Model-Based Engineering for the Development of ARINC653 Architectures

Julien Delange
Olivier Gilles
Jérôme Hugues
Laurent Pautet

SAE 2009 AeroTech Congress and Exhibition
Context

- **ARINC653 systems**
  - Time & space isolation across partitions
  - Avoid error propagation, limit damages

- **Rigorous development process**
  - Compliance with certification standards (DO178B)
  - Verify that implementation is compliant with specifications

- **ARINC653 architectures development is still difficult**
  - Manual translation of specifications to implementation
  - Huge costs of verification and certification
Model-Driven Engineering

- Approaches/languages for modeling
- Support for system verification
- Modeling language as source language for implementation

AADL, a modeling language for safety-critical systems

- Safety-critical systems modeling
- Describe hardware and software concerns
- Prone to system analysis
- Used in several projects (AVSI, Flex-eWare, ASSERT, ...)

Addressing development process complexity
**AADL, backbone language for 653 systems**

- **ARINC653 modeling**
  - Partitions representation/specification
  - Time and space isolation modeling

- **System verification**
  - Check system requirements
  - Detect design errors at an early stage in the development process

- **Automatic implementation**
  - Enforce specification requirements
  - Avoid errors of traditional development methods
ARINC653 system modeling with AADL

- **ARINC653 module modeling with processor component**
  - Specify partitions scheduling requirements (major frame, time slots and slots allocation)
  - Describe health monitoring (HM) configuration at module-level

- **Partitions modeling with process and virtual processor**
  - Process components represent partitions content
  - Virtual processor represent partition runtime
  - HM configuration at partition-level
  - Memory requirements (partition size ...)

- **ARINC653 annex of the AADL**
  - Describe modeling patterns
Requirements Enforcement Analysis Language (REAL)

Verification language for the AADL
- Extension of the AADL; annex language
- Integrated in the Ocarina AADL toolsuite

Verification with dedicated theorems
- Formulas to check components specification
- Theorems associated to AADL component
  - i.e: « I check that each process contains at least one thread »

Verification of ARINC653 architectures
- Set of theorems to check ARINC653 architectures with AADL
### Verification patterns example (1)

<table>
<thead>
<tr>
<th>Partition 1</th>
<th>Partition 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>needs 1 MB</td>
<td>needs 10 MB</td>
</tr>
<tr>
<td>Timeslot: 100ms</td>
<td>Timeslot: 200ms</td>
</tr>
</tbody>
</table>

**Partition-level scheduling correctness**
- Verification on each node of the distributed system
- Check that sum of partitions time slots equals the major frame
- Ensure that each partition is executed

**Memory requirements**
- Partitions requirements (partition content < requested size)
- Module requirements (module can allocate enough memory)
**Verification patterns example (2)**

- **Error coverage**
  - Errors that may be raised are handled
  - A recovery procedure is associated
  - The recovery procedure is correct (i.e: a task cannot restart the module)

- **Recovering strategies trade-off**
  - Fault propagation analysis
  - Potential transient or permanent errors
  - Impact between different criticality levels

Diagram:
- Partition 1
  - Criticality A
  - ARINC module
- Partition 2
  - Criticality B

Questions:
- Divide by zero in partition 2?
- Transient impact on partition 1?
- Permanent impact on partition 1?
Automatic implementation of ARINC653 systems

- **Specific toolchain for automatic implementation**
  - Ocarina: AADL toolsuite with code generation facilities
  - POK: ARINC653 OS (BSD-license)

- **Enforce specifications requirements**
  - Specifications as source language
  - Avoid different specification interpretations

- **Improve system confidence**
  - Produce highly-criticaly code (ex: HM or scheduling configuration)
  - Specifications previously verified
From AADL specifications to ARINC653 code

- Generate module and partitions code
  - Configure partitions scheduling
  - Set memory requirements
  - Configure communications

- Compilation with an ARINC653 OS
  - POK, highly configurable OS
  - Automatic configuration from AADL

- Integration of application-level code
  - Potentially from Simulink, Scade ...
Generated code for safety-critical systems

- **Analyze kernel and partitions**
  - Generate minimal code
  - Remove unused services

- **Compliance with safety-critical domain**
  - No dynamic allocation
  - Low complexity algorithms

- **Low memory footprint**
  - Fit with safety-critical requirements
Conclusion

- **AADL, backbone language for ARINC653 systems**
  - Supports modeling, verification and automatic code generation
  - AADL annex for ARINC653 system modeling

- **Improve implementation confidence**
  - Implementation produced from specifications
  - Errors verified at model-level

- **Ongoing work**
  - Automatic code coverage analysis
Thanks for your attention

Visit our AADL portal
http://aadl.telecom-paristech.fr