Model-Based Design, Automated Code Generation and Safety Analysis of ARINC653 Architectures using the AADL

Jérôme Hugues, ISAE

Julien Delange, CMU/SEI

Software Engineering Institute

Carnegie Mellon

isae

Institut Supérieur de l'Aéronautique et de l'Espace

Introduction

> One fault instance of an ADIRU (Air Data Inertial Reference Unit) on-board a Boeing 777-2H6ER caused an hazardous accident to Malaysian Air flight 124 in 2005,

> Key question is: could we avoid similar scenario in future system design? How? Associated cost?

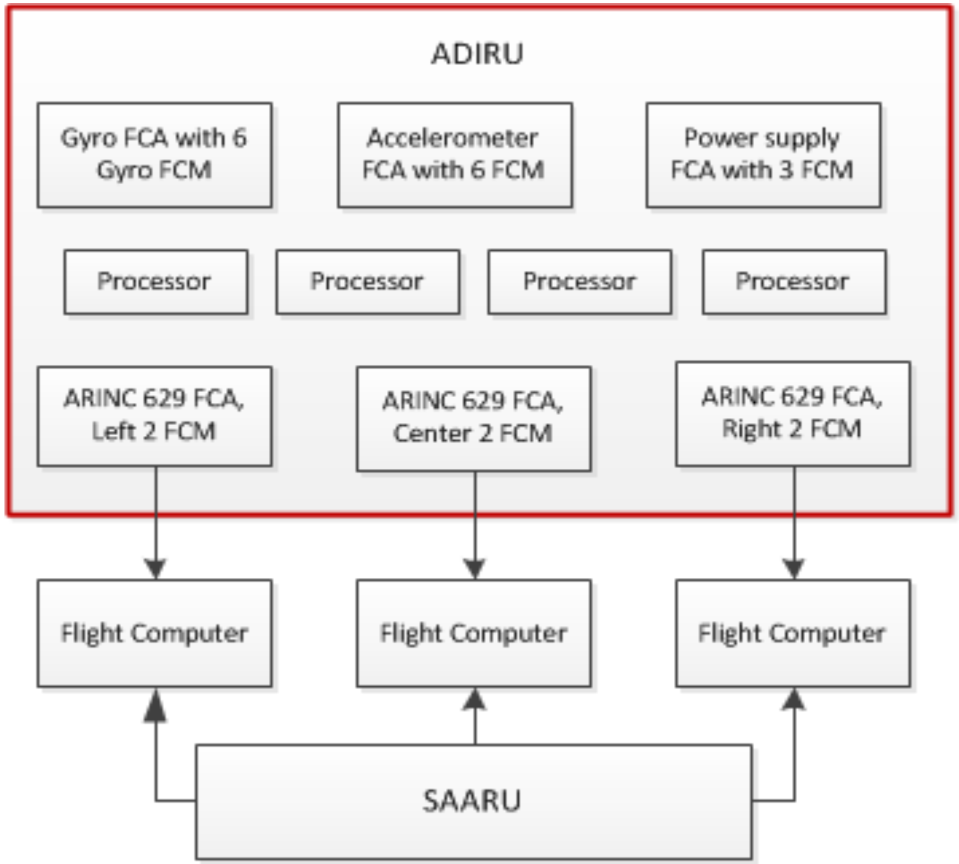
> Failure has been (partially) described in publically available reports by NTSB, and Vanderbilt University, used for study

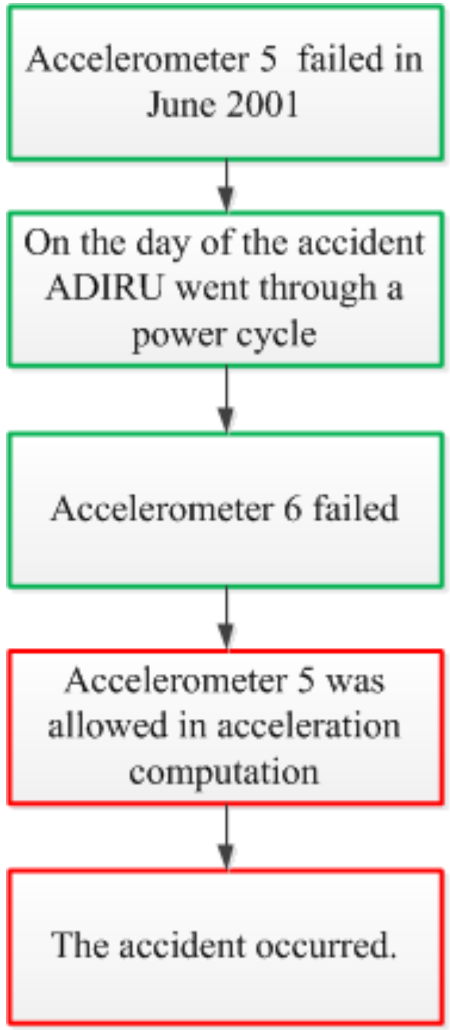
Agenda

1. How to capture architecture key elements using AADL
   - Real-time architecture, ARINC653 patterns, etc.
2. Link them to implementation artifacts
   - Simulation through code generation
3. Trace them w.r.t. safety analysis objectives

About Boeing 777-2H6ER ADIRU

> (from ATSB report 200503722)



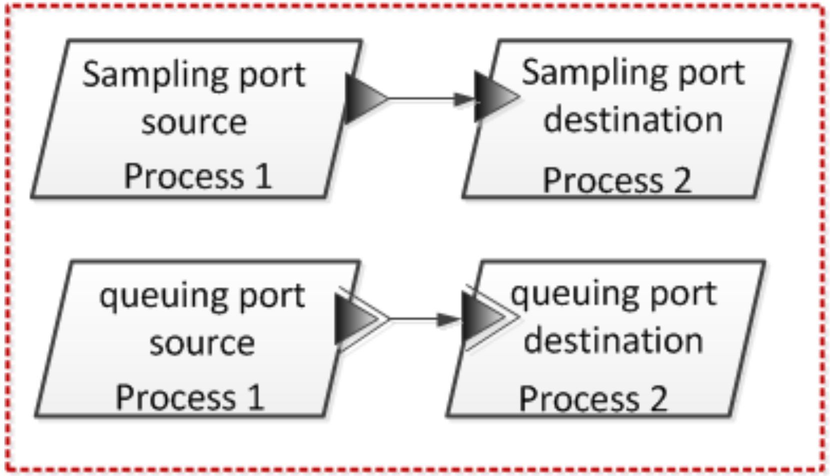


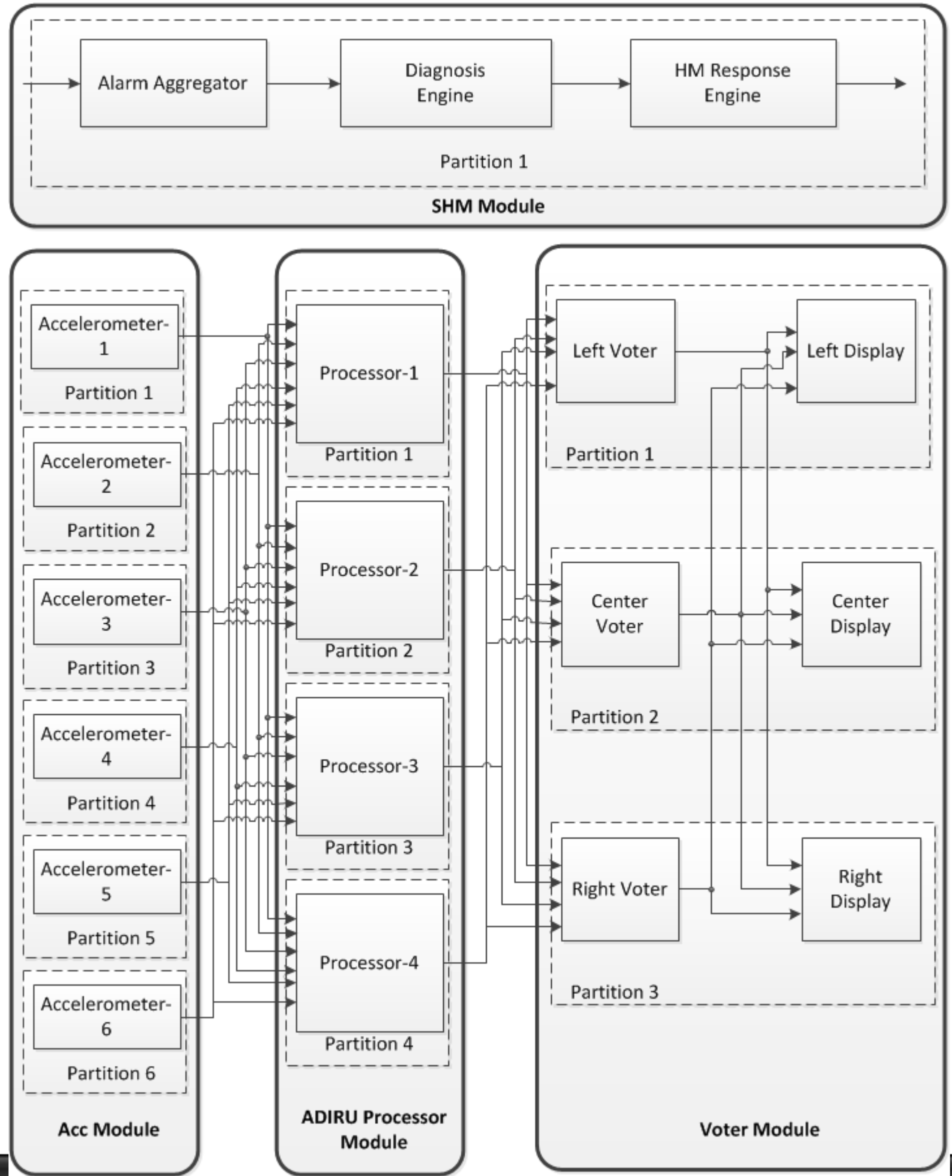
- Multiple levels of redundancy.
- work without maintenance with one fault in each FCA.

The Model in ARINC 653 Architecture

> **ISIS-11-101 TR by Vanderbilt Univ.**

- Four modules
- Two types of ports





Outline

1. **Capturing architecture key elements using AADL**
   - Real-time architecture, ARINC653 patterns, etc.
2. **Link them to implementation artifacts**
   - Simulation through code generation
3. **Trace them w.r.t. safety analysis objectives**

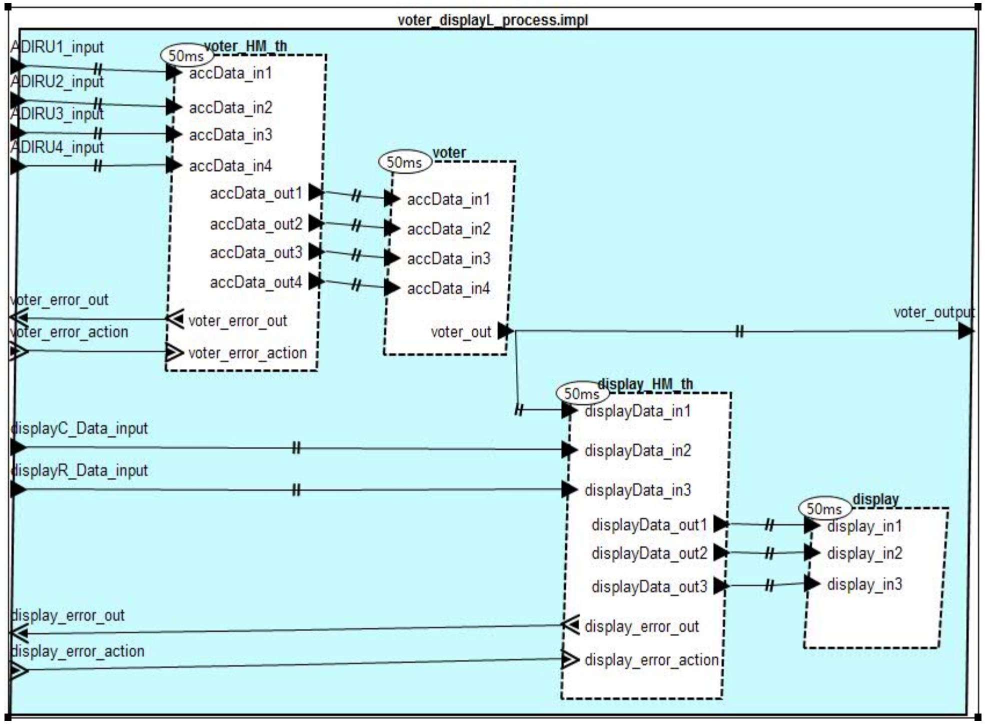
AADL: Architecture Analysis & Design Language

- **International standard promoted by SAE, AS-2C committee**
  - Released as AS5506 family of standards
  - Version 1.0 (2004), version 2 (2009), 2.1 (2012)
  - Based on feedback from the aerospace industry
- **Annex document to address specific needs**
  - ARINC653, Behavior, data, error modeling, code generation, …
- **AADL objectives are "to model a system"**
  - With analysis in mind
  - To ease transition from well-defined requir
    final system : code production
- **Require semantics => any AADL entity has a semantics (natural language or formal methods).**

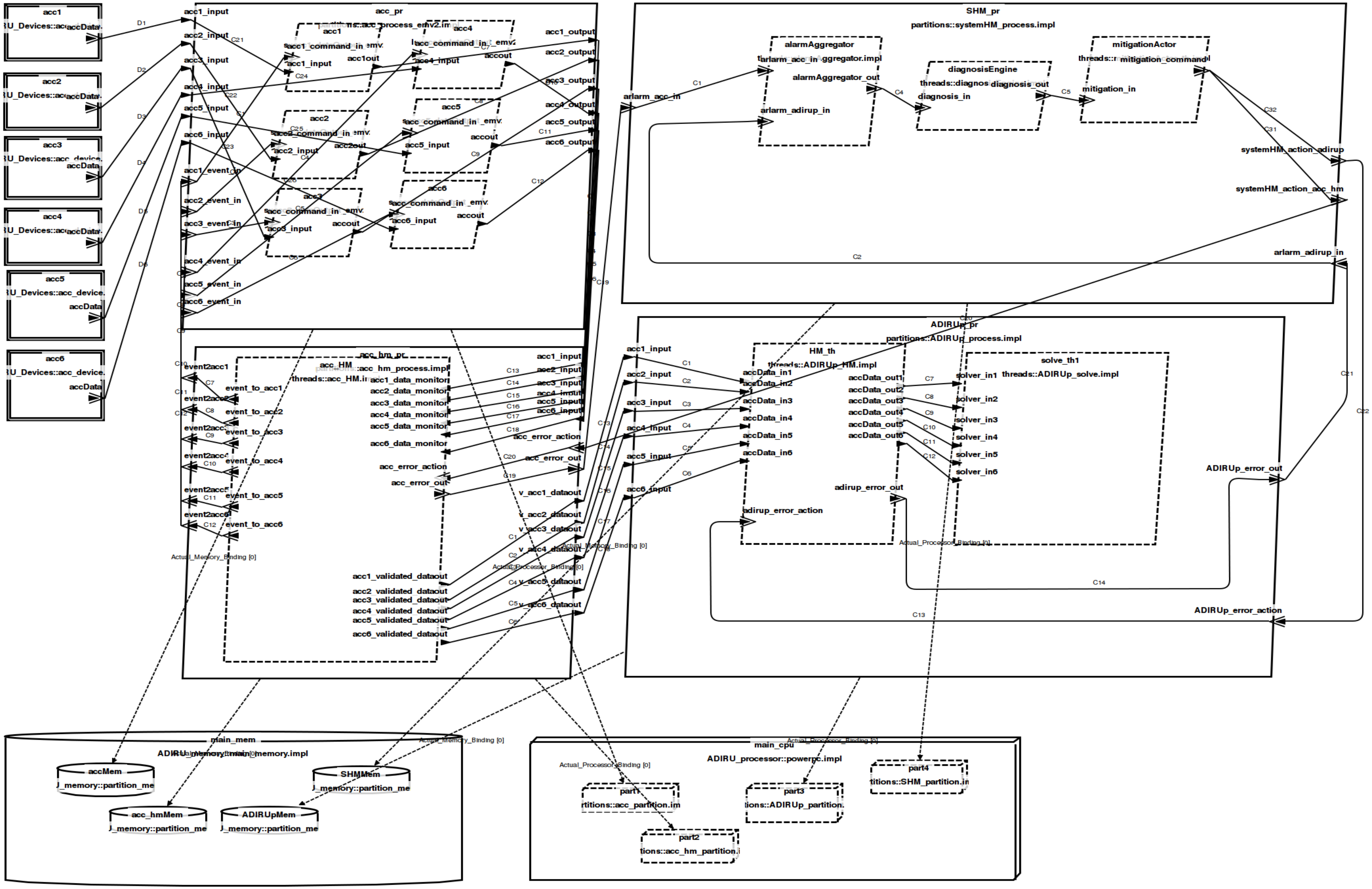
Modeling of the ADIRU with AADL

> **Regular modeling process**
  - » Define sub-system boundaries, interfaces, configuration
  - » Mixing text, graphics, property editor to manage model complexity

Properties	Thread voter	Thread voter_HM	Thread display	Thread display_HM
Dispatch_Protocol	Period	Period	Period	Period
Priority	1	2	3	4
Period	50 ms	50 ms	50 ms	50 ms
Compute_Execution_Time	20 ms..50 ms	20 ms..50 ms	20 ms..50 ms	20 ms..50 ms
Deadline	50 ms	50ms	50ms	50ms



Overview of the AADL model



First level of analysis: core and plug-ins

> **AADL default semantics check**
  » Containment hierarchy, applicability of configuration parameters (units, types, etc), types of message exchanged, port connection, etc.

> **ARINC 653 verification plugs-ins**
  » Part of AADL eco-system: OSATE, MASIW, Ocarina, …
  » Check connections
  » Validity of ARINC653 Configuration parameters:
    - Major Frame Correctness, Properties of Memory Components, Dimensioning of Memory Components, Partitions Bindings, Partitions Executions, Separation of Memory
  » Additional checks: constraints set by RTOS vendors, e.g. alignment of memory segments, max number of threads, ports, size of queues, etc.

Outline

1. **Capturing architecture key elements using AADL**
   - **Real-time architecture, ARINC653 patterns, etc.**
2. **Link them to implementation artifacts**
   - Simulation through code generation
3. **Trace them w.r.t. safety analysis objectives**

AADL and XML configuration data

> **ARINC653 Executives require an additional configuration file, but …**

> **A (full) AADL model must define all components**

» For analysis or code generation purposes

> **Can derive configuration file from the AADL model**

» Implemented in Ocarina, targets DeOS and VxWork653

> **Part of the model bus philosophy**

» One repository that can be mined for various purposes

» Analysis, code generation, management of configuration parameters

AADL: modeling data types

```
-- Part of the Annex D – Data Modeling Annex

data C_Unsigned_Long_Int
 --  This data component defines a C unsigned long int type, with a
 --  dual nature The first properties defines its representation in
 --  memory, the two last its mapping in C.
properties
 Data_Model::Data_Representation => integer;
 Data_Model::Number_Representation => unsigned;
 Data_Size => 4 bytes;
 Source_Language => (C);
 Type_Source_Name => "unsigned long int";
end C_Unsigned_Long_Int;

data accData extends C_Unsigned_Long_Int
end accData;

subprogram acc1_dataOutput_spg
features
   acc1DataOut: out parameter SHM_DataType::accData;
   event_in:   in parameter SHM_DataType::actionData;
end acc1_dataOutput_spg;
```

AADL and subprograms

> Binding code to AADL components

```
subprogram acc1_dataOutput_spg
features
  acc1DataOut: out parameter SHM_DataType::accData;
  event_in:   in parameter SHM_DataType::actionData;
properties
  Source_Language => (C);
  Source_Name =>"acc1dataoutput";
  Source_Text => ("../../../acc_code.o");
end acc1_dataOutput_spg;
```

> Mapping from AADL model to code

```
subprogram acc1_dataOutput_spg
features
  acc1DataOut: out parameter SHM_DataType::accData;
  event_in:   in parameter SHM_DataType::actionData;
end acc1_dataOutput_spg;
```

```
void acc1_dataOutput_spg ( /* C */
(acc1DataOut *SHM_DataType_accData,
 event_in:   SHM_DataType_actionData);
```

AADL and code generation

> **The AADL architecture has all details about**
  - » Task, queues, buffers, etc.
  - » Used for schedulability analysis, generation of ARINC653 configuration

> **Ocarina: massive code generation**
  - » Take advantage of global knowledge to optimize code, and generate only what is required
  - » Reduce as much as possible error-prone and tedious tasks

> **Targets DeOS and VxWorks 653**

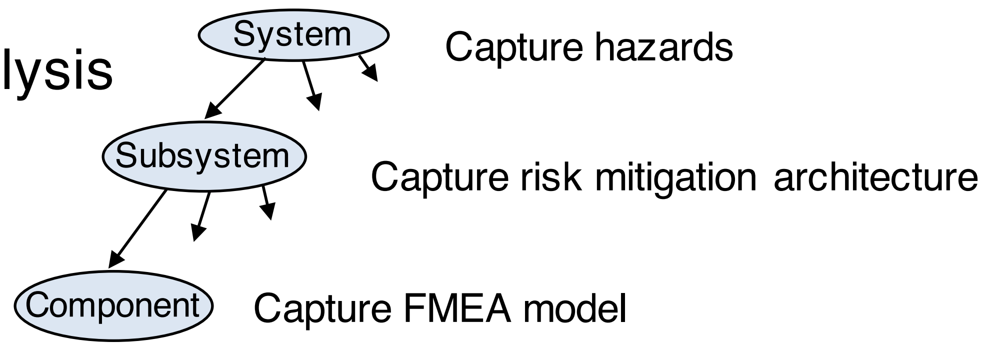
- See all demos and videos from http://aadl.info/aadl/demo-arinc653/

Outline

1. **Capturing architecture key elements using AADL**
   - **Real-time architecture, ARINC653 patterns, etc.**
2. **Link them to implementation artifacts**
   - Simulation through code generation
3. **Trace them w.r.t. safety analysis objectives**

AADL Error Model Scope and Purpose

- **System safety process uses many individual methods and analyses, e.g.**
  - hazard analysis
  - failure modes and effects analysis
  - fault trees
  - Markov processes



SAE ARP 4761 *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*

- **Related analyses are also useful for other purposes, e.g.**
  - maintainability
  - availability
  - Integrity

Annotated architecture model permits checking for consistency and completeness between these various declarations.

- **Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.**

Automation of SAE ARP4761 System Safety Assessment Practice

AADL & EMV2

FHA

Spreadsheet

Uses error sources

Component	Error	Hazard Description	Crossrefer	Functional Failure	Operational P
StabilatorPositionSe	"ServiceOmission o	"No stabilator position readings due to s	"1.1.3"	"Loss of sensor readings"	"all"
StabAct1	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabAct2	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"

FMEA

Spreadsheet

Uses error flows & propagations

Three CPU FMEA

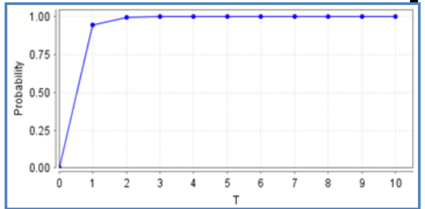
Date: 12/1/2010 Author:

Item	Initial State	Initial Failure	1st Level Effect	Transition	2nd Level Effect
CPU_1.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_L	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_2.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_R	ErrorFree		ErrorFree		ErrorFree
PR_FGS_L1	ErrorFree		ErrorFree		ErrorFree
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree
CPU_1.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_L	ErrorFree		ErrorFree		ErrorFree
CPU_2.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_R	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
PR_FGS_L1	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree

Markov Chain

PRISM

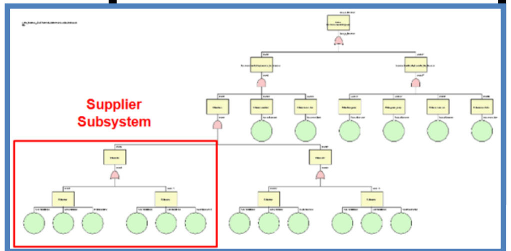
Uses error flows & behavior



FTA

CAFTA, OpenFTA

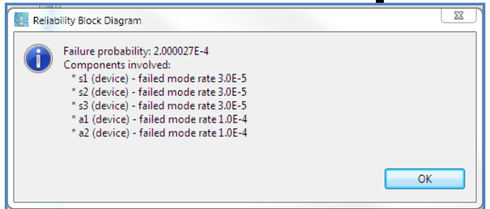
Uses composite error behavior



RBD/DD

OSATE plugin

Uses composite error behavior



Annotating the model with Error Information (1)

```
device implementation acc_device.impl
annex EMV2
 {**
   use types ADIRU_errLibrary;
   use behavior ADIRU_errLibrary::simple;

   error propagations
     accData : out propagation{ValueErroneous};
   flows
     f1 : error source accData{ValueErroneous} when failed;
   end propagations;

    properties
      EMV2::hazards =>
      ([   crossreference => "N/A";
           failure => "Accelerometer value error";
           phases => ("in flight");
           description => "Accelerometer starts to send an erroneous value";
           comment => "Can be critical if not detected by the health monitoring";
      ])
      applies to accData.valueerroneous;

      EMV2::OccurrenceDistribution => [ ProbabilityValue => 3.4e-5 ; Distribution => Fixed;]
      applies to accData.valueerroneous;
 **};
 end acc_device.impl;
```

Declaring error sources

Documenting the error

Annotating the model with Error Information (2)

```
process implementation acc_process_emv2.impl extends acc_process.impl
subcomponents
  -- We extend the initial implementation, and add error modeling elements.

  acc1: refined to thread threads::acc1_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };
  acc2: refined to thread threads::acc2_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };
  acc3: refined to thread threads::acc3_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };
  acc4: refined to thread threads::acc4_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };
  acc5: refined to thread threads::acc5_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };
  acc6: refined to thread threads::acc6_dataOutput_emv2.impl
    { Classifier_Substitution_Rule => Type_Extension; };

connections
  C21 : port acc1_input -> acc1.acc1_input;
  C22 : port acc2_input -> acc2.acc2_input;
  C23 : port acc3_input -> acc3.acc3_input;
  C24 : port acc4_input -> acc4.acc4_input;
  C25 : port acc5_input -> acc5.acc5_input;
  C26 : port acc6_input -> acc6.acc6_input;

annex EMV2{**
  use types ADIRU_errLibrary;
  use behavior ADIRU_errLibrary::simple;

  error propagations
    acc1_input  : in  propagation{ValueErroneous};
    acc1_output : out propagation{ValueErroneous};
    acc2_input  : in  propagation{ValueErroneous};
    acc2_output : out propagation{ValueErroneous};
    acc3_input  : in  propagation{ValueErroneous};
    acc3_output : out propagation{ValueErroneous};
    acc4_input  : in  propagation{ValueErroneous};
    acc4_output : out propagation{ValueErroneous};
    acc5_input  : in  propagation{ValueErroneous};
    acc5_output : out propagation{ValueErroneous};
    acc6_input  : in  propagation{ValueErroneous};
    acc6_output : out propagation{ValueErroneous};
  flows
    f1 : error path acc1_input{ValueErroneous} -> acc1_output{ValueErroneous};
    f2 : error path acc2_input{ValueErroneous} -> acc2_output{ValueErroneous};
    f3 : error path acc3_input{ValueErroneous} -> acc3_output{ValueErroneous};
    f4 : error path acc4_input{ValueErroneous} -> acc4_output{ValueErroneous};
    f5 : error path acc5_input{ValueErroneous} -> acc5_output{ValueErroneous};
    f6 : error path acc6_input{ValueErroneous} -> acc6_output{ValueErroneous};
  end propagations; **};
end acc_process_emv2.impl;
```

Passing the error directly through components features

Annotating the model with Error Information (3)

```
annex EMV2{**
 use types ADIRU_errLibrary;
 use behavior ADIRU_errLibrary::simple;

 error propagations
   acc1_input : in propagation{ValueErroneous};
   acc2_input : in propagation{ValueErroneous};
   acc3_input : in propagation{ValueErroneous};
   acc4_input : in propagation{ValueErroneous};
   acc5_input : in propagation{ValueErroneous};
   acc6_input : in propagation{ValueErroneous};
 flows
   f1 : error sink acc1_input{ValueErroneous};
   f2 : error sink acc2_input{ValueErroneous};
   f3 : error sink acc3_input{ValueErroneous};
   f4 : error sink acc4_input{ValueErroneous};
   f5 : error sink acc5_input{ValueErroneous};
   f6 : error sink acc6_input{ValueErroneous};
 end propagations;

 component error behavior
 transitions
     t1 : operational -[acc1_input{ValueErroneous}]-> failed;
     t2 : operational -[acc2_input{ValueErroneous}]-> failed;
     t3 : operational -[acc3_input{ValueErroneous}]-> failed;
     t4 : operational -[acc4_input{ValueErroneous}]-> failed;
     t5 : operational -[acc5_input{ValueErroneous}]-> failed;
     t6 : operational -[acc6_input{ValueErroneous}]-> failed;
 detections
    operational -[1 ormore(acc1_input{ValueErroneous})]-> acc_error_out!;
    operational -[1 ormore(acc2_input{ValueErroneous})]-> acc_error_out!;
    operational -[1 ormore(acc3_input{ValueErroneous})]-> acc_error_out!;
    operational -[1 ormore(acc4_input{ValueErroneous})]-> acc_error_out!;
    operational -[1 ormore(acc5_input{ValueErroneous})]-> acc_error_out!;
    operational -[1 ormore(acc6_input{ValueErroneous})]-> acc_error_out!;
 end component;
**};
```

Receiving a erroneous value makes the component to fail

EMV2 at work

> **Functional Hazard Assessment:**

- List all potential error sources, Include documentation from the model

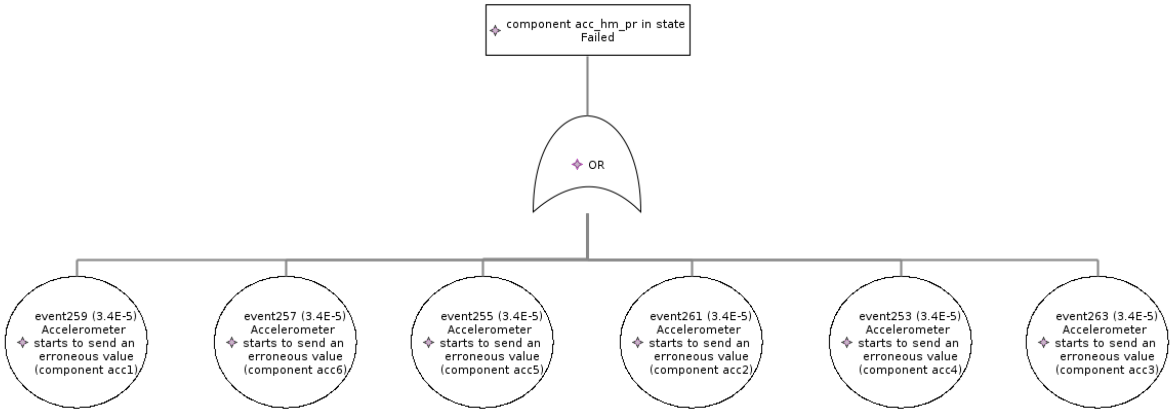
Component	Error	Hazard Description	ossreferer	Functional Failure	Operational Phases	Comment
acc1	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"
acc2	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"
acc3	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"
acc4	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"
acc5	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"
acc6	"ValueErroneous on accData"	"Accelerometer starts to send an erroneous value"	"N/A"	"Accelerometer value error"	"in flight"	"Can be critical if not detected by the health monitoring"

> **Fault Impact Analysis**

- Bottom-up approach, Trace the error flow defined in the architecture

Component	Initial Failure Mode	1st Level Effect	Failure Mode	second Level Effect	Failure Mode
acc1	Failed	{ValueErroneous} accData -> acc_pr:acc1_input	acc_pr {ValueErroneous}	{ValueErroneous} acc1_output -> acc_hm_pr:acc1_input	acc_hm_pr {ValueErroneous} [Masked]
acc2	Failed	{ValueErroneous} accData -> acc_pr:acc2_input	acc_pr {ValueErroneous}	{ValueErroneous} acc2_output -> acc_hm_pr:acc2_input	acc_hm_pr {ValueErroneous} [Masked]
acc3	Failed	{ValueErroneous} accData -> acc_pr:acc3_input	acc_pr {ValueErroneous}	{ValueErroneous} acc3_output -> acc_hm_pr:acc3_input	acc_hm_pr {ValueErroneous} [Masked]
acc4	Failed	{ValueErroneous} accData -> acc_pr:acc4_input	acc_pr {ValueErroneous}	{ValueErroneous} acc4_output -> acc_hm_pr:acc4_input	acc_hm_pr {ValueErroneous} [Masked]
acc5	Failed	{ValueErroneous} accData -> acc_pr:acc5_input	acc_pr {ValueErroneous}	{ValueErroneous} acc5_output -> acc_hm_pr:acc5_input	acc_hm_pr {ValueErroneous} [Masked]
acc6	Failed	{ValueErroneous} accData -> acc_pr:acc6_input	acc_pr {ValueErroneous}	{ValueErroneous} acc6_output -> acc_hm_pr:acc6_input	acc_hm_pr {ValueErroneous} [Masked]

> **Fault Tree**



Conclusion

- AADLv2 leveraged to model the ADIRU system
  - Full architectural description of the avionics system
  - Link with consistency checks for ARINC653 patterns
  - Code generation towards ARINC653 APEX
  - Safety analysis using the AADL EMV2 annex

- AADL ecosystem provide all required tools, using OSATE2 and Ocarina, completed with spreadsheets, FTA tool and target RTOS

- Future work will consider connection with requirement engineering, and better coverage of faulty scenarios