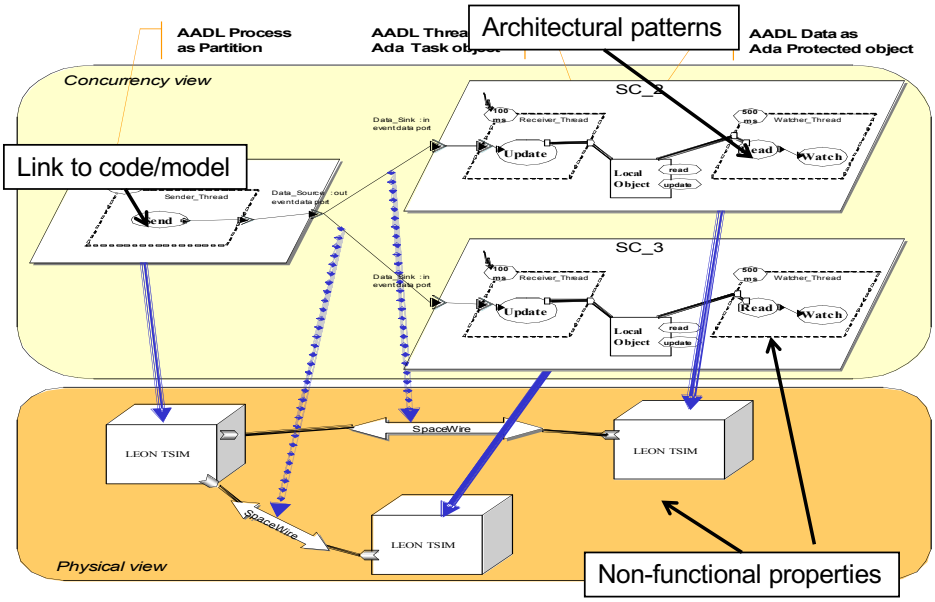
Ocarina Status Update 2016Q3

Outline

1. About Ocarina
2. Updates on runtime
3. Future projects

MBS²E/AADL@ISAE

Architecture helps you focusing on the actual system



AADL covers many parts of the V cycle: model checking, scheduling, safety and reliability (ARP4754) and code generation

ISAE contributions to SAE AADL since 2009

Lead on the Ocarina toolset, used by ESA

Code generation : Ada, C (POSIX, ARINC653), RTOS

TRL 7 with ESA (ECSS E-40)

Scheduling: Cheddar, MAST

TRL 4 with ESA

Model checking: Petri Nets (Time and CPN), LNT

TRL 2 (PhD contribution)

Architectural Constraints/Requirements checks

TRL 6, being standardized

Link with SysML & Capella

TRL 1, under evaluation

Ocarina: an AADL code generator http://www.openaadl.org

- **Ocarina is a Compiler for AADL and Runtimes for High-INtegrity Architectures**
  - Previously was Ocarina is a Compiler for AADL and Runtime IN Ada
- **Ocarina is a stand-alone tool for processing AADL models**
  - Free Open Source Software (as in *Free* speech and *Free* beer)
  - Command-line, or integrated third-party tools
    - OSATE (CMU/SEI), TASTE (ESA), AADL Inspector (Ellidiss)
- **Code generation facilities target PolyORB-HI runtimes**
  - Ada HI integrity profiles, with Ada native and bare board runtimes
  - C POSIX, RTEMS, Xenomai, FreeRTOS, Xtratum for Embedded
  - C ARINC653 for avionics systems
- **Generated code quality tested in various contexts**
  - For WCET exploration, support for device drivers, …
- **Written to meet most High-Integrity requirements**
  - Follow Ravenscar model of computations, static configuration of all elements (memory, buffers, tasks, drivers, etc.)
- **Contributions from PhD students, partners (SEI, ESA)**

Example: a basic periodic system

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```


Example: a basic periodic system

Software view

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```

Example: a basic periodic system

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

Deployment view

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```


Example: a basic periodic system

User-provided code

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```

Example: a basic periodic system

User-provided code

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

Schedulability analysis
Bound-T + Cheddar

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```


Example: a basic periodic system

User-provided code

```
--  Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

Schedulability analysis
Bound-T + Cheddar

Stack size analysis, e.g.
Bound-T or GNATStack

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

process implementation node_a.impl
subcomponents
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```

Example: a basic periodic system

Code generation
Ocarina

User-provided code

```
-- Reduced version, only 1 task shown

subprogram Hello_Spg_1
properties
  source_language => Ada95;
  source_name     => "Hello.Hello_Spg_1";
end Hello_Spg_1;

thread Task end Task;

thread implementation Task.impl_1
calls {
  P_Spg : subprogram Hello_Spg_1;
};
properties
   Dispatch_Protocol                => periodic;
   Period                           => 1000ms;
   Compute_Execution_time           => 0 ms .. 3 ms;
   Deadline                         => 1000 ms;
   Source_Stack_Size                => 13952 B;
end Task.impl_1;
```

Schedulability analysis
Bound-T + Cheddar

Stack size analysis, e.g.
Bound-T or GNATStack

```
processor cpurm
properties
  Deployment::Execution_Platform => ERC_32;
end cpurm;

processor implementation cpurm.impl
properties
   Scheduling_Protocol => HPFP;
end cpurm.impl;

process node_a end node_a;

cess implementation node_a.impl
components
   Task1 : thread Task.impl_1;
end node_a.impl;

system rma end rma;

system implementation rma.impl
subcomponents
   node_a : process node_a.impl;
   cpu_rm : processor cpurm.impl;
properties
   Actual_Processor_Binding =>
       reference(cpu_rm) applies to node_a;
end rma.impl;
```


Some lessons learnt on code generation

- **Now standardized as part of AS5506/1A**
- **Rationale: AADL capture a real-time architecture, connected to analysis results**
  - Tools of the AADL ecosystem to run analysis at model-level
    - E.g. scheduling, memory budget, task/processor allocation, etc.
  - Third-party code: C/C++/Ada/SCADE/Simulink/SDL, …
- **Evaluated in various R&D projects: TASTE, Flex-eWare, SAVI**
  - Performance equivalent to hand-written code,
  - < 10% memory overhead because of internal tables
- **Code generation is one "minor" addition to the model, but provide one huge benefits: closing the loop**
  - Model -> generate code -> evaluate binary -> Model' in reduced time
  - E.g. adjusting memory budgets, WCET, stack size, etc.
- **Reason #1 for considering code generation**
  - Leverage automation through potential qualified tools and runtimes
  - Easier to combine with code analyzers: WCET, code coverage, style checks, etc.

Outline

1. About Ocarina
2. **Updates on runtime**
3. Future projects

Ocarina runtimes: PolyORB-HI/Ada

- **Target Ada Ravenscar and High-Integrity runtimes + SPARK2014**
  - SPARK2014 uses Ada2012 aspects to define contracts that are evaluated at run-time, or proved formally off-line
- **Based on the Ravenscar & HI Ada profiles**
  - Meets stringent requirements for High-Integrity systems, e.g. ESA
  - Checked at compile-time by Ada compiler, GNAT
  - Also support for SPARK/Ada in the pipeline
    - Proof of absence of Run-Time Errors, contract-based programming
- **Supports native, RTEMS, Ada bare-board**
- **Easy to retarget thanks to Ada portability**
  - Reduced to configuration of the compilation chain
  - Any Ravenscar-capable runtime should work out-of-the box
  - Yet remain compact and efficient

Ocarina runtimes: PolyORB-HI/C

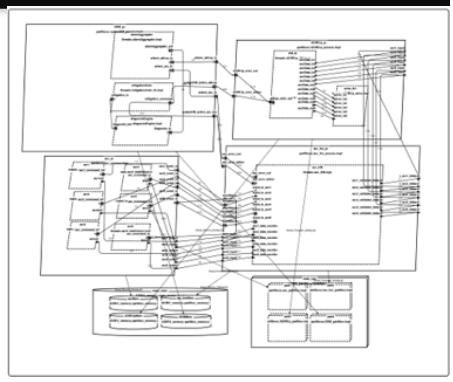
- **Follow the same design principles from the Ada runtime**
  - No memory allocation: static resources, threads, etc.
- **Set of primitives to build all AADL entities (threads, ports)**
- **Set of macros to adapt runtime to target-dependent APIs**
  - Supported: RT-POSIX, C/RTEMS, VxWorks classic API, Xenomai, Windows, FreeRTOS, XtratuM, ..
- **Tested on different configurations:**
  - Restricted libc: GNU/Linux on Nintendo DS and Nokia 770
  - LEON2/3 boards, SBC8349E, STM32 Discovery, …
- **One mode to target directly ARINC653 APEX**
  - Tested with DDC-I DeOS and WRS VxWorks 653

AADLib: towards library of model elements

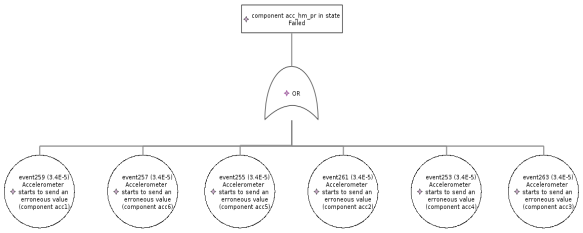
- **Need reusable library of models**
  - Patterns for platform code, integrated for both tool exploration and code generation
- **Combined in a library of model: AADLib**
  - Additional property sets for deployment, configuration capture
  - Library of devices, boards, etc.:
    - Processors: ARM, AVR, PowerPC, SPARC, x86 processors
    - Buses: AFDX, ARINC429, CAN, Ethernet, I2C, MIL-STD 1553, PCI, SpaceWire, UART, USB, with known limits in bandwidth, packet size, etc.,
    - Reference design: Arduino, Aeroflex Gaisler boards, Wind River SBCs
    - Device drivers for Unix, Windows, RTEMS, Xenomai, Ada RTS, …
- **Reusable set of examples**
  - To evaluate tool support, benchmark for tool interoperability
- **Available as a set of Open Source models**

Example: ADIRU case study (with CMU/SEI)

- **ADIRU case study from Boeing 777EHR**
  - Many elements publically available
- **Code generation & safety analysis**
  - All in one model, approx. 50 entities
  - 2500 SLOCs at AADL level
- **Code generation for ARINC653**
  - VxWorks 653 & DDC-I DeOS
  - Integration of SCADE blocks
- **Safety analysis**
  - Soundness of the models
  - Fault-tree, Fault Impact generated
- **http://aadl.info/aadl/demo-arinc653/**



Component	Initial Failure Mode	1st Level Effect	Failure Mode	second Lev
acc1	Failed	{ValueErroneous} accData -> acc_pr:acc1_input	acc_pr {ValueErroneous}	{ValueErro
acc2	Failed	{ValueErroneous} accData -> acc_pr:acc2_input	acc_pr {ValueErroneous}	{ValueErro
acc3	Failed	{ValueErroneous} accData -> acc_pr:acc3_input	acc_pr {ValueErroneous}	{ValueErro
acc4	Failed	{ValueErroneous} accData -> acc_pr:acc4_input	acc_pr {ValueErroneous}	{ValueErro
acc5	Failed	{ValueErroneous} accData -> acc_pr:acc5_input	acc_pr {ValueErroneous}	{ValueErro
acc6	Failed	{ValueErroneous} accData -> acc_pr:acc6_input	acc_pr {ValueErroneous}	{ValueErro

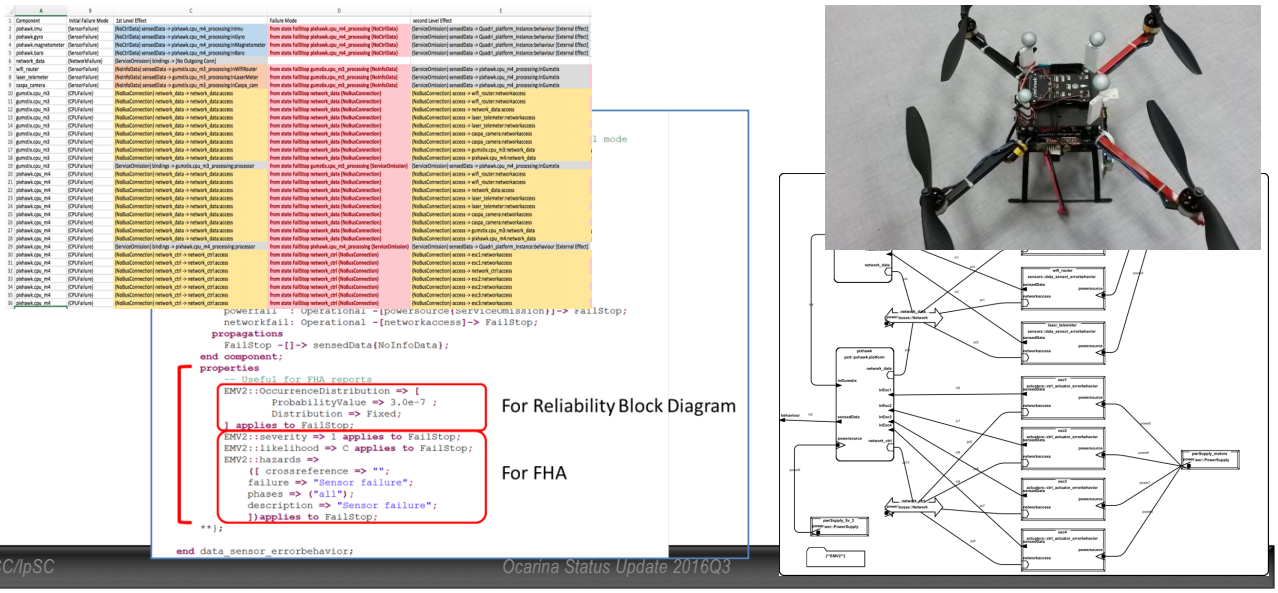


Outline

1. About Ocarina
2. Updates on runtime
3. **Future projects**

Highlight#1: UAV safety analysis and code gen.

- **ISAE develops a line of UAVs: fixed wings, quad copters, …**
  - Incoming regulations to ensure safety during operation
  - Safety, scheduling and code generation, targetting Cortex-R
- **2016Q4: new project to build analyzable AP brick using AADL**

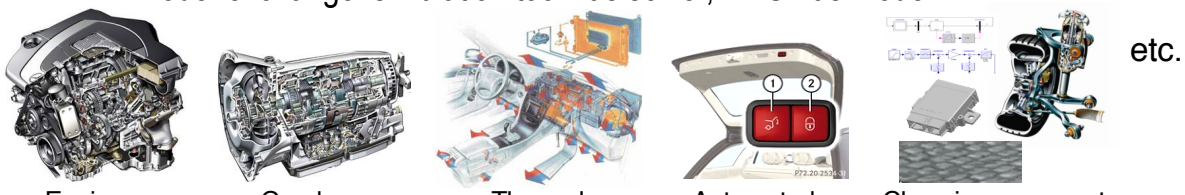


Highlight#2: space-qualification of Ocarina

- **As part of H2020 European project PERASPERA/ESROCOS**
  - With GMV as project leader
  - Running November 2016 – March 2019
- **Objective: develop extensions for TASTE towards space-qualified robotics applications, with variants:**
  - Lab-Quality: regular Linux
  - Space-Quality: RTEMS for LEON3 + qualification materials
- **Impact on Ocarina development:**
  - Qualification material to be produced
  - In-depth testing and analysis of code for major platforms
  - Ultimately, be embedded on a space rover

Highlight#3: Coupling AADL and FMI

- **Recurring needs to combine embedded architectures with physical environment in a rapid prototyping process**
  - » E.g. UAV autopilot in simulated world: wind, buildings, …
  - » This is captured with dedicated models, e.g. Simulink, Modelica
- **Simulation of systems combining heterogenous model is being standardized as part of the FMI standard,**
  - » FMI defined FMU components (zip + XML descriptor) for
    - Co-simulation: simulation tool orchestrates FMU (model + solver)
    - Model-exchange: simulation tool has solver, FMU has model



Engine with ECU | Gearbox with ECU | Thermal systems | Automated cargo door | Chassis components, roadway, ECU (e.g. ESP) | etc.

courtesy Daimler

functional mockup interface for model exchange and tool coupling

Highlight#3: Coupling AADL and FMI

- **Code can be generated from AADL models for various RTOSes**
  - » Could add a target for a simulated-time API, e.g. HLA or others
- **AADL/FMI integration scenarios will be investigated as part of new grant, starting Sep. 2016 with Samares Eng. as co-PI**

#1: FMU as implementation for AADL devices

- » Provide (real-time) model-in-the-loop capabilities, using AADL code generation capabilities to build simulation master
- » FMU = simulate real-world as viewed from “Cyber” part

#2: Turn an AADL model into an FMU

- » Extend target by providing a simulation kernel instead of RTOS
- » Cyber part becomes a facility, integratable in system-level tools

#3: Drive system-level simulation from AADL

- » AADL semantics + properties to build a simulation master
- » Current research challenge from FMI perspective: notion of time, composability of simulation models, data types, ...

- **Case studies to be developped based on ISAE existing platforms: UAV or others**