Requirements introduction

Requirements express conditions or capabilities that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed properties.

Requirements and Architecture

- The core EAST-ADL model captures basic structure of system
- Requirement elements annotate model to identify mandatory aspects
- Requirement elements relate requirements engineering information to architectural model

Requirement Representation

To provide language means to

•specify required properties of the system (at varying degrees of abstraction)

•trace requirements between system refinement and system decomposition levels

•require satisfaction of requirements for system components

•refine the specification of requirements by behavioral models

•plan, organize and log activities for verification and validation of requirements

Requirements vs Model

Requirement entity identify textually defined, compulsary properties

Formalisation of requirements with Constraints, behavoural entities, etc. are possible with RefineReq relation

Depending on the roles in a model exchange:

- An entire model may be seen as a requirement
- Selected parts of the model are required
- An entire model may be seen as a specification

Requirements – Idea & Concepts

Req. support in EAST-ADL2 is based on **“Generic Requirements“**

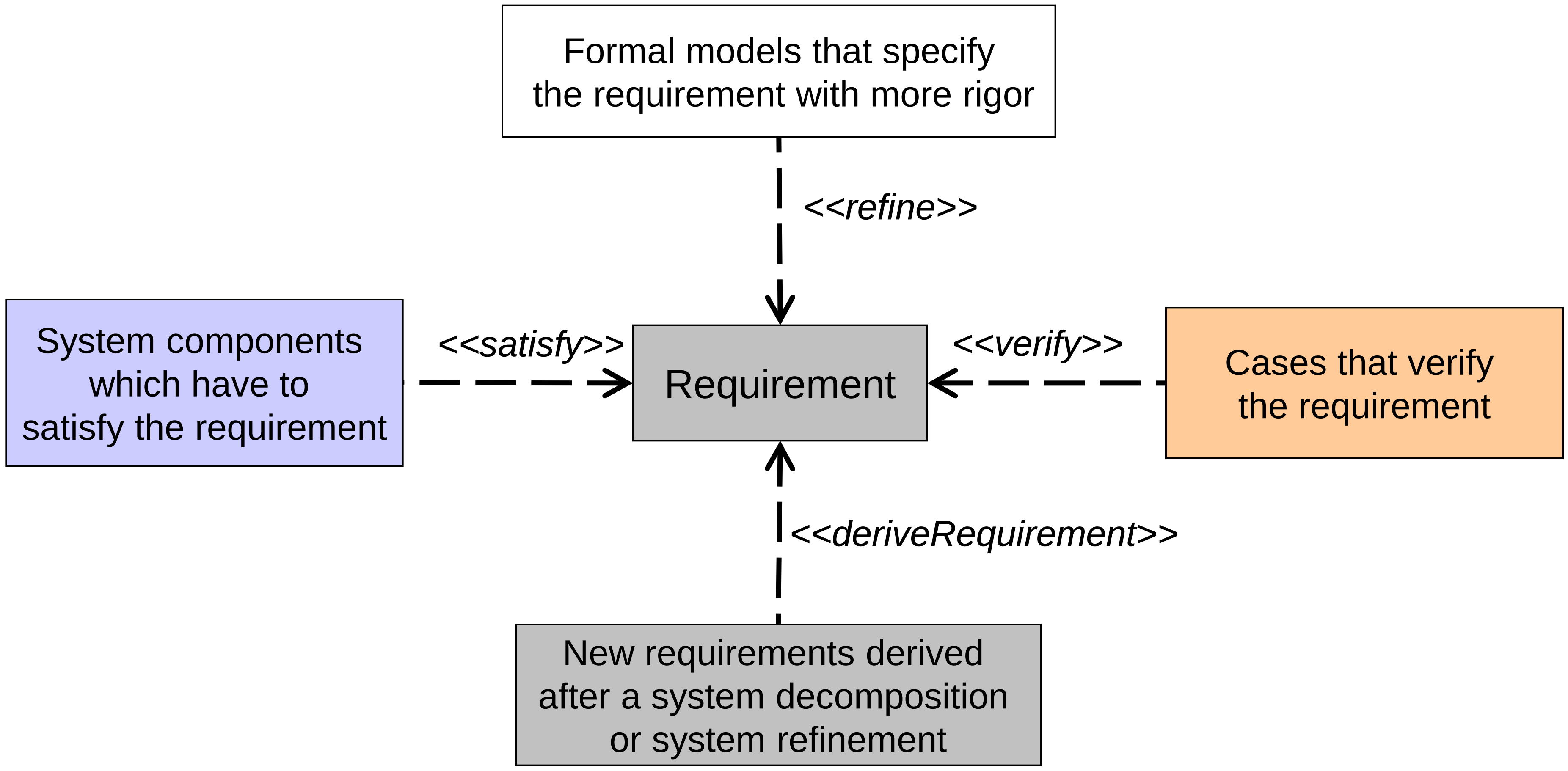
= simple objects that contain all information (except ID and text. description) in customized attributes (same philosophy as DOORS and RIF) **+** support for links and groups

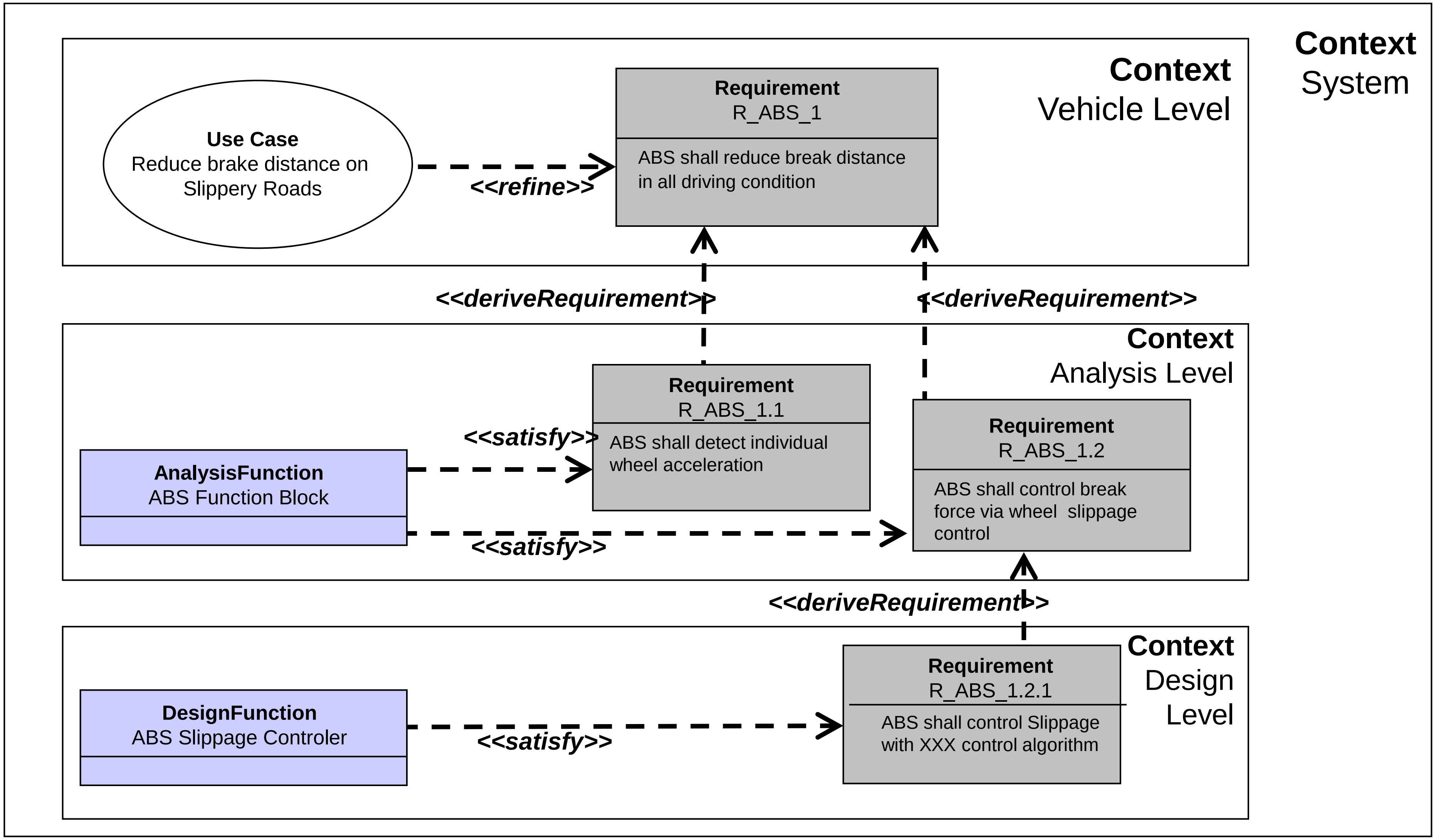
New concept for the customized attributes is called **“User Attributes“**

User Attributes can then be made available for other EAST-ADL2 elements as well (e.g. ADLFunctionType) ➔ they become concept for overall project-/company-specific customization of the EAST-ADL2

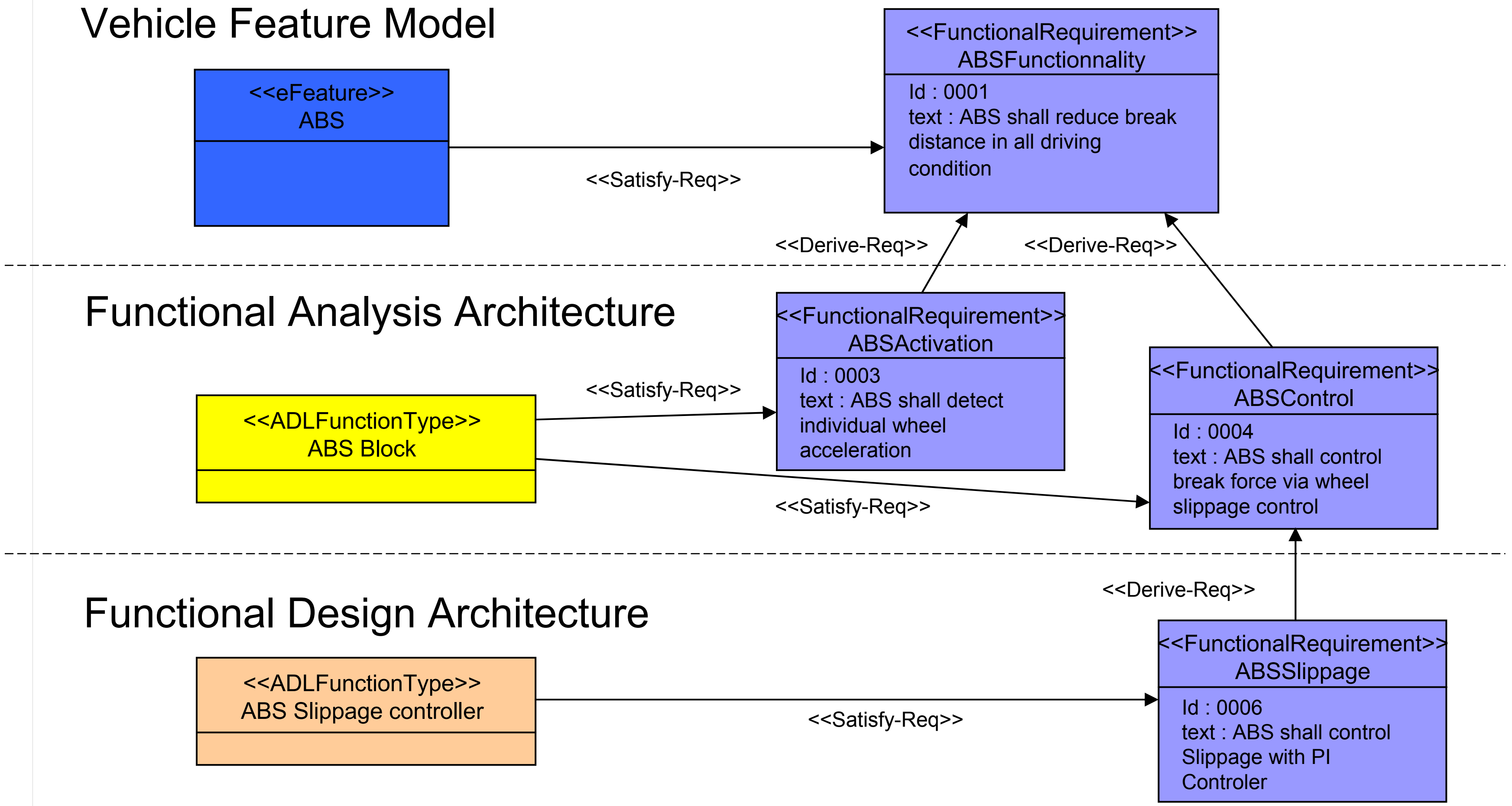
In addition to generic requirements: **Specialized Requirements** add certain attributes and associations for a special purpose and with a special semantic (e.g. timing requirements)
➔ allow for tight coupling with system definition (FAA, FDA, HA, etc.)

Requirements – basic relations


Formal models that specify the requirement with more rigor
<<refine>>
System components which have to satisfy the requirement
<<satisfy>>
Requirement
<<verify>>
Cases that verify the requirement
<<deriveRequirement>>
New requirements derived after a system decomposition or system refinement


Context
System
Context
Vehicle Level
Use Case
Reduce brake distance on Slippery Roads
<<refine>>
Requirement
R_ABS_1
ABS shall reduce break distance in all driving condition
<<deriveRequirement>>
<<deriveRequirement>>
Context
Analysis Level
AnalysisFunction
ABS Function Block
<<satisfy>>
Requirement
R_ABS_1.1
ABS shall detect individual wheel acceleration
<<satisfy>>
Requirement
R_ABS_1.2
ABS shall control break force via wheel slippage control
<<deriveRequirement>>
Context
Design
Level
DesignFunction
ABS Slippage Controler
<<satisfy>>
Requirement
R_ABS_1.2.1
ABS shall control Slippage with XXX control algorithm

Requirements – Tracing and Linking to system components

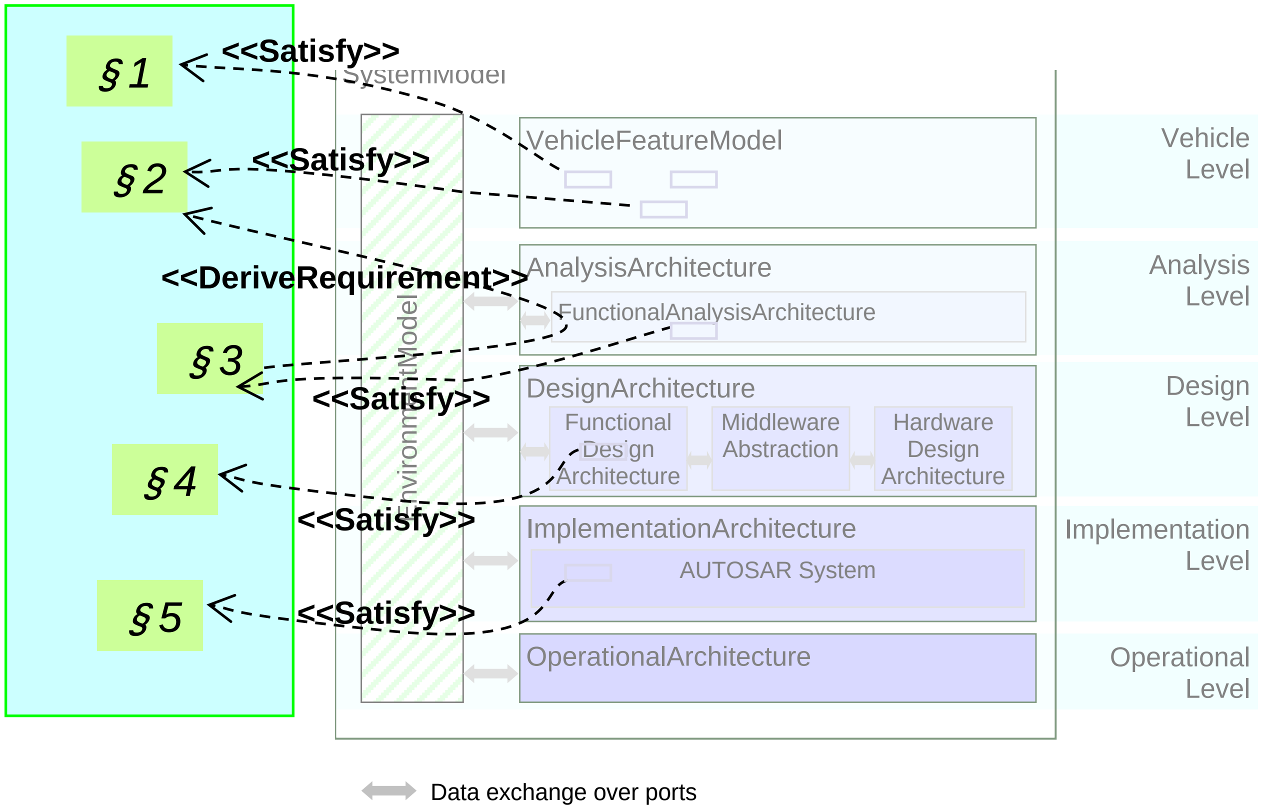

Vehicle Feature Model
<<eFeature>>
ABS
<<Satisfy-Req>>
<<FunctionalRequirement>>
ABSFunctionnality
Id : 0001
text : ABS shall reduce break distance in all driving condition
<<Derive-Req>>
<<Derive-Req>>
Functional Analysis Architecture
<<FunctionalRequirement>>
ABSActivation
Id : 0003
text : ABS shall detect individual wheel acceleration
<<Satisfy-Req>>
<<ADLFunctionType>>
ABS Block
<<FunctionalRequirement>>
ABSControl
Id : 0004
text : ABS shall control break force via wheel slippage control
<<Satisfy-Req>>
<<Derive-Req>>
Functional Design Architecture
<<ADLFunctionType>>
ABS Slippage controller
<<Satisfy-Req>>
<<FunctionalRequirement>>
ABSSlippage
Id : 0006
text : ABS shall control Slippage with PI Controler

Requirements

Requirements are normally defined in a central repository.

Satisfy relates requirements to features, functions and components.

Derive relates a derived requirement to its original.

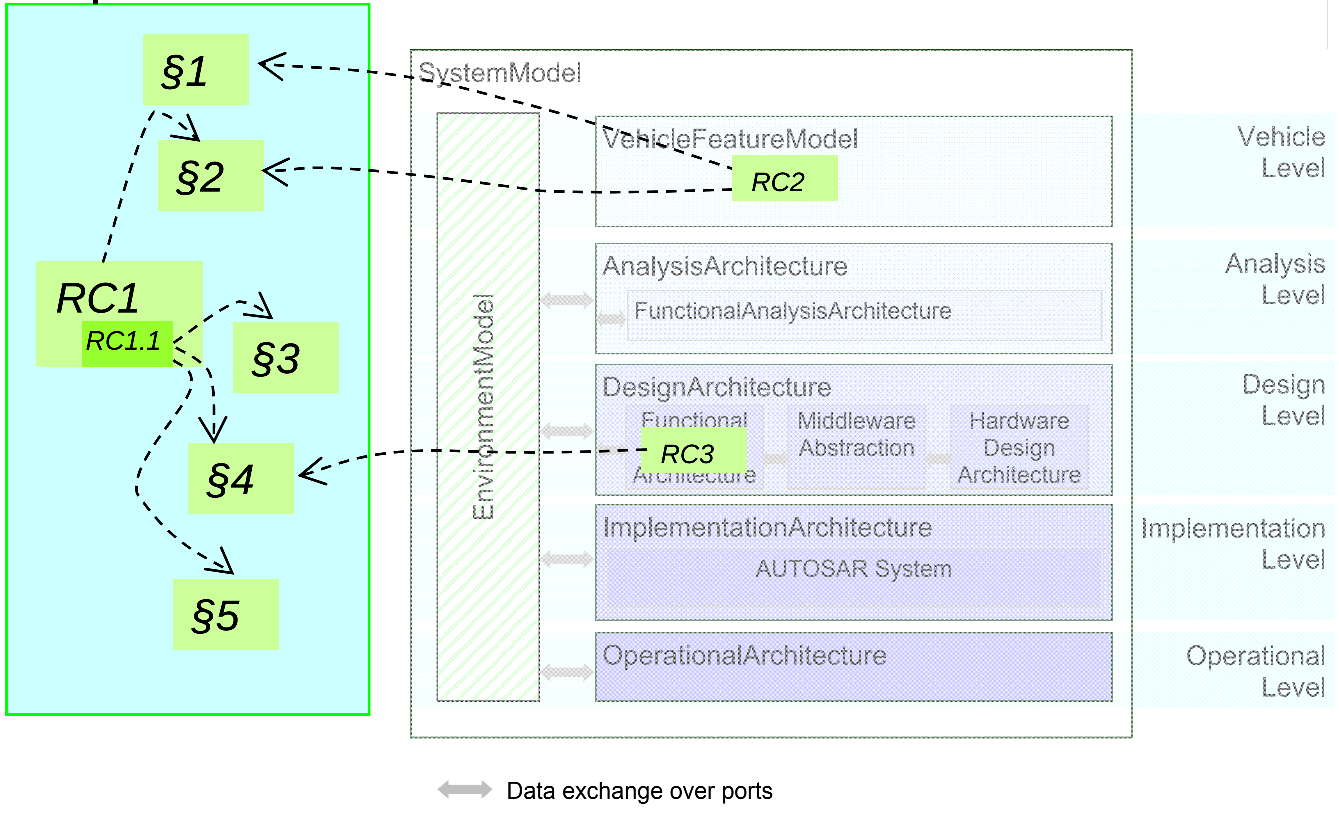


Requirements

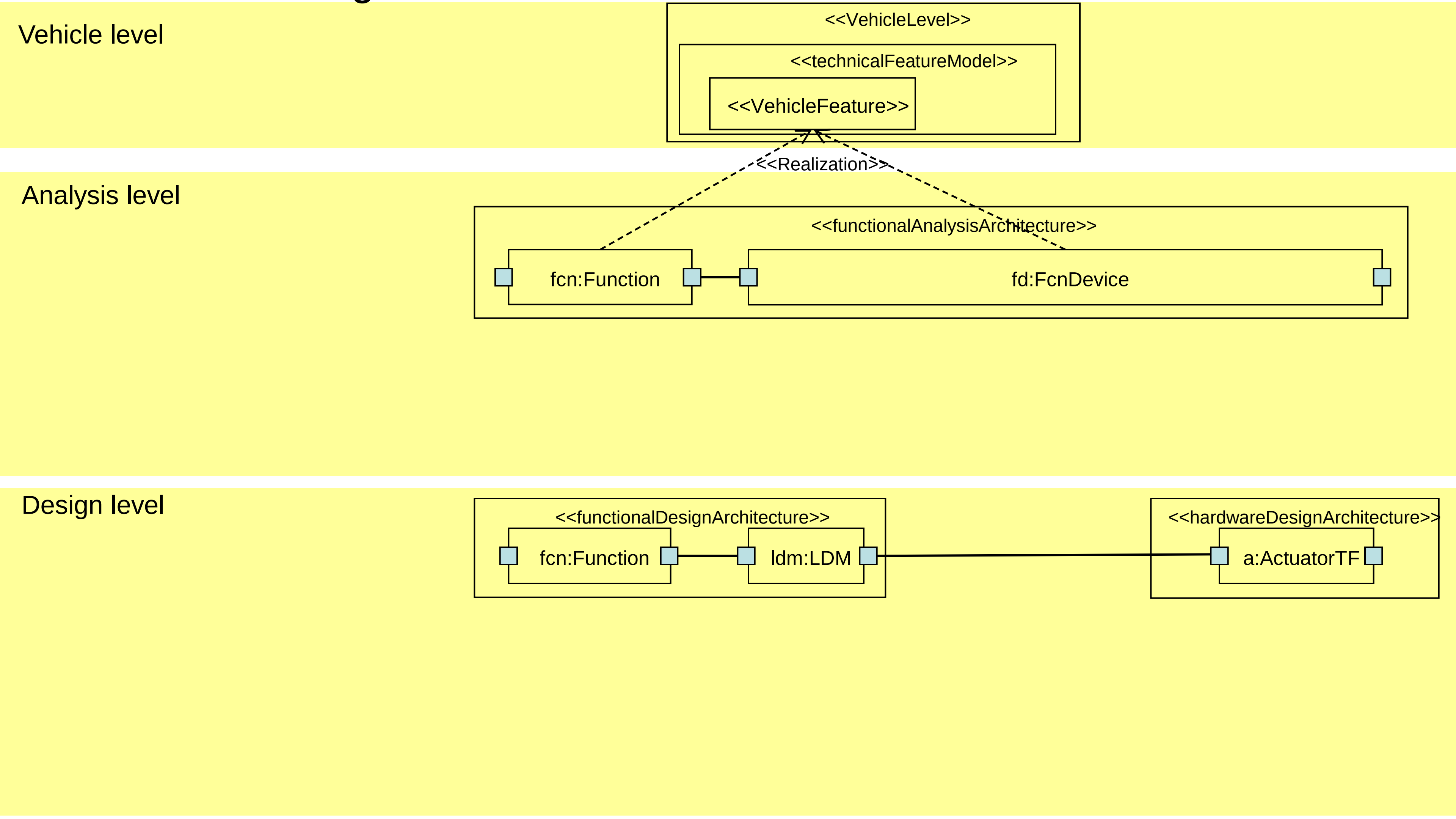
RequirementContainers may be used to organize requirements

Containers may be local or global

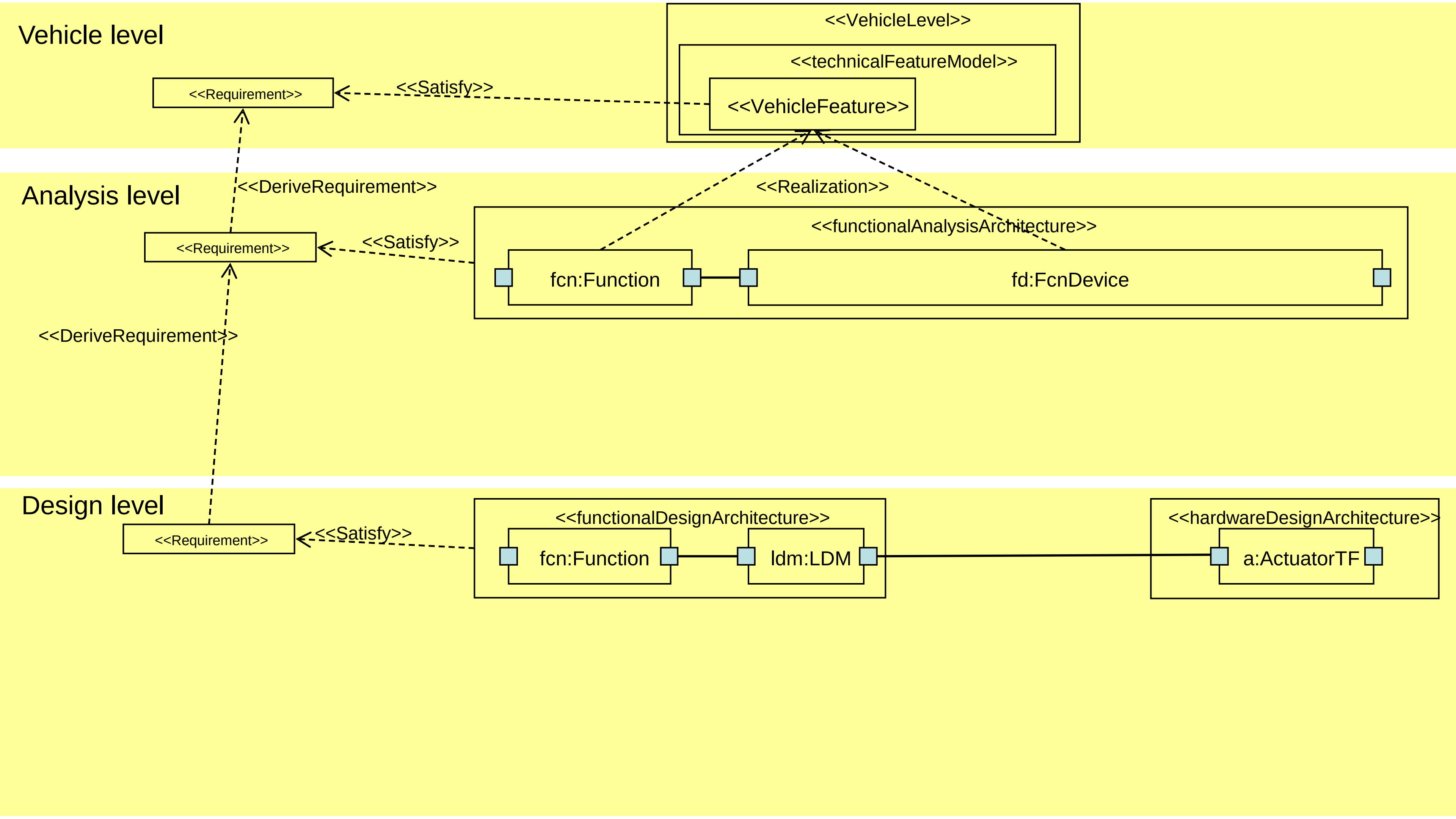
The same requirement may be linked to several containers.



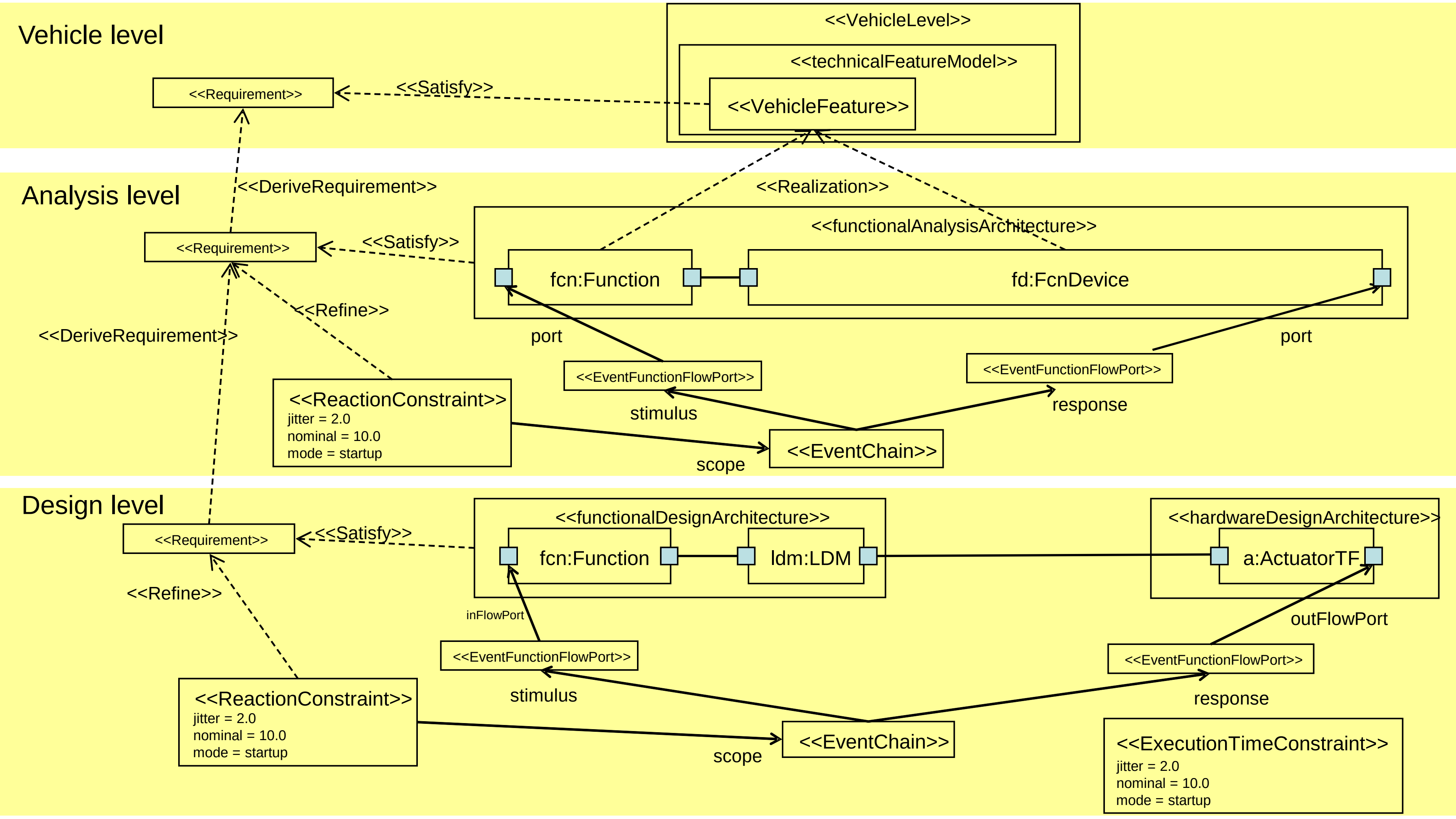
Elements Realizing More Abstract Elements



Requirements associated to elements and between abstraction levels



Refining requirements with formalized constraints



Verification & validation support – basic concepts

- VVCase = a certain, overall V&V effort of varying scope and intention
  - core concept of V&V support in EAST-ADL
  - Safety analysis; specification, design or implementation review; analysis or design level simulation, SIL-testing, HIL-testing, vehicle testing
- VVProcedure = individual task in the context of an overall V&V effort (i.e. a VVCase), which has to be performed in order to achieve that effort's overall objective.
- VVTarget = concrete testing environment in/on which a particular V&V activity (i.e. VVProcedure) can be performed
  - can be physical hardware or pure software (e.g. design level simulations)
- VVLog = captures outcome of an actual execution of a V&V activity

Controller vs Plant : MBD Approach

- One builds control software to interact with a mechatronic system:
  - Control software/Mechatronic system referred to as the "controller" and the "plant“ in MBD workflow.
- One develops a “controller” to influence a “plant” based on “requirements”

Model In Loop (MIL)

- Once you've developed your controller model, you would run a simulation to "verify" that your controller influences the plant in such a way that "requirements" are met;

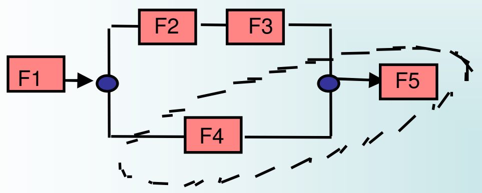
- To verify the design, develops "test cases" and "expected results", runs the simulation using the test cases and verifies the generated results against the expected results;

- MIL is referred to a behavioral simulation, as the objective is to get the desired behavior;

- “At this point you have a good controller model design that has been verified against the requirements and ideally an independently verified plant model”

Validation Stages

MiL: Model in the loop



Models run in a virtual environment

- *Functional models* – at the system level
- *Implementation models* – at an individual module and a whole system levels

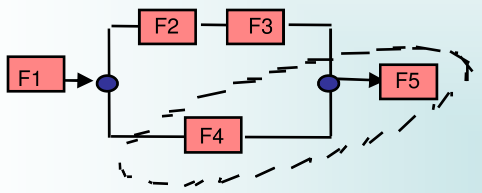
Software In Loop (SIL)

- The control model is slightly more "real" in the sense that you are no longer executing the model but rather you have probably coded the model into C or C++ and then inserted this coded model back into your overall plan simulation;

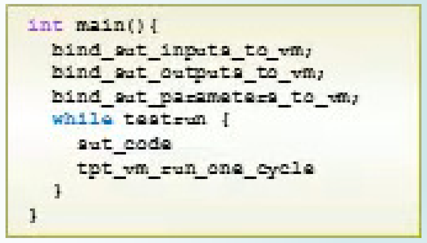
- Ready to Implement the controller:
  - This could take the form of code to run on a processor target or HDL code for an FPGA target;
  - The software/HDL code execution could replace the controller model in the simulation and this type of simulation would be referred to as SIL;

Validation Stages

MiL: Model in the loop



SiL: Software in the loop

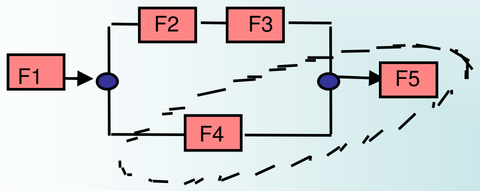


```
int main(){
  bind_sut_inputs_to_vm;
  bind_sut_outputs_to_vm;
  bind_sut_parameters_to_vm;
  while testrun {
    sut_code
    tpt_vm_run_one_cycle
  }
}
```

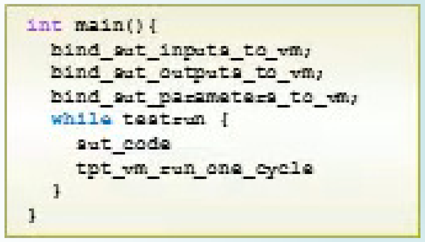
Software runs in
a virtual environment,
without any hardware

Validation Stages

MiL: Model in the loop



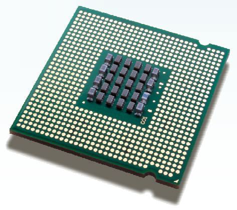
SiL: Software in the loop



```
int main(){
  bind_sut_inputs_to_vm;
  bind_sut_outputs_to_vm;
  bind_sut_parameters_to_vm;
  while testrun {
    sut_code
    tpt_vm_run_one_cycle
  }
}
```

Software runs on a target processor or an emulator, but with a proprietary hardware (ECU)

PiL: Processor in the loop



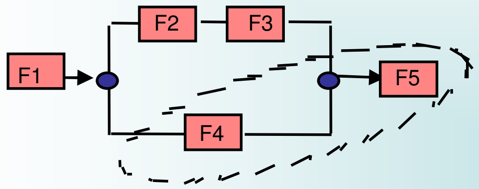
Hardware In Loop (HIL)

- Once the target controller hardware is available, HIL testing can begin:
  - HIL testing can be done against the plant model simulation, but now the plant model must interact with actual I/O that the controller hardware is expecting;
  - This could service as a factory acceptance test prior to installation and test against the real plant.

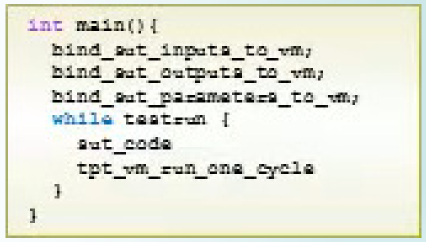
- In MBD, this workflow, e.g., MIL, SIL, HIL, is the basis of many functional safety certifiable workflows. → At anytime you can go back to previous states, make changes and rerun the simulation.

Validation Stages

MiL: Model in the loop

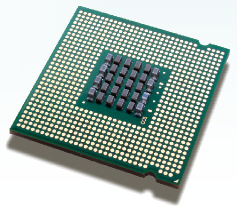


SiL: Software in the loop



```
int main(){
  bind_sut_inputs_to_vm;
  bind_sut_outputs_to_vm;
  bind_sut_parameters_to_vm;
  while testrun {
    sut_code
    tpt_vm_run_one_cycle
  }
}
```

PiL: Processor in the loop

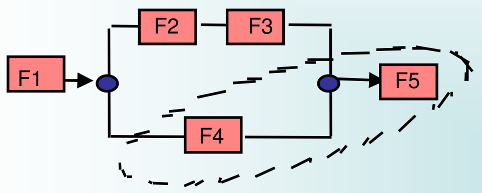


HiL: Hardware in the loop

Final ECU with simulated environment

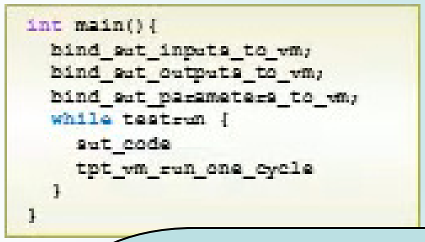
Validation Stages

MiL: Model in the loop



HiL: Hardware in the loop

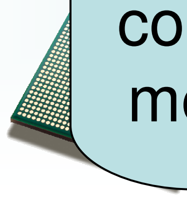
SiL: Software in the loop



```
int main(){
  bind_sut_inputs_to_vm;
  bind_sut_outputs_to_vm;
  bind_sut_parameters_to_vm;
  while testrun {
    sut_code
    tpt_vm_run_one_cycle
  }
}
```

Test Rig

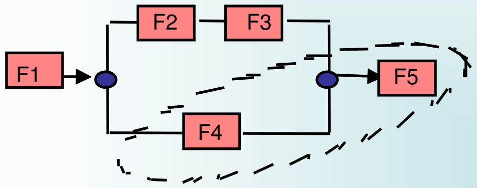
PiL: Pro



The environment consist of physical components (electrical, mechanical, hydraulic)

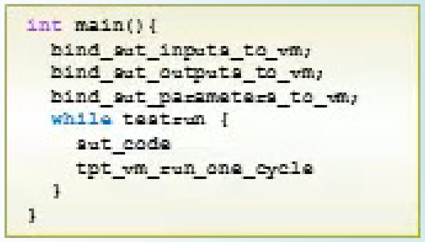
Validation Stages

MiL: Model in the loop



HiL: Hardware in the loop

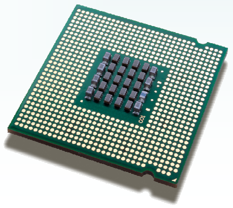
SiL: Software in the loop



```
int main(){
  bind_sut_inputs_to_vm;
  bind_sut_outputs_to_vm;
  bind_sut_parameters_to_vm;
  while testrun {
    sut_code
    tpt_vm_run_one_cycle
  }
}
```

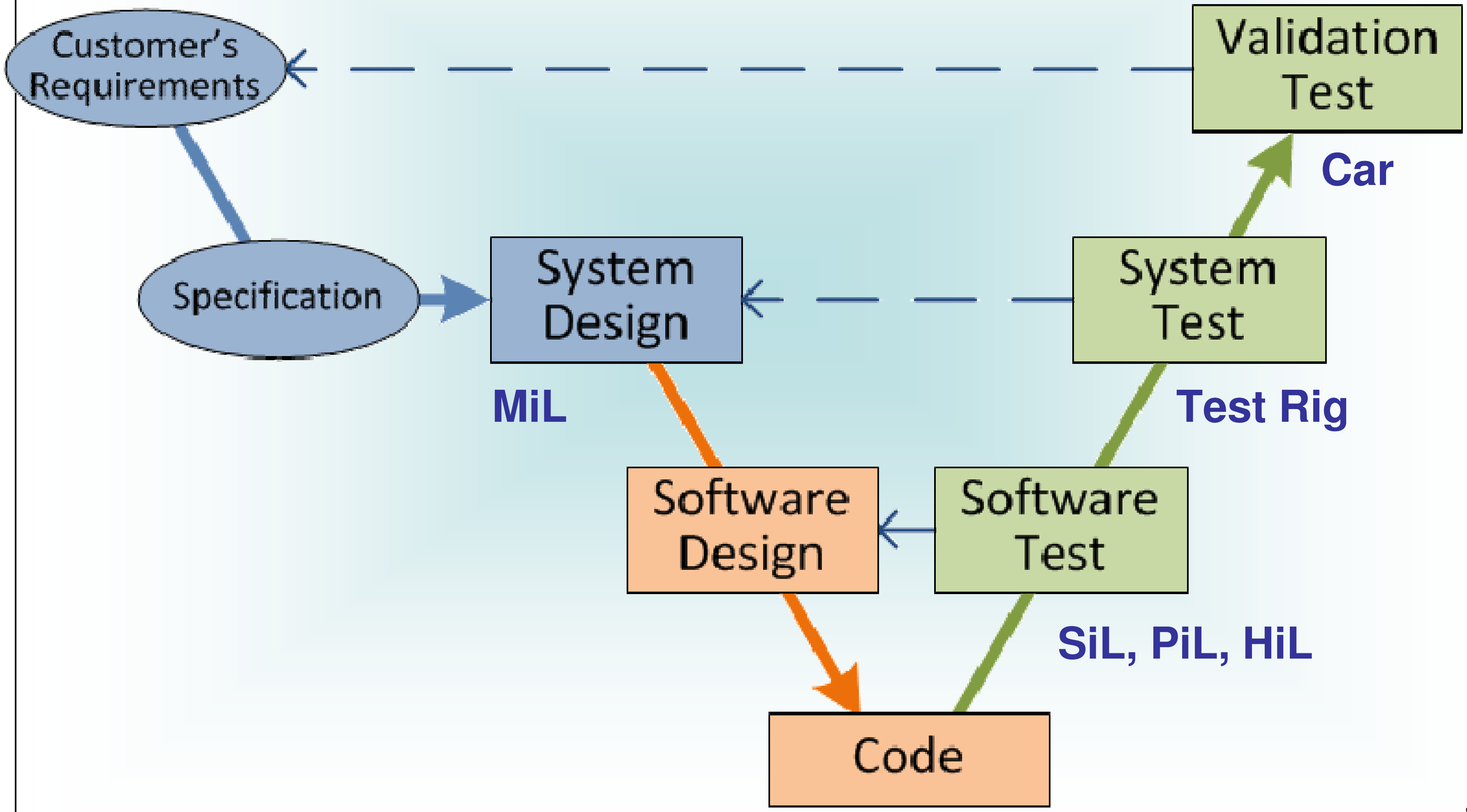
Test Rig

PiL: Processor in the loop

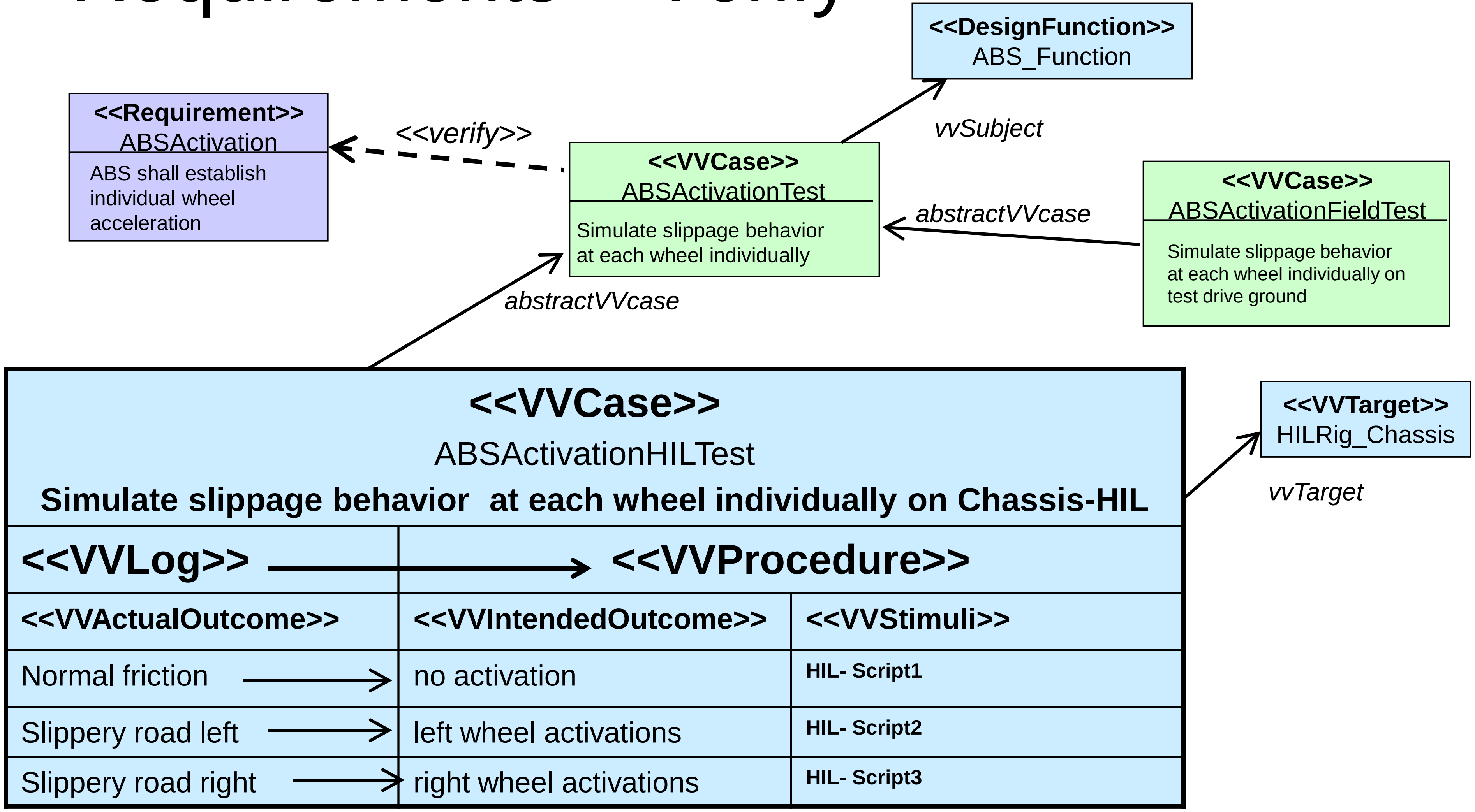


Car

The V Development Model



Requirements – Verify



<<VVActualOutcome>>	<<VVIntendedOutcome>>	<<VVStimuli>>
Normal friction →	no activation	HIL- Script1
Slippery road left →	left wheel activations	HIL- Script2
Slippery road right →	right wheel activations	HIL- Script3

Summary

- Requirement elements are textual (but may use a formal notation)
- Requirements are linked to architectural elements
- Requirements are linked to derived requirements
- Formalizations can be linked to requirements (Constraints and behavioural models)
- Requirements are linked to V&V constructs
- V&V constructs represent test cases and test results