Papier sur Fiacre/Hippo/GenoM3

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

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

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

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

hal-02927311

hal-03017661
```c
/* ---------------- 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 times 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 <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 of activity
        // lost, // lost the color
        CompCmd, // found the color, compute the speed
        ether; // in case of error.
    codel <Lost> ComputeSpeedWhenLost(// stop is a predefined activity
        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
        ether; // in case of error.
    codel <stop> StopRobot(ids out cmd, port out CmdPort) // stop is a predefined activity
        yield ether; // ColorTrack execution will jump to this activity
        // service is interrupted
    after SetPatrolSpeed; // we can only call this service after SetPatrolSpeed
    throw bad_cmd_port, bad_image_port, // Possible errors in the code
        opencv_error, bad_color; // Any will force execution to end
    interrupts ColorTrack; // Only one ColorTrack service running at a time
};
```