



# ARINC653 and AADL

Julien Delange [delange@enst.fr](mailto:delange@enst.fr)

Laurent Pautet [pautet@enst.fr](mailto:pautet@enst.fr)



# Background

## ■ ARINC653, avionics standard

- Partitioning support
- Runtime services (communication, ...)

Partition 1

Partition 2

ARINC653 module

## ■ AADLv1 already used for ARINC653 modeling

- Processes as partitions
- Needs improvements, addressed in AADLv2

## ■ ARINC653 annex for the AADLv2

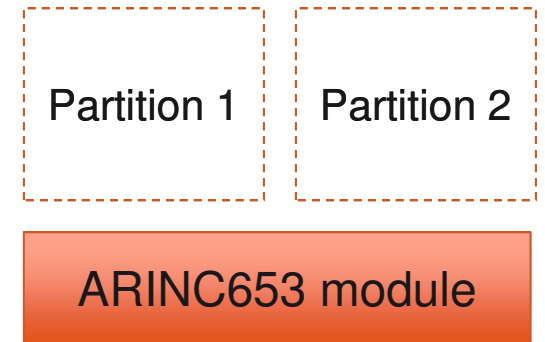
- Common guidelines for ARINC653 modeling
- Extensible to other partitioned architectures



# Background

## ■ ARINC653 deployment with XML file

- Specify module requirements
- Do not provide anything about partitions internals
- Analysis limited to the module
- Code generation for module configuration

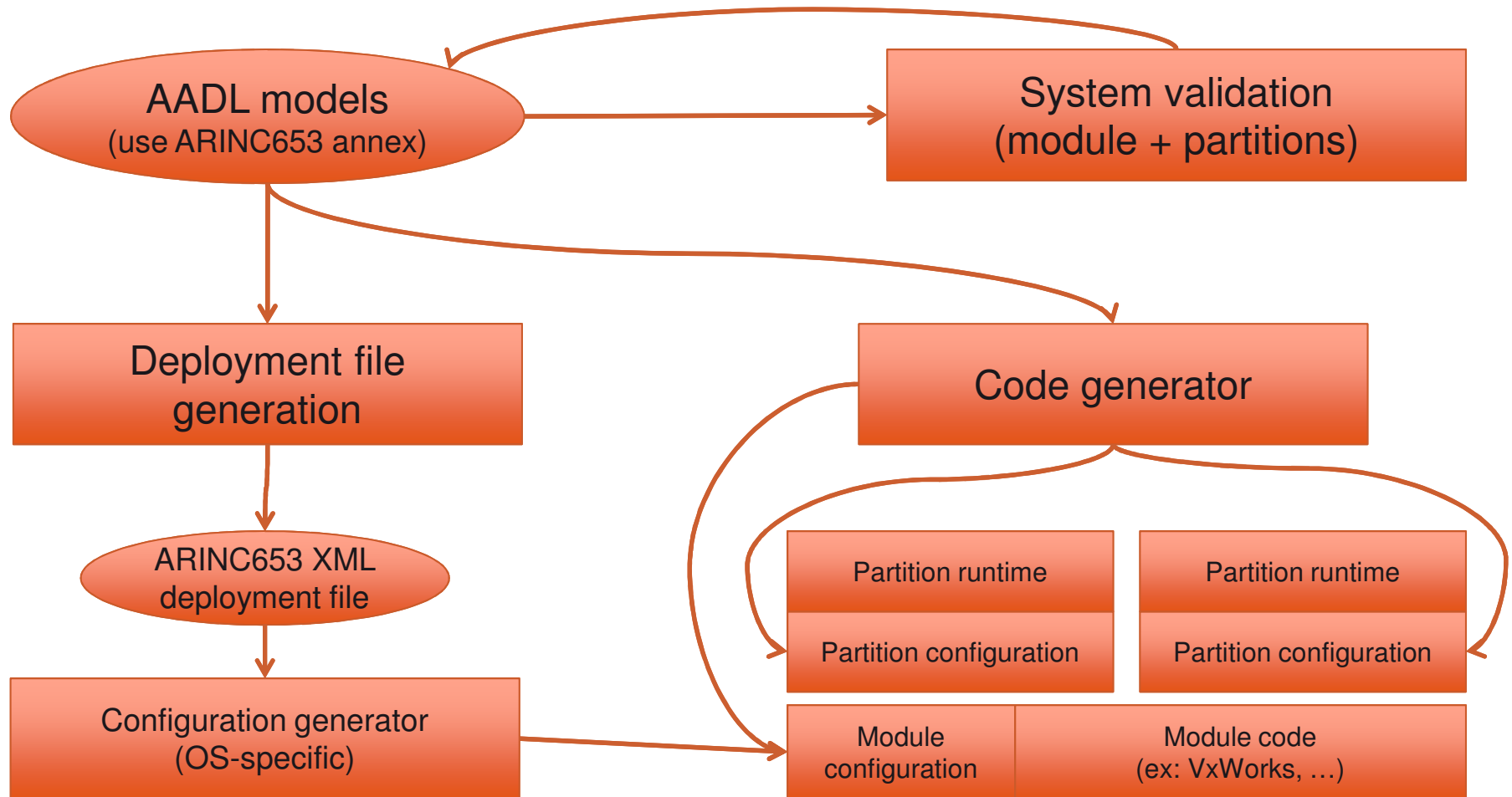


## ■ AADL describes the whole runtime architecture

- Description of module requirements
- Modeling of partition contents and requirements
- Led to a deeper exploitation (analysis and code generation)



# Development of ARINC systems with AADL



# System validation



## ■ Check ARINC653 requirements

- Same idea as OSATE plugins
- Use the REAL language (available in Ocarina)

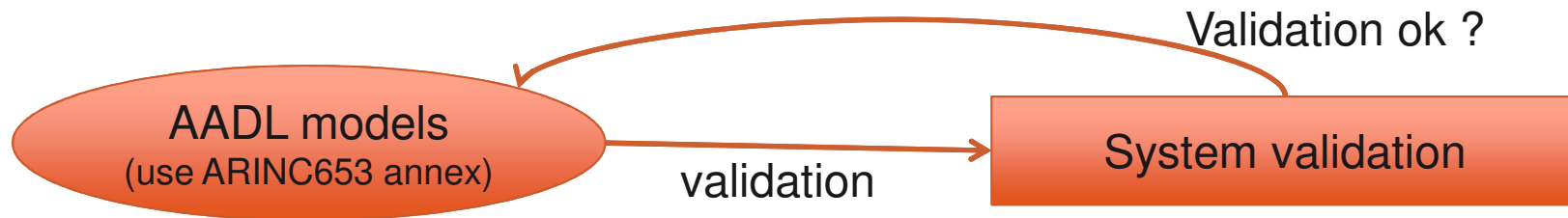
## ■ Scheduling requirements

- Partition scheduling and major time frame correctness

## ■ Memory requirements

- Memory segment is correct regarding partition requirements

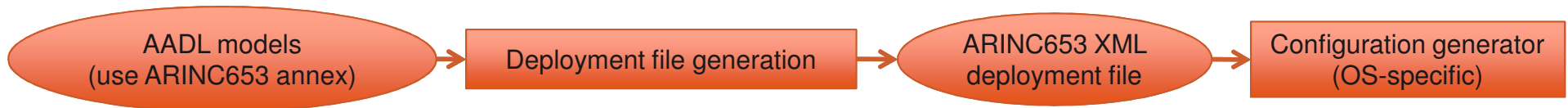
# AADL Validation vs. other techniques



	AADL models	XML file format
Scheduling	Module + Partitions	Module
Memory	Module + Partitions	Non available
Health monitoring	Module + Partitions	Module + Partitions + Processes
Recovering strategies	Module + Partitions	Module + Partitions + Processes
...	...	...

**AADL provides more information about the runtime architecture  
(including processes and their requirements)**

# XML deployment file generation (1)



## ■ Enforce AADL description

- Scheduling, health monitoring, memory requirements, connection table
- Preserve model description

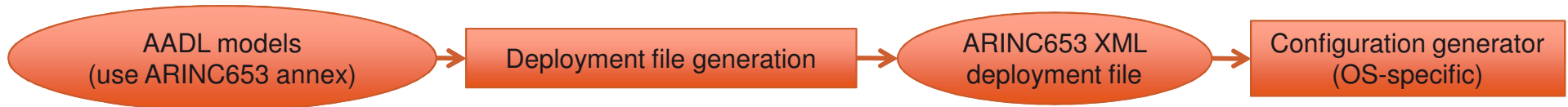
## ■ Improve configuration reliability

- Avoid configuration mistake, make it automatic
- Use validation tools based on ARINC653 XML

## ■ Independence of ARINC653 operating system

- XML to configuration code generator
- Vendor-dependent tool

## XML file generation (2)



### ■ Still work in progress

- Scheduling and Memory requirements work
- Health Monitoring need to be finished

### ■ Support and example

- Generation tool available in Ocarina 2.0w
- Examples available in POK
- More information on <http://aadl.telecom-paristech.fr>

# XML deployment file example

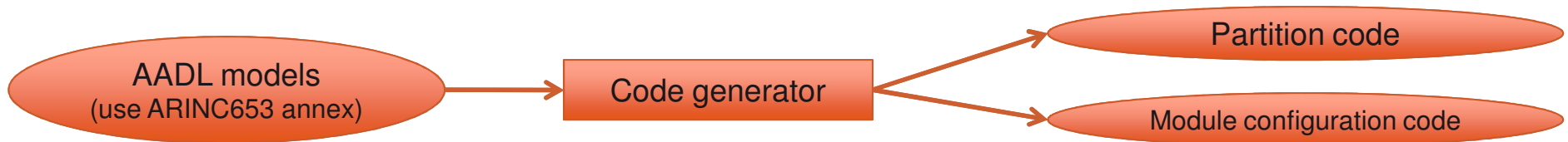
## AADL

```
processor implementation module.impl
subcomponents
  part1 : virtual processor partition.one;
  part2 : virtual processor partition.two;
properties
  ARINC653::Module_Major_Frame => 1000 ms;
  ARINC653::Partition_Slots => (500 ms, 500 ms);
  ARINC653::Slots_Allocation =>
    ( reference (part2), reference (part1));
end module.impl;
```

## XML

```
...
<Module_Schedule MajorFrameSeconds="1.0">
  <Partition_Schedule PartitionName="part2" PartitionIdentifier="1" >
    <Window_Schedule WindowStartSeconds="0.0" WindowIdentifier="0" />
  </Partition_Schedule>
  <Partition_Schedule PartitionName="part1" PartitionIdentifier="2" >
    <Window_Schedule WindowStartSeconds="0.500" WindowIdentifier="1" />
  </Partition_Schedule>
</Module_Schedule>
...
```

# ARINC653 code generation (1)



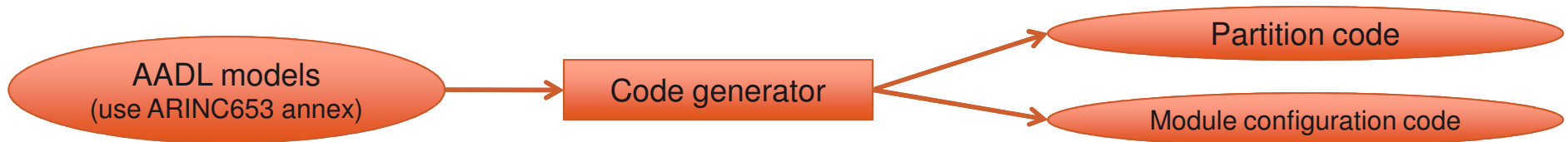
## ■ Generate module configuration code

- Fine configuration of the module
- Automatically add/remove service with respect to AADL description
- More than ARINC XML + configuration generation

## ■ Generate partition code

- Fine tuning of the partition (resources dimensioning and so on)
- Include only required services (avoid memory overhead & dead code)
- Generate data types/accessors, subprograms, partition initialization function and threads periodic activity

## ARINC653 code generation (2)



### ■ Tested with our AADL/ARINC653 runtime, POK

- Support of part1 of the ARINC653 standard (ports, partitions, processes, health monitoring and so on)
- Runtime dedicated to AADL/ARINC653, tunable from AADL models
- Support of ARINC653 APEX
- Freely available on x86 and PowerPC

### ■ Implemented in Ocarina 2.0w

# Generated code of ARINC653 partition, example

## AADL

```
process receive_ping
features
  pdatain : in data port integer
  {ARINC653::Sampling_Refresh_Period => 25 ms;};
end receive_ping;

process implementation receive_ping.impl
subcomponents
  thr : thread receive_ping.impl;
connections
  port pdatain -> thr.datain;
end receive_ping.impl;
```

Code compliant with  
ARINC APEX (API)

## C (partition init)

```
int main ()
{
  PROCESS_ATTRIBUTE_TYPE tattr;
  RETURN_CODE_TYPE ret;
  CREATE_SAMPLING_PORT
    ("pr2_pdatain", sizeof (int), DESTINATION, 25, &(pr2_pdatain_id), &(ret));
  tattr.ENTRY_POINT = thr_job;
  tattr.DEADLINE = 200; tattr.PERIOD = 200; tattr.TIME_CAPACITY = 2;
  CREATE_PROCESS (&(tattr), &(arinc_threads[1]), &(ret));
  SET_PARTITION_MODE (NORMAL, &(ret));
  return (0);
}
```

# Code generation: AADL vx. XML

	AADL	XML
Module configuration		
Partition configuration		
Partition code		

- **Benefits of the whole runtime description**
- **Same reasons than for validation**



# Conclusion

## ■ AADL as a central backbone for ARINC653 systems

- Model validation (everything is fine in my model)
- Configuration generation (ensure that configuration follows validated artifacts)
- Code generation (ensure that partitions code follows validated artifacts)

## ■ Tool support

- Ocarina model validation & ARINC653 code generator
- POK AADL/ARINC653 runtime

## ■ Need to test with industrial case studies