Applying WCET analysis at architectural level

Olivier Gilles, Jérôme Hugues
In Distributed Real-time Embedded (DRE) systems, we need to check:

- Timing-related constraints
- Resources consumption-related constraints

Tools for analysis

- Exist for legacy code
- Code generation and frameworks are widely used
  - Implies manual configuration of analysis tools

Solution: a model-based and automatized approach

- Using model information to build the system
- Using existing tools to analyze the system
End-to-end analysis requirements

- An automatized evaluation framework would:
  - Deduce binaries from the system architectural model
  - Evaluate the DRE system

- Model-level analysis
  - Non-functional data: thread priority, periodicity...
  - Inter-process communications framework

- Binary-level analysis
  - Actual subprogram WCET and footprint

- Using them together allows:
  - Scheduling and latency analysis
  - Memory utilization
A 3-steps evaluation framework

- Code generation & compilation
- Binaries analysis
- Model evaluation

Based on 3 system representations

- Architectural model
- Binary code
- Annotated model
**Architecture evaluation framework**

- **A 3-steps evaluation framework**
  - Code generation & compilation
  - Binaries analysis
  - Model evaluation

- **Based on 3 system representations**
  - Architectural model
  - Binary code
  - Annotated model
Architecture evaluation framework

- A 3-steps evaluation framework
  - Code generation & compilation
  - Binaries analysis
  - Model evaluation

- Based on 3 system representations
  - Architectural model
  - Binary code
  - Annotated model
The actual evaluation pipeline

- Based on the AADL
- For each step of the evaluation framework, we assemble existing tools or design new ones
  - Code generation: Ocarina + PolyORB-HI
  - Binaries evaluation: Bound-T, GnatStack...
  - Annotated model evaluation: REAL/REC, Cheddar...
The AADL

**Architecture Analysis & Design Language** allows to:

- Describe a DRE system architecture (hardware and software)
- AADL description = set of typed components
- Components communicate through features (ports, accesses...)
- Linked with connections
- Properties associated to any entity

```plaintext
thread implementation Plant.i
  calls
    plant: subprogram ssp.i;
  connections
    parameter ctrl_in ->
      plant.ctrl_in;
    parameter plant.feedback ->
      Outputfeedback;
  properties
    Dispatch_Protocol => Periodic;
    Period => 10 Ms;
end Plant.i;
```
Code generation

- Ocarina: a tool to manipulate AADL
  - AADL parsers and analyzer
  - Legacy code + AADL full models => Ada/C code generation
  - Can support multiple middleware

- PolyORB-HI: a High Integrity middleware
  - Enforces the Ravenscar profile for Ada code & Ada high-integrity restrictions
  - Automatic configuration from AADL specs
  - Resources statically computed and allocated
  - A minimum HI middleware generator
Tool configuration

Configuring Bound-T: WCET analysis by Tidorum

- A Threads and Protected Objects File (TPOF) specify which subprograms must be analyzed.
- An assertions file specify which subprograms must not be analyzed.
  - Both can be deduced from knowledge of the AADL model and Ocarina and PolyORB-HI patterns.

Bound-T return a Execution Skeleton File

- Extract relevant data (WCETs), translate them from CPU cycles => duration.
- Insert them into the AADL model.
TPOF generation

- **TPO file**
  - thread name: `<process_name>_<thread_name>`
  - main subprogram name: `polyorb_hi_generated__activity_<thread_name>_job`
  - Periodicity type: according to the `Dispatch_Protocol` property

```verbatim
thread plant
   -- (...)
properties
   dispatch_protocol => periodic;
end plant;

process implementation sunseek
subcomponents
   th_plant : thread plant.i;
end sunseek;
```

```verbatim
thread sunseek_th_plant
   type cyclic
   root
   polyorb_hi_generated__activity_th_plant_job
end sunseek_th_plant;
```
Assertion file

- Use a predefined assertion template file
  - Output primitives: provided by the drivers specs
  - Fixed-point operations: recursive
  - Exception handling: no exceptions in HRT systems
  - Kernel primitives: provided by the runtime specs
- Subprograms WCET can be deduced from the AADL or runtime framework specifications

Example
AADL thread th_plant =>

polyorb_hi_generated_activity__th_plant_interrogators__send_outputXn
depends on the sent data size =>

can be deduced from the AADL specs and device driver specs
Annotating models

- Parsing of the ESF
  - For each thread, parse the WCET, find the related processor, translate the cycles to an execution time, and put it back into the AADL model

```
processor erc32
  properties
    Processor_Speed => 50 Mhz;
  end erc32;

thread implementation Plant_Type.Plant
  (...)
  properties
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time => 80 Us .. 81 Us; -- Computed by BoundT
  end Plant_Type.Plant;
```
Using Cheddar

- Cheddar: schedulability analysis tool (RMA, EDF...)
- Use AADL properties
- Compute Execution Time includes data from static analysis + the runtime framework specs

```
thread implementation Plant.i
  -- (...)
connections
  -- (...)
properties
  Dispatch_Protocol => Periodic;  -- Modeler-defined
  Period             => 10 Ms;    -- Modeler-defined
  Cheddar_Properties::Fixed_Priority => 2;  -- Modeler-defined
  Compute_Execution_Time => 80 Us .. 81 Us; -- Computed by BoundT
end Plant.i;
```
Conclusion

- Our framework allows evaluation:
  - For scheduling & memory occupation
  - For generic requirements enforcement

Perspectives
- Finer evaluation of PolyORB-HI primitives WCET
- Use the evaluation framework in an optimization process
  - Model based-optimization
Any question?