CPSC 875

John D. McGregor

Interface Design

- The next few slides are from the Autosar architecture documentation:

Document ID 053 : AUTOSAR_EXP_LayeredSoftwareArchitecture

- Despite the Autosar cofidential tag on each slide these pages are from the publicly available website: www.autosar.org

Interface

An **Interface (interface module)** contains the functionality to abstract from modules which are architecturally placed below them. E.g., an interface module which abstracts from the hardware realization of a specific device. It provides a generic API to access a specific type of device independent on the number of existing devices of that type and independent on the hardware realization of the different devices.

The interface does not change the content of the data.

In general, interfaces are located in the **ECU Abstraction Layer.**

Example: an interface for a CAN communication system provides a generic API to access CAN communication networks independent on the number of CAN Controllers within an ECU and independent of the hardware realization (on chip, off chip).

Handler

A **handler** is a specific interface which controls the concurrent, multiple and asynchronous access of one or multiple clients to one or more drivers. I.e. it performs buffering, queuing, arbitration, multiplexing.

The handler does not change the content of the data.

Handler functionality is often incorporated in the driver or interface (e.g. SPIHandlerDriver, ADC Driver).

Manager

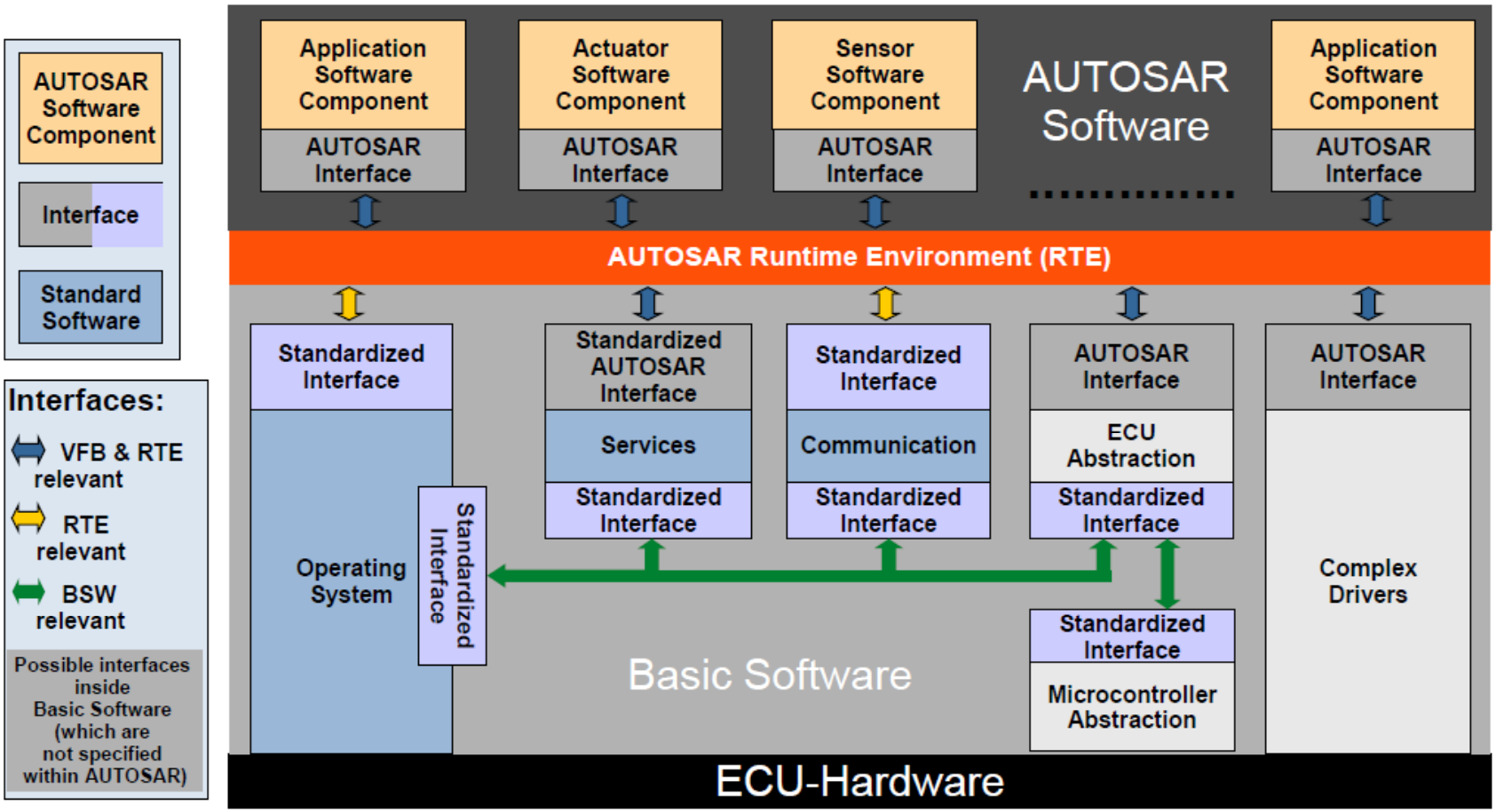
A **manager** offers specific services for multiple clients. It is needed in all cases where pure handler functionality is not enough to abstract from multiple clients.

Besides handler functionality, a manager can evaluate and change or adapt the content of the data.

In general, managers are located in the **Services Layer**

Example: The NVRAM manager manages the concurrent access to internal and/or external memory devices like flash and EEPROM memory. It also performs distributed and reliable data storage, data checking, provision of default values etc.

AUTOSAR Interface	An "AUTOSAR Interface" defines the information exchanged between software components and/or BSW modules. This description is independent of a specific programming language, ECU or network technology. AUTOSAR Interfaces are used in defining the ports of software-components and/or BSW modules. Through these ports software-components and/or BSW modules can communicate with each other (send or receive information or invoke services). AUTOSAR makes it possible to implement this communication between Software-Components and/or BSW modules either locally or via a network.
Standardized AUTOSAR Interface	A "Standardized AUTOSAR Interface" is an "AUTOSAR Interface" whose syntax and semantics are standardized in AUTOSAR. The "Standardized AUTOSAR Interfaces" are typically used to define AUTOSAR Services, which are standardized services provided by the AUTOSAR Basic Software to the application Software-Components.
Standardized Interface	A "Standardized Interface" is an API which is standardized within AUTOSAR without using the "AUTOSAR Interface" technique. These "Standardized Interfaces" are typically defined for a specific programming language (like "C"). Because of this, "standardized interfaces" are typically used between software-modules which are always on the same ECU. When software modules communicate through a "standardized interface", it is NOT possible any more to route the communication between the software-modules through a network.



Note: This figure is incomplete with respect to the possible interactions between the layers.

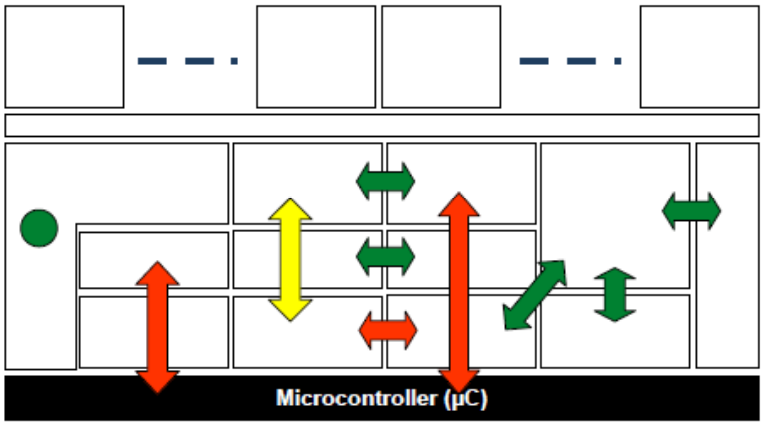
Horizontal Interfaces

Services Layer: horizontal interfaces are allowed
Example: Error Manager saves fault data using the NVRAM manager

ECU Abstraction Layer: horizontal interfaces are allowed

A complex driver may use selected other BSW modules

µC Abstraction Layer: horizontal interfaces are not allowed. Exception: configurable notifications are allowed due to performance reasons.



Vertical Interfaces

One Layer may access all interfaces of the SW layer below

Bypassing of one software layer should be avoided

Bypassing of two or more software layers is not allowed

Bypassing of the µC Abstraction Layer is not allowed

A module may access a lower layer module of another layer group (e.g. SPI for external hardware)

All layers may interact with system services.

This matrix shows the possible interactions between AUTOSAR Basic Software layers

✓ "is allowed to use"
× "is not allowed to use"
Δ "restricted use (callback only)"

The matrix is read **row-wise**:
Example: "I/O Drivers are allowed to use System Services and Hardware, but no other layers".

(gray background indicates "non-Basic Software" layers)

uses

	System Services	Memory Services	Communication Services	Complex Drivers	I/O Hardware Abstraction	Onboard Device Abstraction	Memory Hardware Abstraction	Communication Hardware Abstraction	Microcontroller Drivers	Memory Drivers	Communication Drivers	I/O Drivers
AUTOSAR SW Components / RTE	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
System Services	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓
Memory Services	✓	✓	×	×	×	×	✓	×	×	×	×	×
Communication Services	✓	✓	✓	×	×	×	×	✓	×	×	×	×
Complex Drivers	restricted access -> see the following two slides											
I/O Hardware Abstraction	✓	×	×	×	✓	✓	×	✓	✓	×	✓	✓
Onboard Device Abstraction	✓	×	×	×	×	✓	×	✓	✓	×	✓	✓
Memory Hardware Abstraction	✓	✓	×	×	×	✓	✓	✓	×	✓	✓	×
Communication Hardware Abstraction	✓	×	✓	×	×	✓	×	✓	×	×	✓	✓
Microcontroller Drivers	✓	×	×	×	✓	✓	×	×	Δ	×	×	Δ
Memory Drivers	✓	×	×	×	×	×	✓	×	×	×	×	×
Communication Drivers	✓	×	×	×	×	✓	×	✓	×	×	×	✓
I/O Drivers	✓	×	×	×	✓	✓	×	×	Δ	×	×	Δ

Session Objective

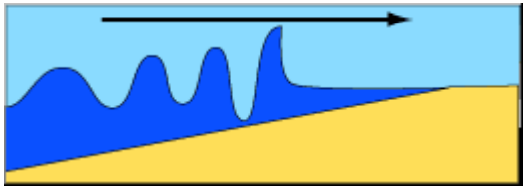
- To explore the design of interfaces among components and systems

Interfaces

- An interface is “a thin layer or boundary between two different substances or entities.”
- At interfaces things change, the possibilities of inconsistencies arise, information can leak.
- Hardware, software components, and humans come together at interfaces.
- But more generally …

Waves on a beach

- **In deep water** the speed (or velocity) of a water wave depends only on its wave length. Specifically, the speed is proportional to the square root of the wavelength. Thus, the longer the wave length, the faster the wave, or vice versa. The speed of a single wave is called the phase speed. Amazingly, the speed of a packet of waves (the group speed) is often not the same.
- The speed of waves of different wave lengths in deep water. "Deep" in this context is not an absolute value, but is relative to wave length. The simple relationship starts to breakdown when the depth of the water is less than 1/4 the wave length. At that depth the bottom exerts sufficient drag on the wave to slow its motion and thus decrease the wavelength.
- **Decreasing speed of waves as water becomes shallow has dramatic consequences on the beach.** As the waves slow, their profile is laterally compressed and since each wave must carry the same energy it becomes higher. As the wave approaches shore this process continues until the **height exceeds 1/7 the wave length and the wave becomes unstable**. Then the wave breaks.



http://www.owrc.com/waves/waves.html

Reflections

- **When waves approach a shore with a gradual depth profile**, e.g. a beach, most of the wave energy is absorbed by the breaking of waves.
- **When waves approach a hard vertical surface,** e.g. a concrete wall or a dock, most of the energy is converted into waves moving in the opposite direction, a reflection. Of course the reflected waves are superimposed onto the original waves, and the two sets of wave moving in opposite directions can create a real mess.

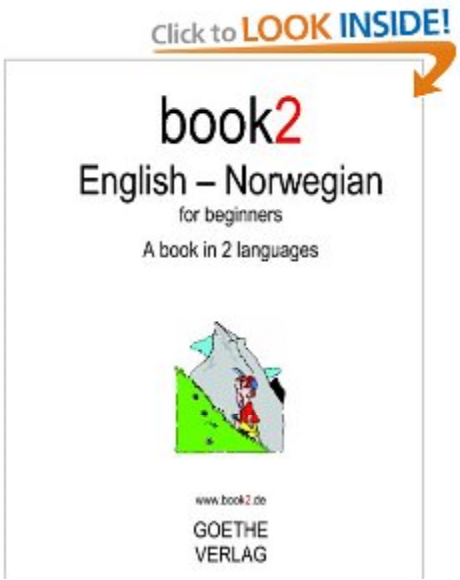
Merging traffic

- Traffic signals, road signs, and lane markings are the interface between drivers

Between people with different languages

- Translators and translation form the interface between people with different languages

Google Translate BETA



International boundaries

- The interface between countries may be simple or complex
- The boundaries may be virtual, e.g. US/Brazil
- The visa requirements on each side are part of the interface.

Human-Computer interfaces

- Between the human and the machine
- Mismatch in speed
- Analogies to the real world

Electrical adapters/converters

- This interfaces between universal and American plugs. It changes the arrangement but not the voltage.

- This interfaces between two different levels of voltage.

Interface terminology

- By ‘interface terminology’, we refer to a set of terms resulting from the interconnection of disciplines which are not normally perceived as contiguous

Interface terminology - 2

- As sciences are evolving at a rapid pace, the need for interface terminology will be felt more and more acutely by the specialists and users concerned. As regards environmental economics, interface terminology cannot be viewed as a no man's land between the two disciplines (Figure 2*a); actually, it refers to a common* ground which has become part of each of the original disciplines, enriching it by broadening its scope, as illustrated in Figure 2*b:*
- Let us be clear about the notion of common ground: it remains to be seen whether 'common' is to be taken as an indication that the interface terminology will be exactly the same in all situations. This question will be examined later; for the moment, we shall assume that 'common' designates an existing link between the two disciplines. Both economics and environmental studies include an additional field which allows them to consider present-day problems from a new angle.

Interface terminology - 3

One way to think of an interface is as not belonging to either element being composed.

Fig. 2*a*: Interface as a no man's land

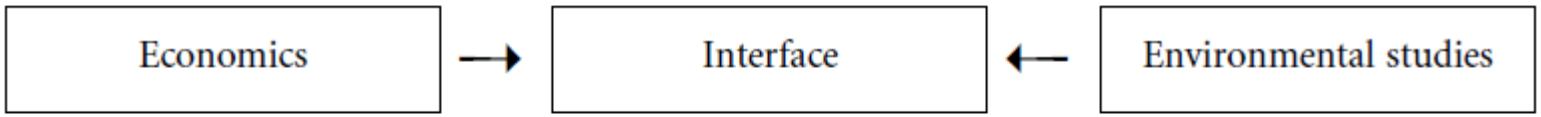
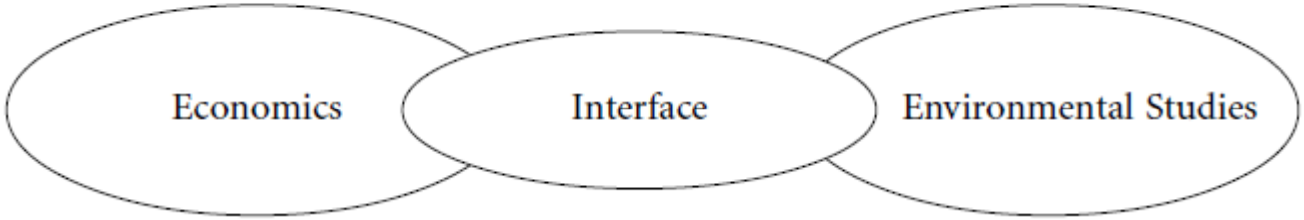


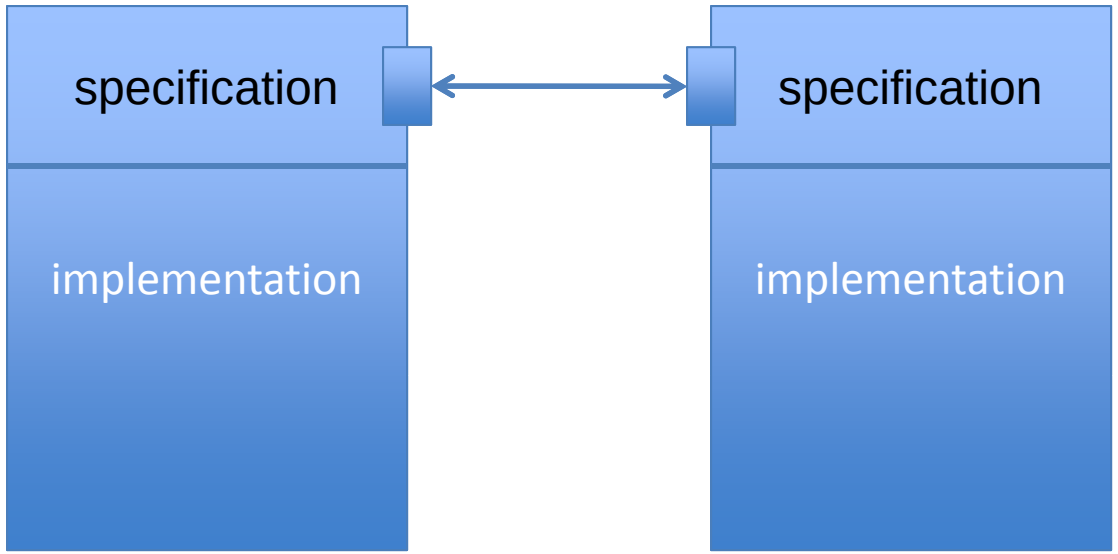
Fig.2*b*: Interface as a common ground



Another way to think of it is as uniting the two elements and overlapping them both.

Interface terminology - 4

- The specifications and the state machine that describes their interaction are the interface.



UART to SPI interface

- universal asynchronous receiver/transmitter (UART) to serial peripheral interface (SPI).
- The concept of a “port” is used to separate the interface from an implementation.

Table 5 • **Port Descriptions**

Port	Direction	Description
NRESET	Input	Active Low reset signal aborts current operation and resets all the control signals.
CLK	Input	Input clock of 20 MHz
RXD	Input	UART serial input
TXD	Output	UART serial output
SPI_OR_MEM	Input	This bit selects single-byte or multiple-byte slave select. If SPI_OR_MEM = 1, the slave select will be enabled for single byte. If SPI_OR_MEM = 0, the slave select will be enabled for multiple bytes.
SCLK	Output	Output clock to the SPI slave. Frequency depends upon the control bit settings.
MOSI	Output	Master out slave in
MISO	Input	Master in slave out
SS_N [7:0]	Output	Active Low slave select output signal
Write_done	Output	Output signal. Pulse High for one CLK cycle after completion of write cycle.
Read_done	Output	Output signal. Pulse High for one CLK cycle after completion of read cycle. Once the read_done signal is High, read back data will be available at TX pin of UART.

API

- Application Programmer Interface (or programming or …)
- This is the term for the interfaces that are visible to developers who are writing software that will integrate with the software behind the interface.
- Part of product planning is determining how others will be allowed to integrate and then defining the interface.

Components and Interfaces

- This is one place where hardware people and software people may be further apart than in other topics we have considered.
- A component in the Unified Modeling Language "represents a modular part of a system, that encapsulates its content and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces". OMG (2008).
- An interface is a thin layer or boundary between two different substances or entities.
- http://mtc.epfl.ch/~tah/Publications/interface_theories_for_component-based_design.pdf

Components and Interfaces -2

- A component description communicates what the component can do.
- An interface description communicates how the component can be used.
- The terms interface and specification are sometimes confused.
- AADL model elements provide this separation.

Components and Interfaces -3

- Components describe behavior independent of the environment in which it is deployed.
- Interfaces constrain the environment so that specific components can work correctly.
- The interface defines the input requirements and the output guarantees.

Interface Design

- Often this defaults to user interface design, but we will focus on component interfaces.
- At the component level interfaces are defined during the architecture design process.
- Abstraction – An interface abstracts away how the behavior is provided. It declares what the component provides.
- Information hiding – Prevents external access to the implementation of those abstractions.

I/O Interfaces

- Ports are often used as the conceptual entity that passes signals, events, whatever in or out of the component
- Input ports give the requirements for the component to do its job
- Output ports define the guarantees from the components if its requirements are met

Strategic use of interfaces

- Control access
- Specify data formats
- Build a community around

Strategic use of interfaces

- Visibility
  - Proprietary – hidden
  - Licensed – for fee or various open source
  - Public –
- Standardized
  - De facto – Eclipse plug-in provided in plugin.xml
  - Industry standard – CORBA adapters

Specification languages

- OCL - Object Constraint Language
- REAL – Requirements Enforcement Analysis Language
  - https://wiki.sei.cmu.edu/aadl/images/2/25/20100204_AADLUserDay-delange-realConstraintLanguage.pdf

Interface description

- Pre/post-conditions on each function
- State model that relates the functions
- Error model that adds the error behavior

Ports

- AADL provides port and feature group
- Feature group allows a set of input/output ports to be grouped and used multiple times.
- Also the “inverse” modifier allows for the other end of a connect to reflect (i.e. male/female connectors)

Software component interfaces

- Not as well defined as hardware
- Why? Because architecture is the key. There are a few hardware architectures but a huge number of software architectures.
- Notice that component marketplaces divide their products into “platform-specific” categories. (More on platforms in Module 8 session 2)

A complete interface

- The single most frequent mistake with interface definition is to omit error semantics.
- AADL has a comprehensive error modeling syntax.
- I want to digress into that area for a few slides with the view of what should be behind the interface and then what is made visible.

Error models

- A **fault is a root cause for an error, e.g. a burned-out transistor in a memory cell in a** memory chip.
- An **error is a persistent undesired or erroneous state or condition in a component, e.g. an** incorrect word retrieved from a memory with a bad cell.
- A **failure occurs when a component is no longer able to satisfy its nominal specification,** e.g. an incorrect word retrieved from a faulty memory cannot be corrected by the provided EDAC.

Error models - 2

- An ***error model type may declare Error states, which may be used to model e.g.***
  - Hazards identified during a hazard analysis
  - Failure Modes identified during a FMEA

  (There must be a distinguished initial error state.)

- ***Error events, which may be used to model e.g.***
  - Internal faults
  - Internal repairs

  (These are made visible to permit external property associations.)

- ***Error propagations, which may be used to model e.g.***
  - Failure effects

Error models - 3

- Error model in AADL
- Parallel structure to other AADL elements
- Defined separately and then instantiated in a "system"

```
error model example
features
ErrorFree: initial error state;
Failed: error state;
Fail: error event {Occurrence => poisson lambda};
Repair: error event {Occurrence => poisson mu};
Failvisible: in out error propagation {Occurrence => fixed p};
end example;

error model implementation example.general
transitions
ErrorFree-[Fail]->Failed;
Failed-[Repair]->ErrorFree;
ErrorFree-[in Failvisible]->Failed;
Failed-[out Failvisible]->Failed;
end example.general;
```

Dependability

- The error models are an integral part of designing dependable systems.
- Please read beginning with section 4.3 and through section 5 in this tech report:
  - http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA472582&Location=U2&doc=GetTRDoc.pdf

AADL Error Annex

```
annex error {**
   Model => My_Models::Example.Notional;
   Model_Hierarchy => Error_Free when 2 ormore (S1[Error_Free],
            S2[Error_Free],
            S3[Error_Free],
            S4[Error_Free])
          and 1 orless (S1[Fail_Unknown],
            S2[Fail_Unknown],
            S3[Fail_Unknown],
            S4[Fail_Unknown]),
          Fail_Unknown when others;
   Fail_1.Vote_Transition =>
                 S2.A_Failure_Detected[Error_Free,Bad_Data]
           and S3.A_Failure_Detected[Error_Free,Bad_Data];
   Fail_2.Vote_Transition =>
               S1.A_Failure_Detected[Error_Free,Bad_Data]
          and S3.B_Failure_Detected[Error_Free,Bad_Data];
   Fail_3.Vote_Transition =>
                S1.B_Failure_Detected[Error_Free,Bad_Data]
          and S2.B_Failure_Detected[Error_Free,Bad_Data];
   Report => Error_Free;
**};
```

Error model syntax

In AADL annexes represent additions to the basic standard

annex error model *{*** -- this opens use of an annex

error model SensorErrorModel – name of model

initial error state – “error” is a modifier for state

out error propagation – an out port for error returns

*{Occurrence => **poisson 1E-3}*** – poisson is a statistical distribution defining frequency of occurrence

Full model starts on next slide=>

Error model

```
package FireAlarmErrorLib
    public
annex error model {**
    error model SensorErrorModel
    features
      error free: initial error state;
      unavailable, babbling: error state;
      smoke detected omission: out error propagation
            {Occurrence => fixed 1};
      smoke detected commission: out error propagation
            {Occurrence => poisson 1E-3};
      fail stop, fail babble: error event;
end SensorErrorModel;
```

Error model - 2

error model implementation SensorErrorModel.Standard

transitions

error free -[fail stop]-> unavailable;

error free -[fail babble]-> babbling;

unavailable -[**out smoke detected omission]->**unavailable;

babbling -[**out smoke detected commission]-> babbling;**

end SensorErrorModel.Standard;

. . .

**};

end FireAlarmErrorLib;

Extended example

- http://comjnl.oxfordjournals.org/content/early/2010/03/17/comjnl.bxq024.full.pdf+html

Extended example - 2

```
device Battery
  features
    empty: out event port;
    tryReset: in data port bool
              default false;
    voltage: out data port real
             default 6.0;
end Battery;
```

Extended example - 3

```
device implementation Battery.Imp
  subcomponents
    energy: data continuous
      default 1.0;
  modes
    charged: activation mode
      while energy' = -0.02
      and energy >= 0.2;
    depleted: mode
      while energy' = -0.03
      and energy >= 0.0;
  transitions
    charged -[then
      voltage := 2.0*energy+4.0]->
      charged;
    charged -[reset when tryReset]->
      charged;
    charged -[empty
      when energy = 0.2]-> depleted;
    depleted -[then
      voltage := 2.0*energy+4.0]->
      depleted;
    depleted -[reset when tryReset]->
      depleted;
end Battery.Imp;
```

Extended example - 4

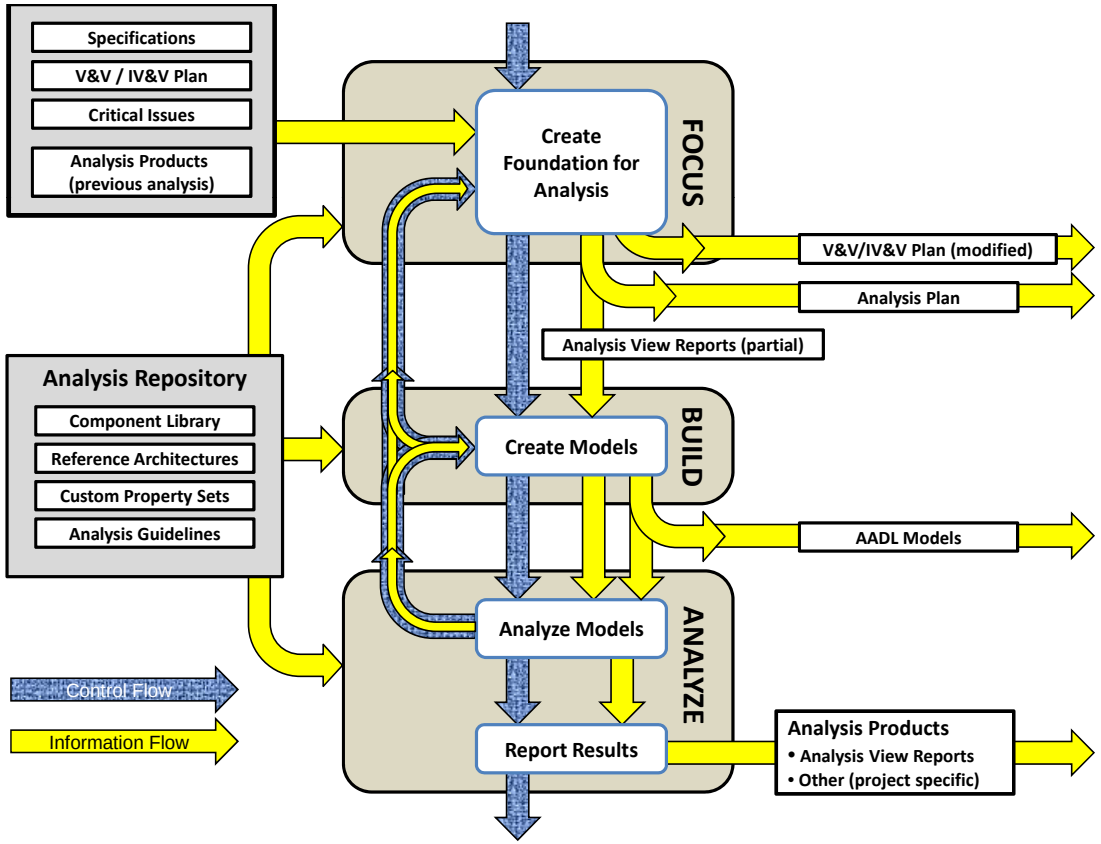
```
error model BatteryFailure
  features
    ok: activation state;
    dead: error state;
    resetting: error state;
    batteryDied: out error propagation;
end BatteryFailure;

error model implementation
  BatteryFailure.Imp
  events
    fault: error event
      occurrence poisson 0.001;
    works: error event
      occurrence poisson 0.2;
    fails: error event
      occurrence poisson 0.8;
  transitions
    ok -[fault]-> dead;
    dead -[batteryDied]-> dead;
    dead -[reset]-> resetting;
    resetting -[works]-> ok;
    resetting -[fails]-> dead;
end BatteryFailure.Imp;
```

AADL and Software Assurance

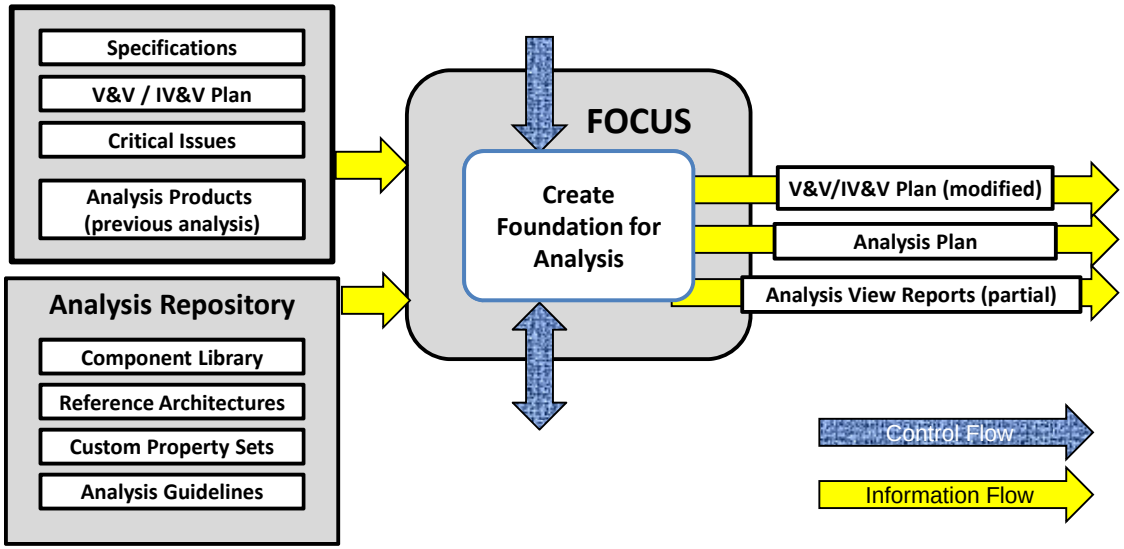
- AADL Practice Framework for Model-Based Analysis
- http://sarpresults.ivv.nasa.gov/ViewResearch/21/156.jsp

NASA Analysis Process for AADL Models



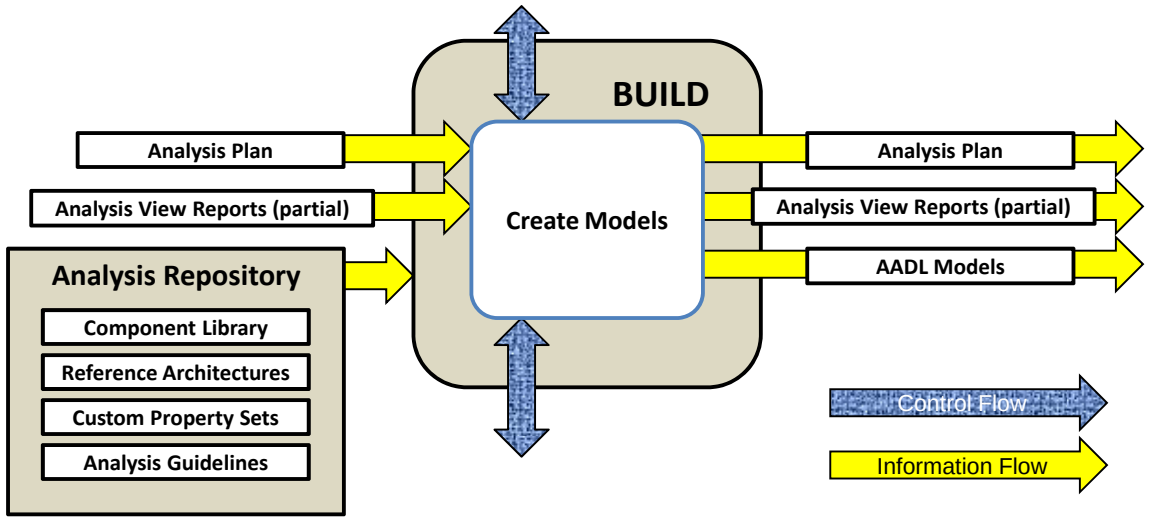
Focus

- Critical issues – found by independent analyses such as a Quality Attribute Workshop (QAW by the SEI), Fault Tree Analysis (a previous session), and others such as risk analysis.
- Analysis Guidelines – developed by NASA, determines which models to build
- Outputs Analysis View reports
- An analysis viewpoint is characterized by a set of related issues and establishes conventions for developing and documenting analysis views to address those issues.



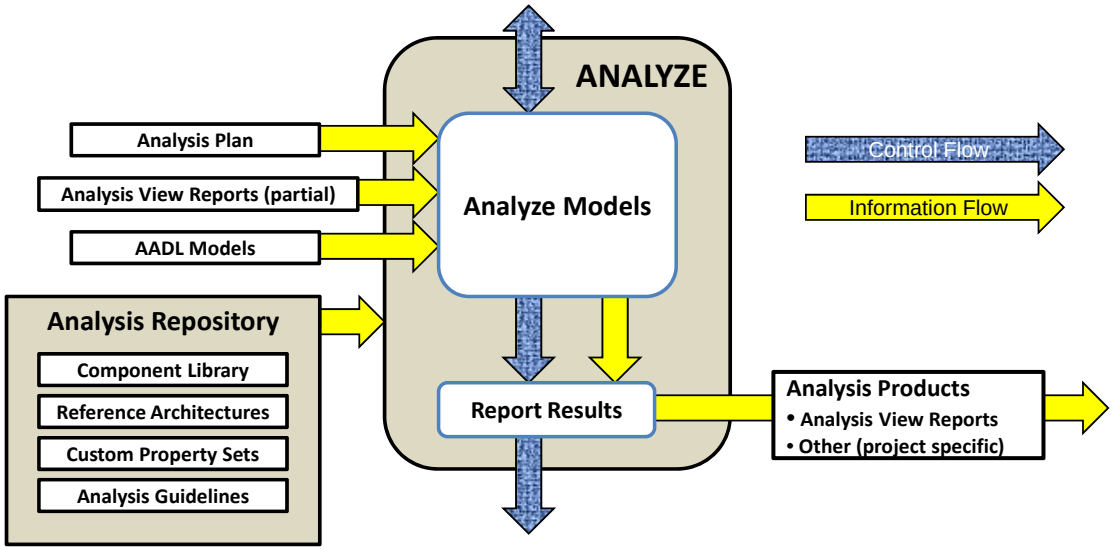
Build

- Analysis View Reports (AVRs) are inputs
- Main activity is building new AADL models or revising existing ones
- Add the elements needed to support the analyses defined in the AVRs



Analyze

- Run the analyses, defined in Focus, on models created in Build.
- The OSATE toolkit is the host for these analyses.



Example Analyses

Viewpoint	Analyses (AADL)

Performance
Issues:
• Data Flow
• Data Consistency
• Time Partitioning
	• Transport latency
	• Latency jitter
	• Schedulability
	• Execution Time/Deadline
	• Mode Specific
	• Data rate
	  o Instantaneous (peak, averages)
	  o over time (integrated)

Summary

- An interface is a specification
- The pre/post-conditions and function signature define what is required and what is provided
- The state models that are part of the interface definition add constraints to the order in which functions can be used.

Here is what you are going to do

- Review, renovate, and comment your interfaces
- Call out each interface and make certain that the interface comments describe how to use this interface and why the interface is designed as it is
- Submit by 11:59pm on April 1 via email