

Using SPARK to ensure System to Software Integrity

Tonu Naks, M. Anthony Aiello, S. Tucker Taft

AdaCore

Agenda

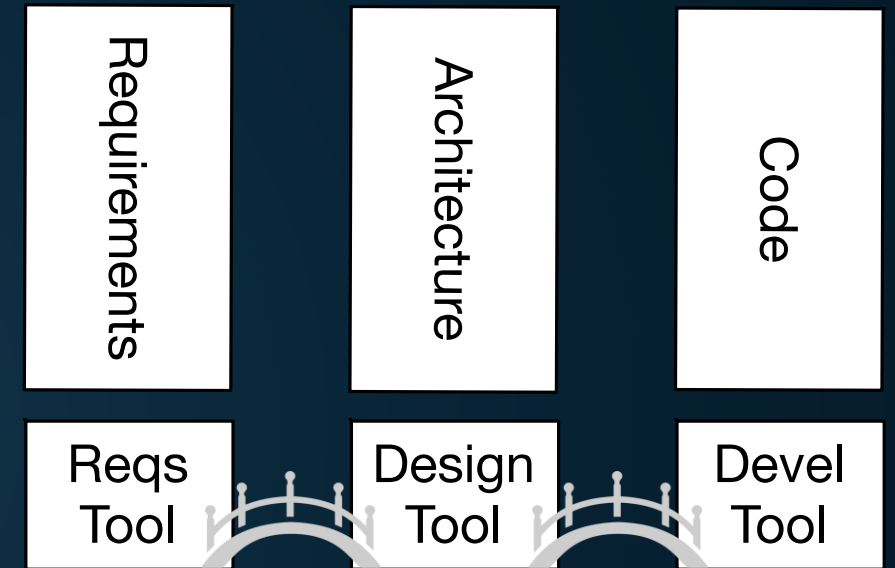
- AdaCore System-to-Software Integrity (SSI) initiative
- Workflow in a nutshell
- Workflow demonstrated by a case study
- Challenges/open questions/next steps

SSI

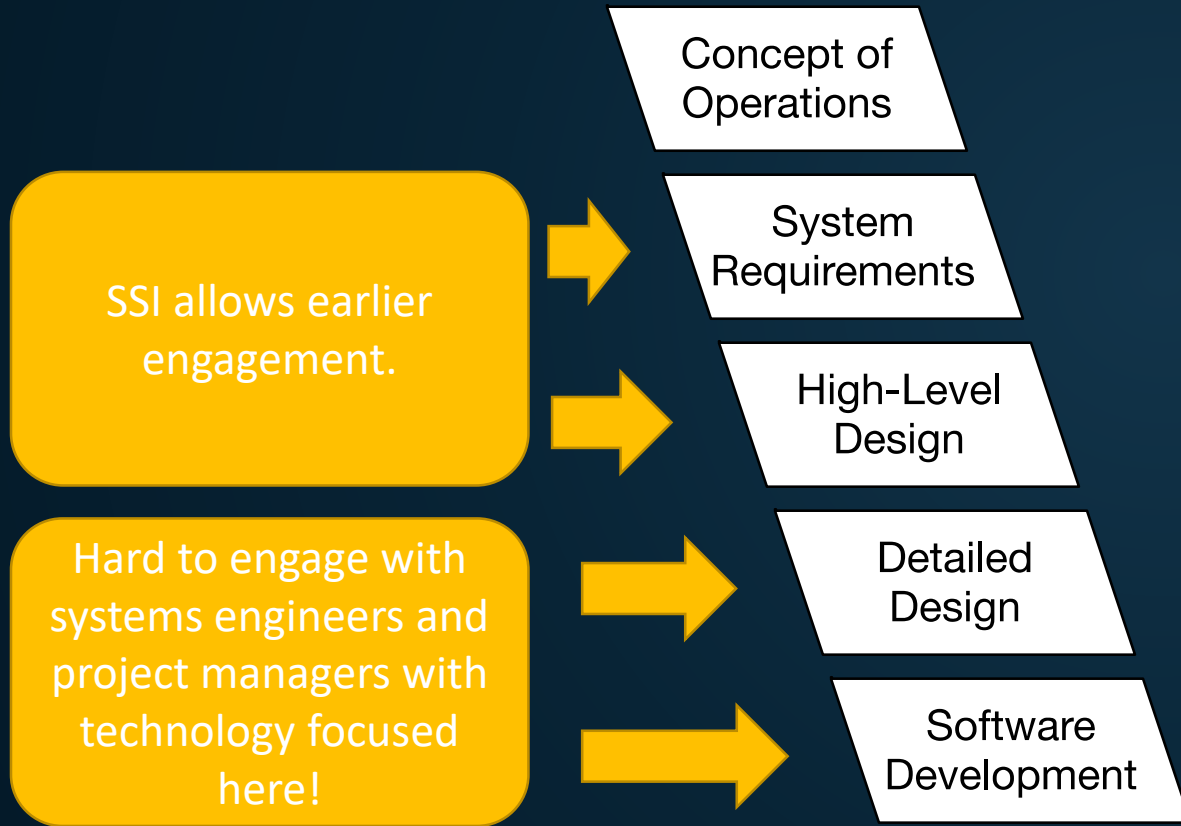
System-to-software integrity is a desired trait of high-assurance systems engineering.

- Ensure development process yields adequate assurance
- Link artifacts at different levels with formal properties & tool support
- Help engineers in moving from level to level with smart translations
- Reduce information loss in communication of various teams

System-level properties maintained through each development step until realized in software.



SSI Tooling Bridges Silos

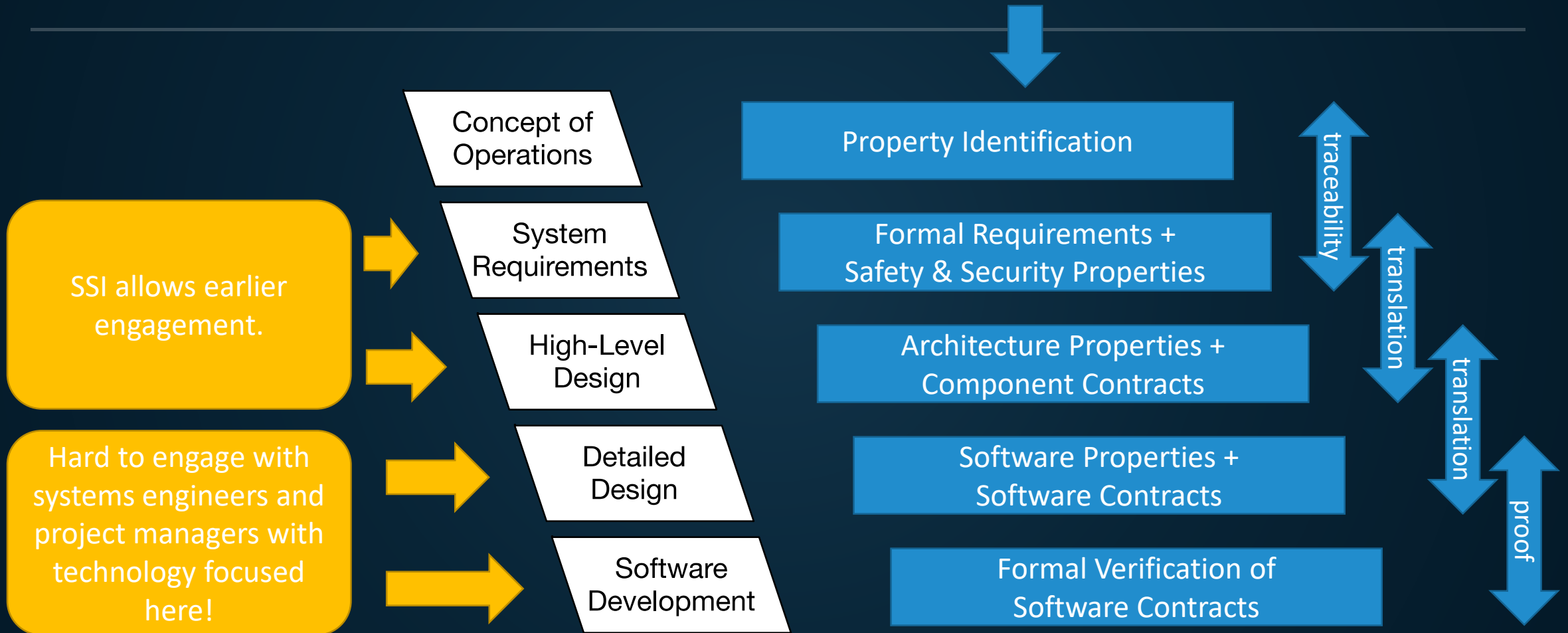


SSI

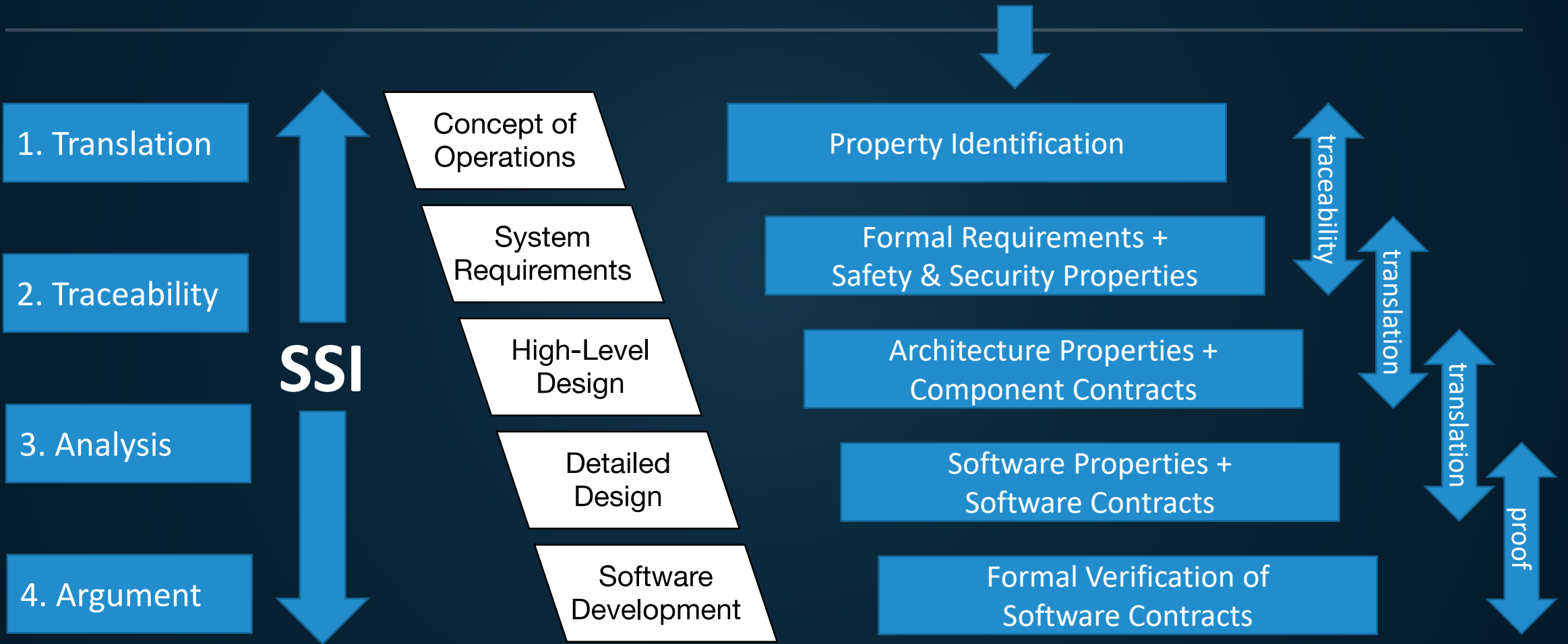
System-level properties maintained in software

- Hard for software engineers to identify application-specific properties
- Hard for systems engineers to think about software-level properties
- SSI allows **early engagement** and **property continuity**

SSI



SSI



SSI

System-level properties maintained in software

1. Translation

- Translate Properties from one “level” to the next
- Example: properties for requirements -> properties as contracts in a design.
- Property decomposition may be required

2. Traceability

- Bidirectional traceability of properties across “levels”
- Trace properties to models & code
- Monitor for broken links

3. Analysis

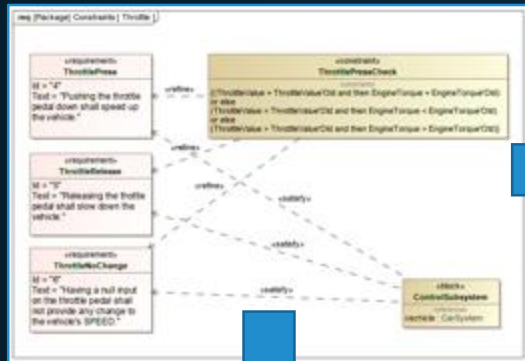
- Vertical: prove that properties are consistent across levels
- Horizontal: prove that decompositions satisfy higher-level properties

4. Argument

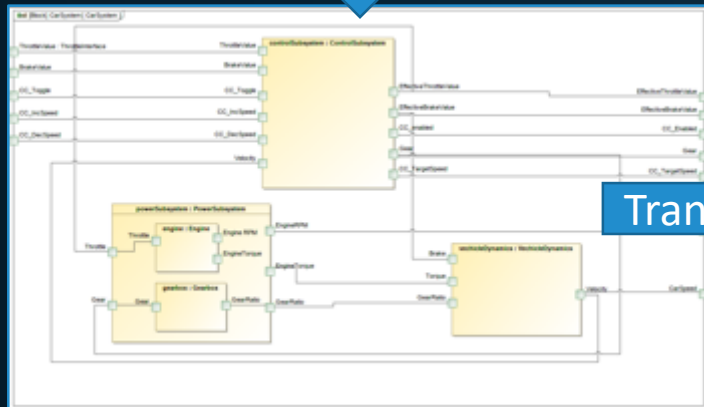
- SSI evidence may need logical induction to justify fully
- Present & justify evidence where deduction is not fully possible
- Provide support for certification

SSI tooling example

SysML Requirements Diagram

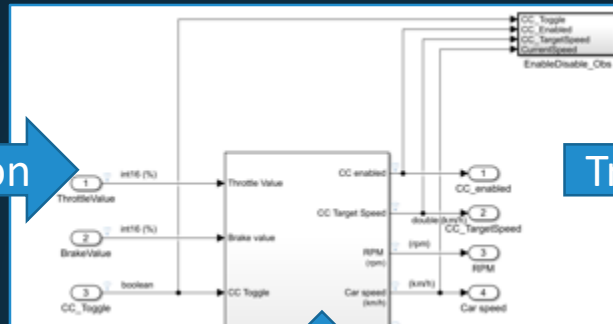


Manual Refinement

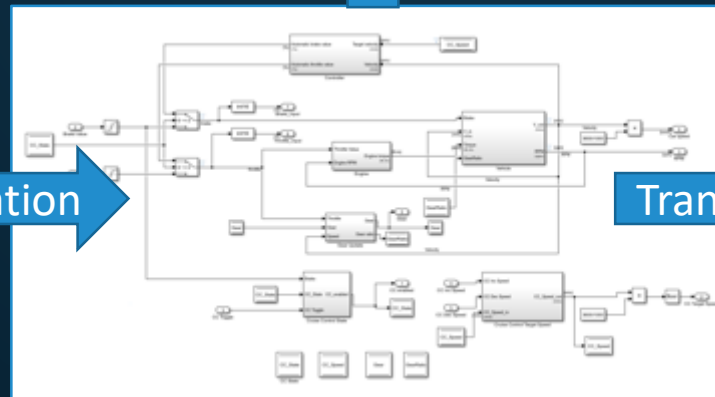


SysML Internal Block Diagram

Simulink Synchronous Observer

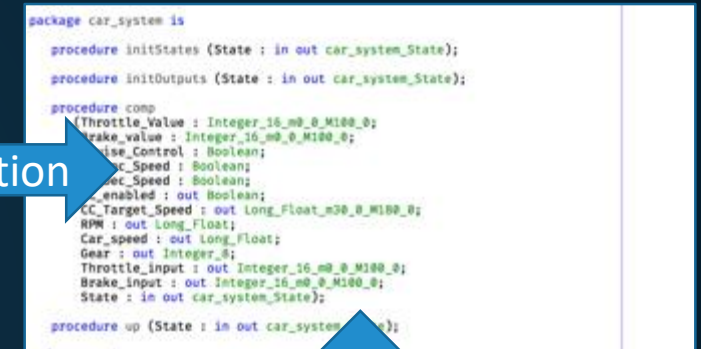


QGen Verifier

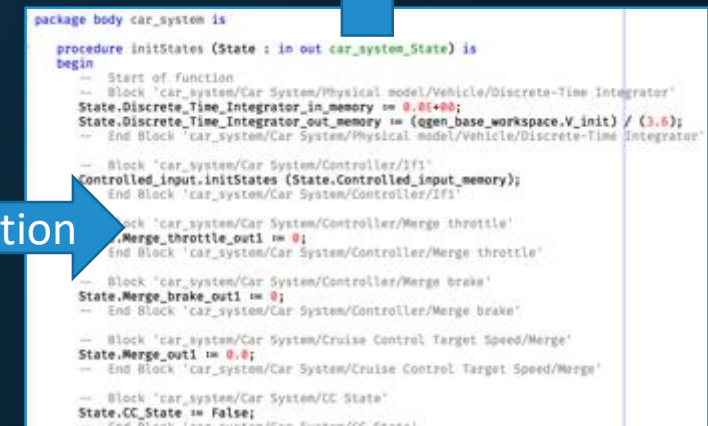


Simulink Subsystem

SPARK Contracts



GNATProve



SPARK Code

A Case Study

Autopilot Simulator

SysML → Simulink → SPARK

A sample workflow

- A simple demo application mimicking behavior of a car cruise controller
 - STM-32 board running the cruise control and car model
 - LCD screen on the board emulating car cockpit displays
 - A dashboard application allowing to control the board from PC

Contents

- Use Cases
- Context
- System structure
- Value types
- HLR
- LLR -- Control
- CarSystem

QGen Automotive demonstration project

The aim of this project is to develop a demo application mimicking behaviour of car cruise controller. The application consists of:

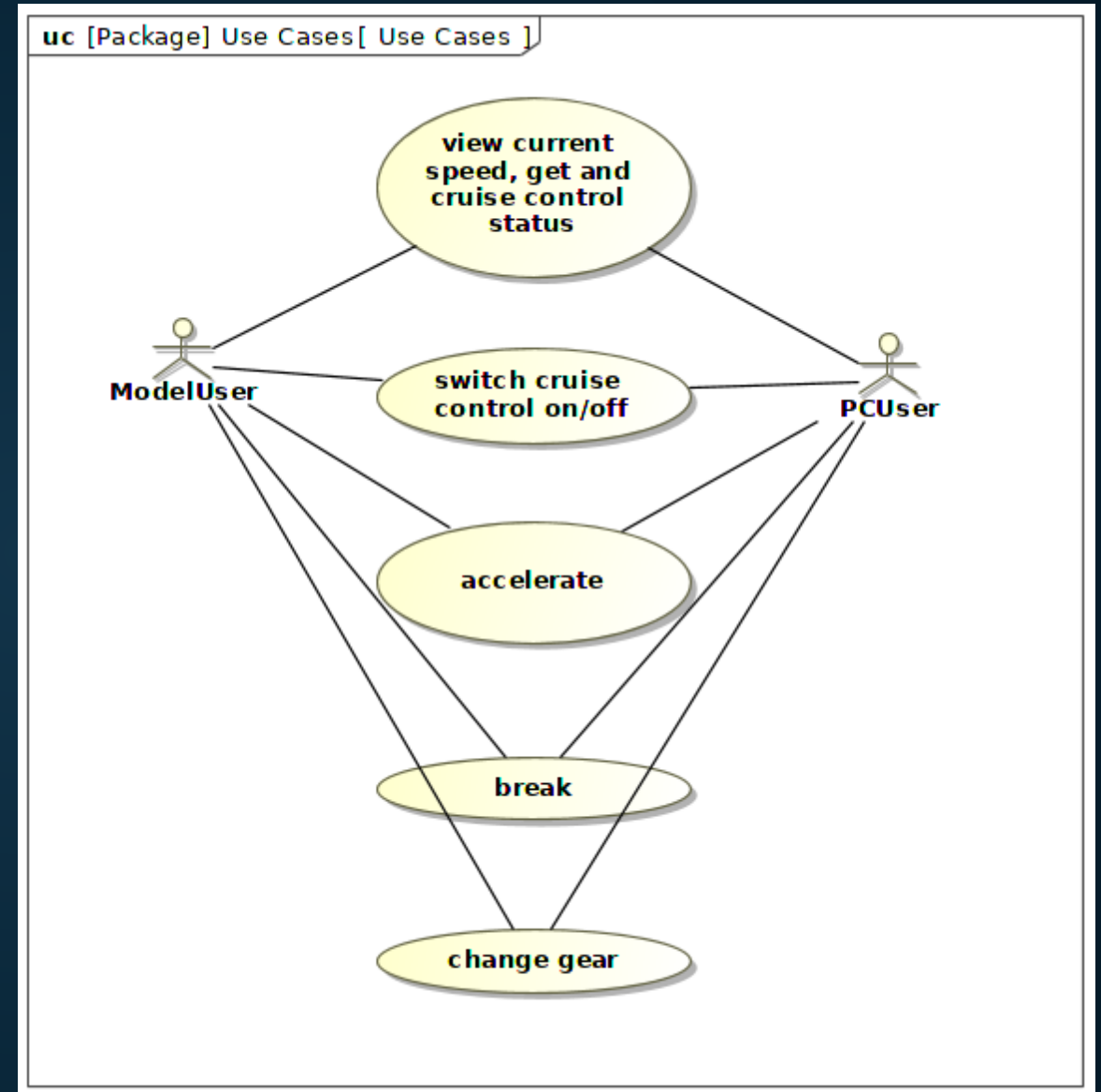
- * a STM-32 board running the cruise controller and a car model
- * an LCD screen on the board displaying the system status
- * a dashboard project allowing to control the application from a PC

The model takes throttle and brake input and outputs the car's speed, RPM, gear. It also provides the user with a cruise control that may be enabled, incremented or decremented, and outputs the corresponding automatic throttle and brake values, cruise control state and target speed.

Throttle and brake input are provided using actual pedals devices. The target also communicates with a separate dashboard application displaying the outputs.

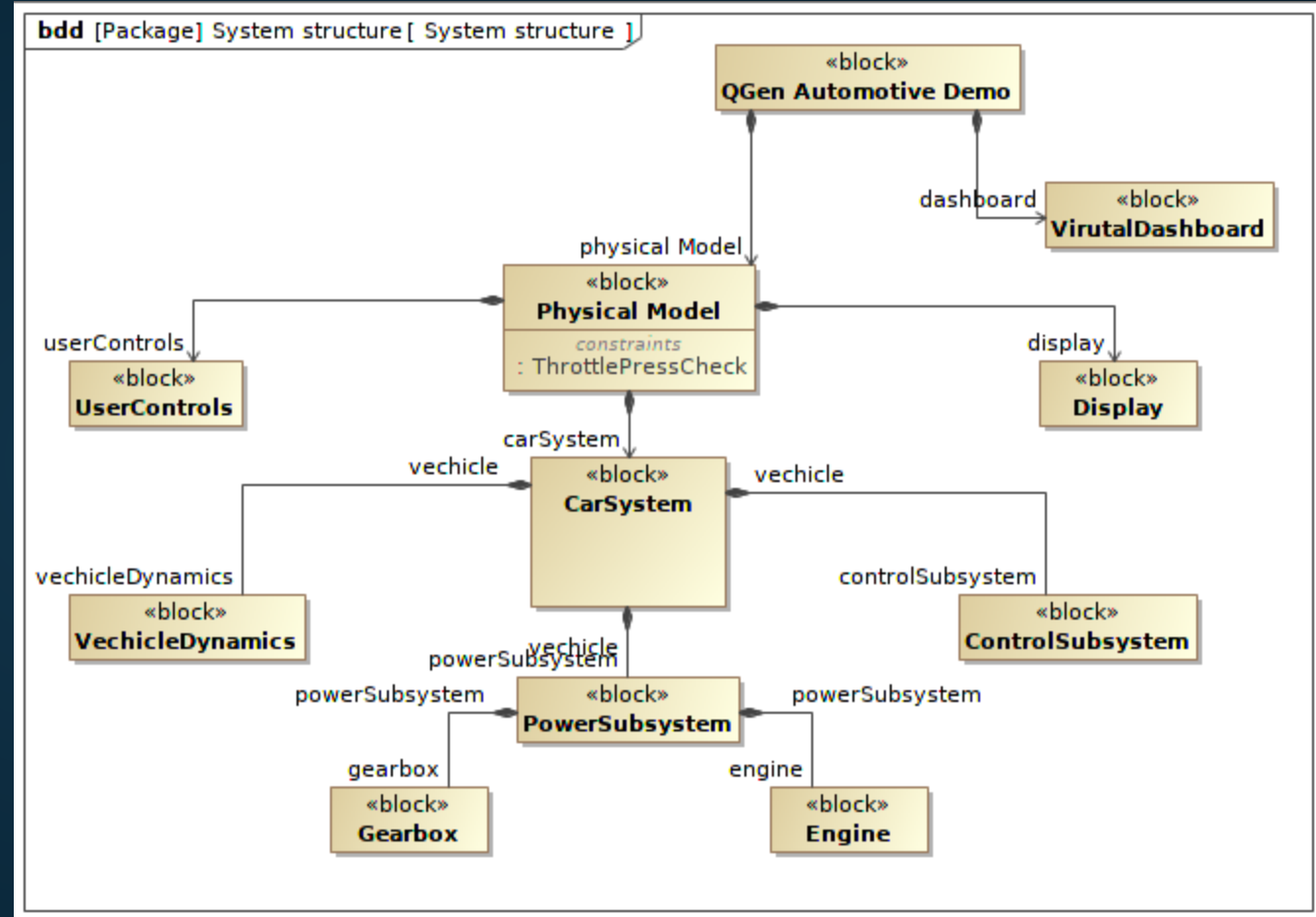
Stakeholders & Use-Cases

- Two types of users:
 1. **ModelUser**: manipulates the system through physical controls on the demo box
 2. **PCUser**: controls the system through PC application
- Both have access to the same use-cases



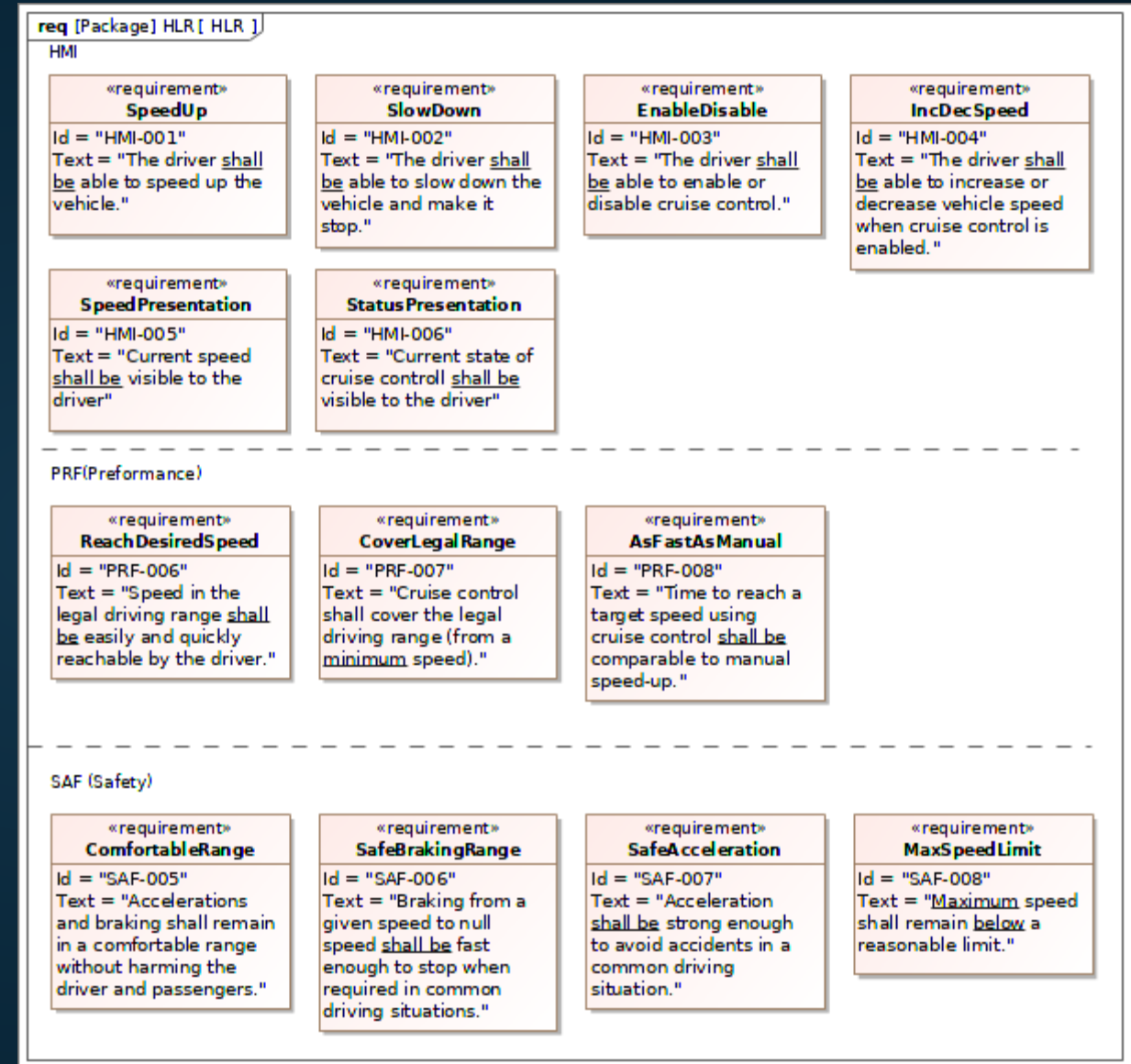
Main Components

- System divided into two main parts:
 1. **PhysicalModel**: implements
 - vehicle simulation and
 - cruise control
 2. **VirtualDashboard**: allows access from PC



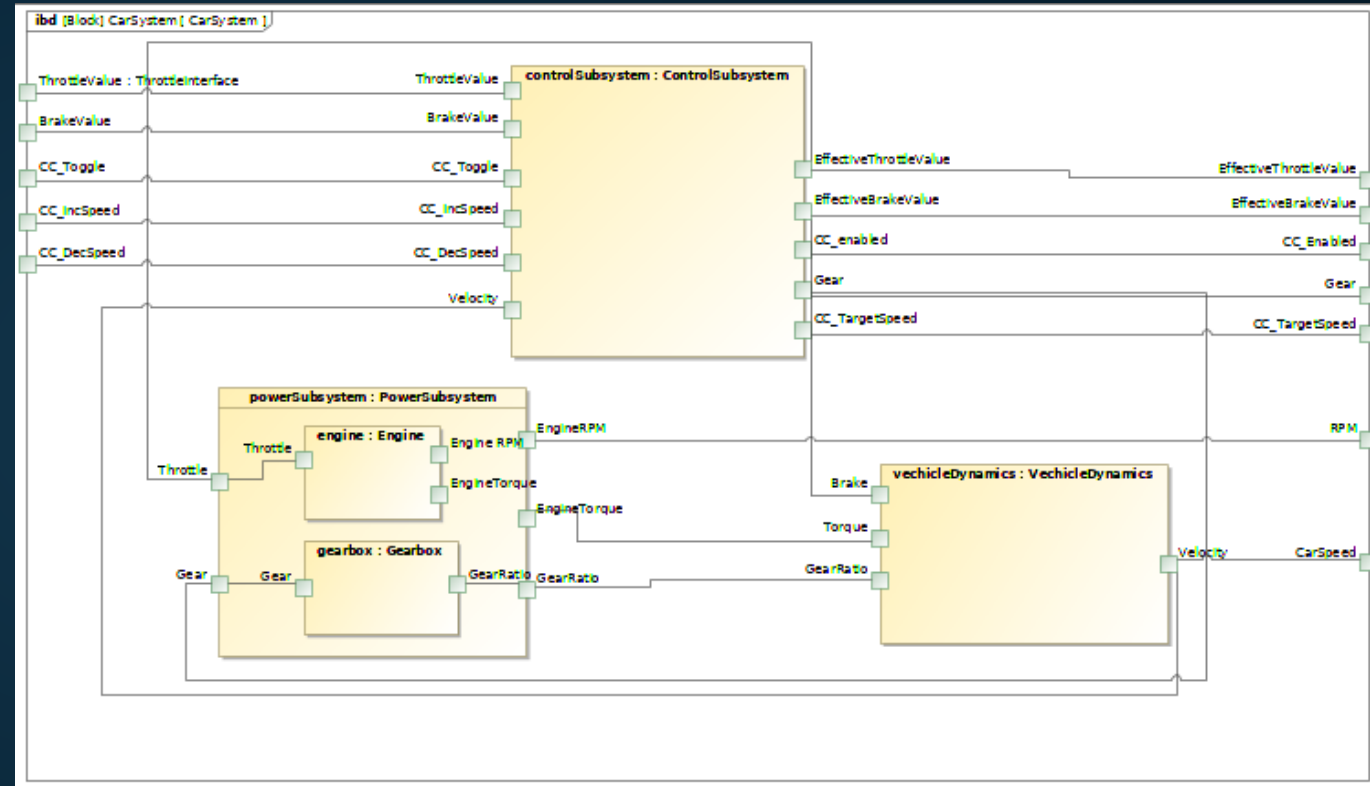
High-level requirements

- Initially, requirements are defined textually
- Formalization of selected subset apply to
 - High-Level Requirements or
 - Low-Level Requirements



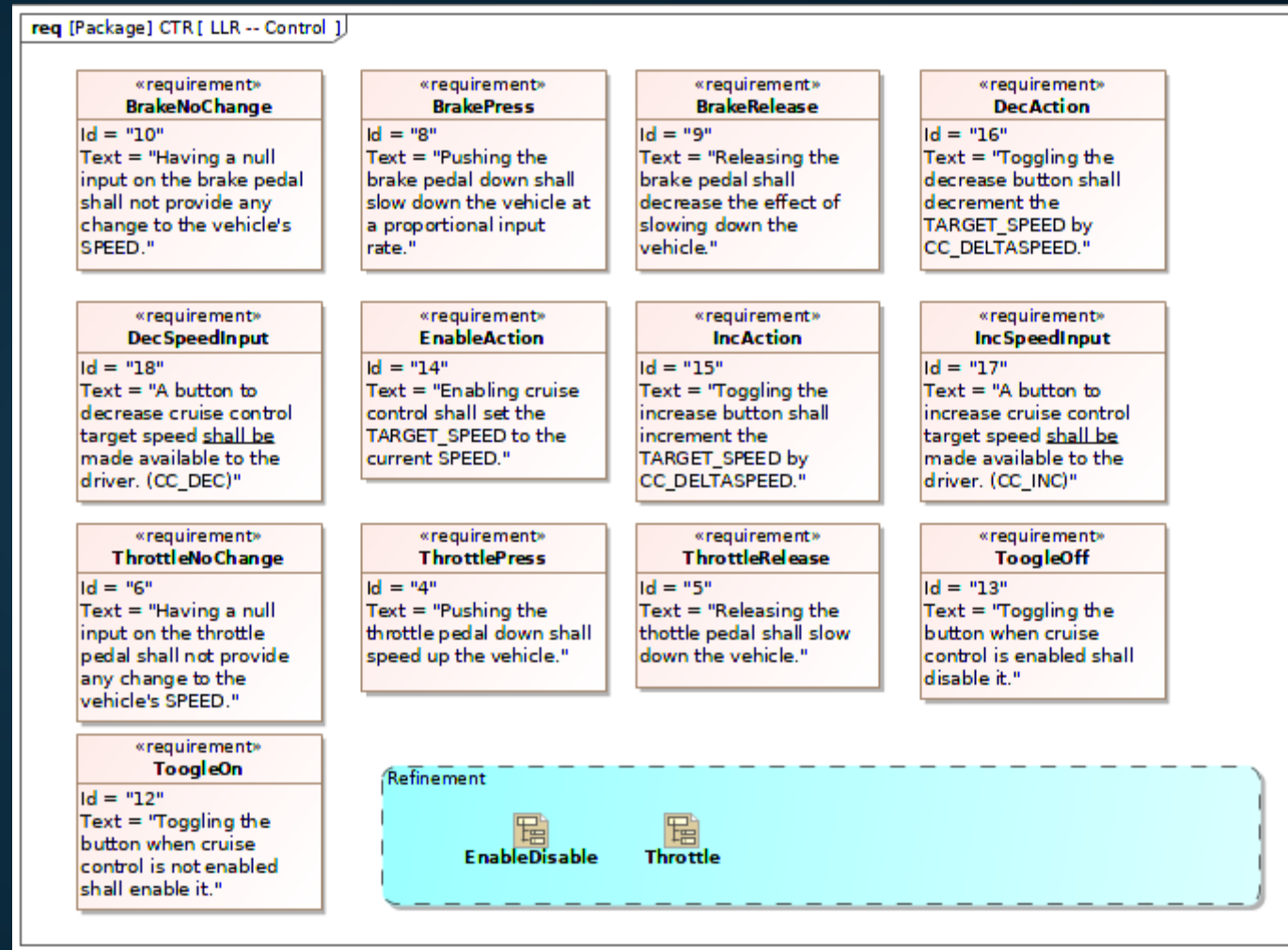
Internal Structure & Data Flows

- Defining the internal structure provides high-level division into software components
- Interface definitions provide names and types for further decomposition and formalization of the requirements



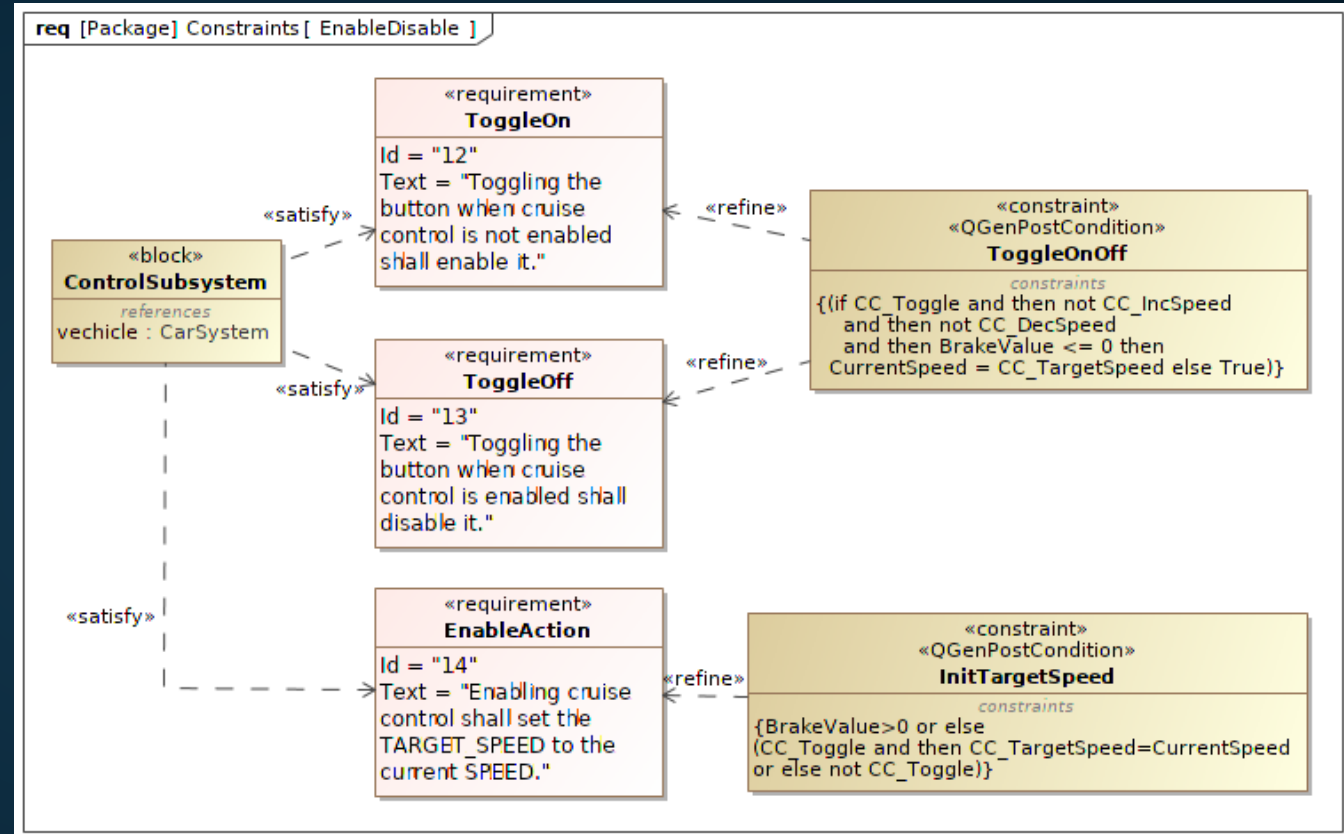
Low-Level Requirements

- Redefine the High-Level Requirements
- Use interface names defined in system high-level architecture
- Specify functional behavior for each component



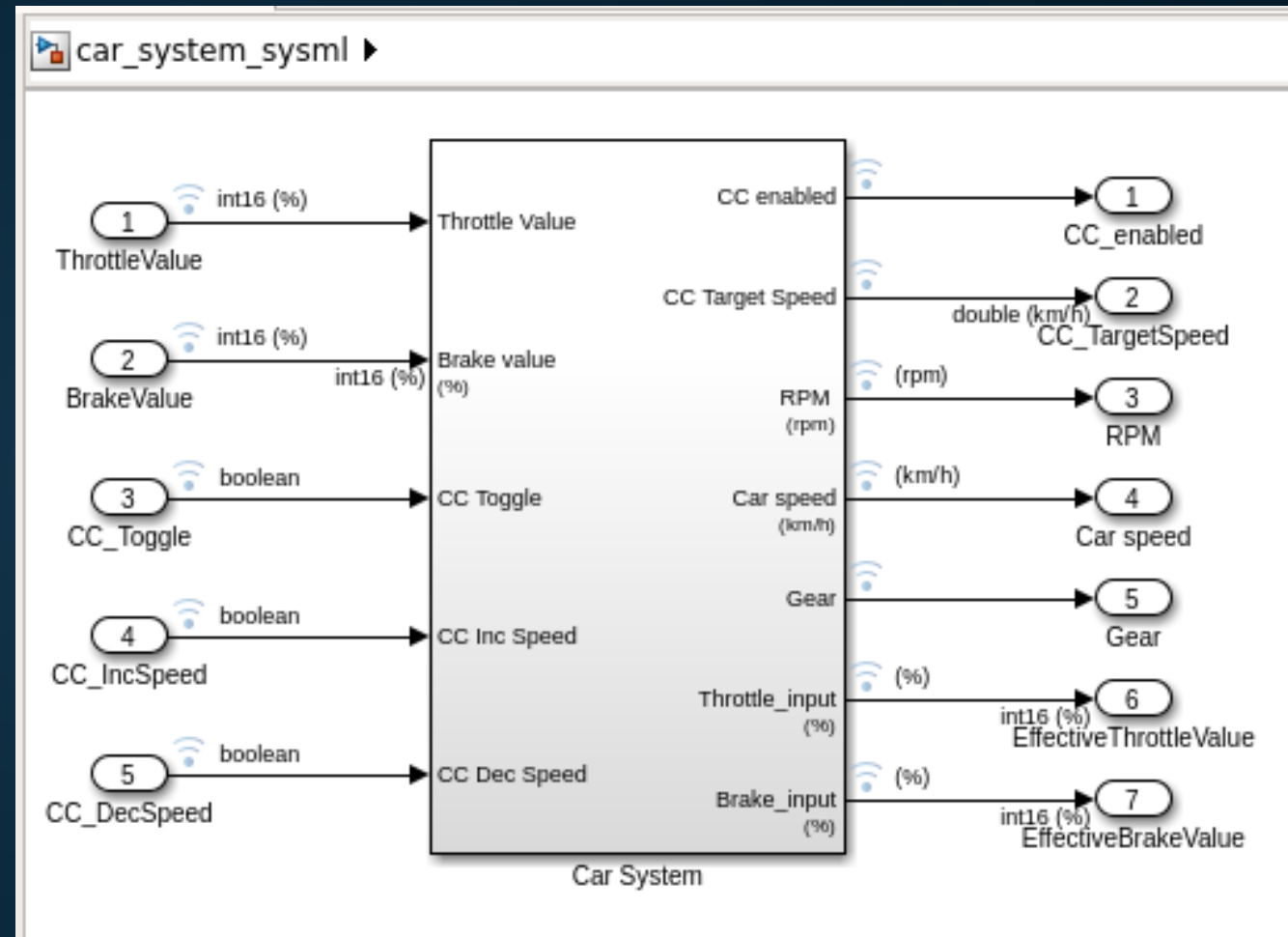
Requirement Formalization

- Rewrite requirements as constraints
 - allows consistency checks between requirements, design, and implementation
- Allocate requirements to components
- Here, we have chosen SPARK as the language for formalization



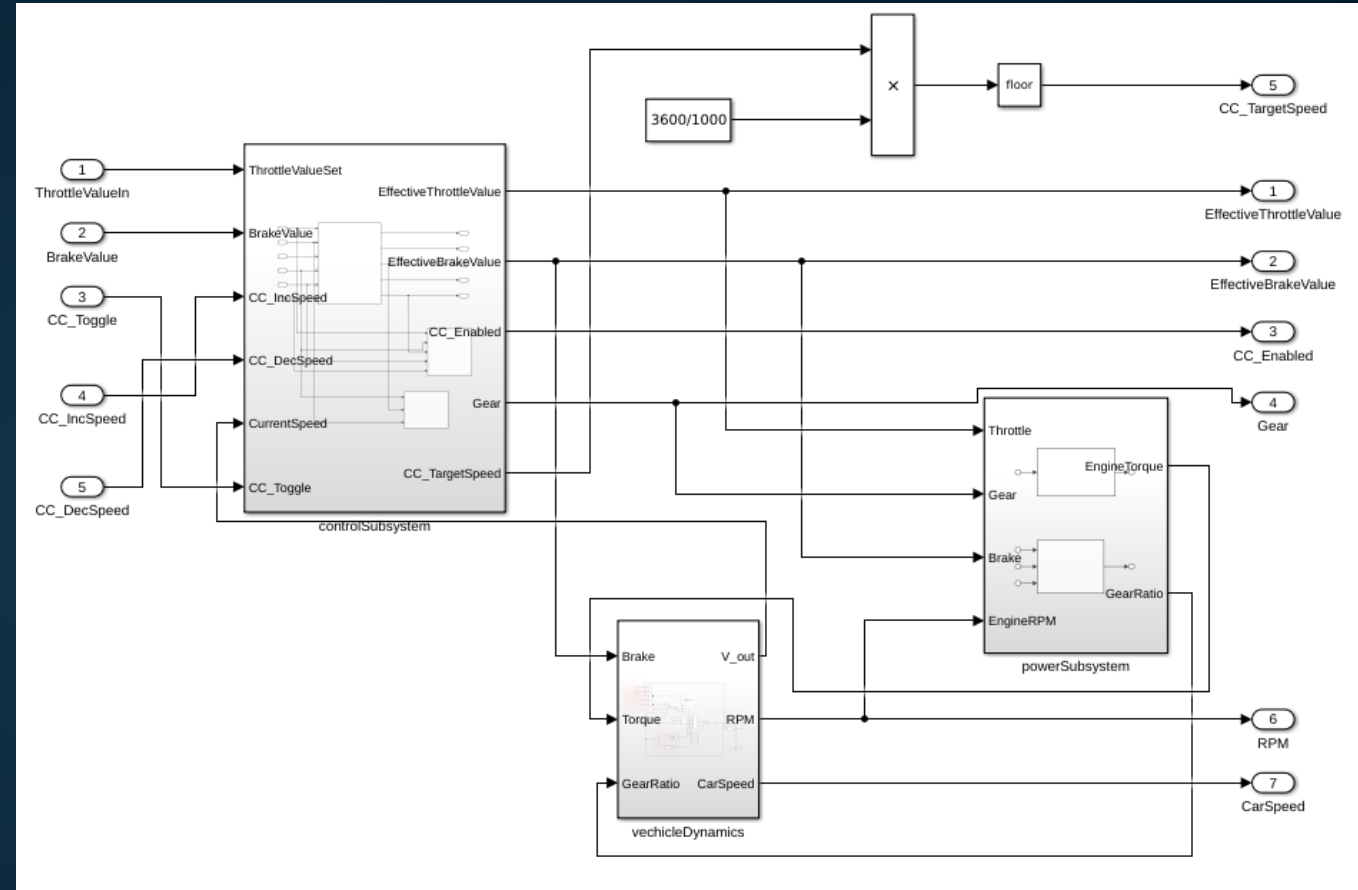
Conversion to Simulink

- Aim of Simulink conversion:
 - provide a skeleton for refining the design by defining computation algorithms
 - validate the system definition by simulation



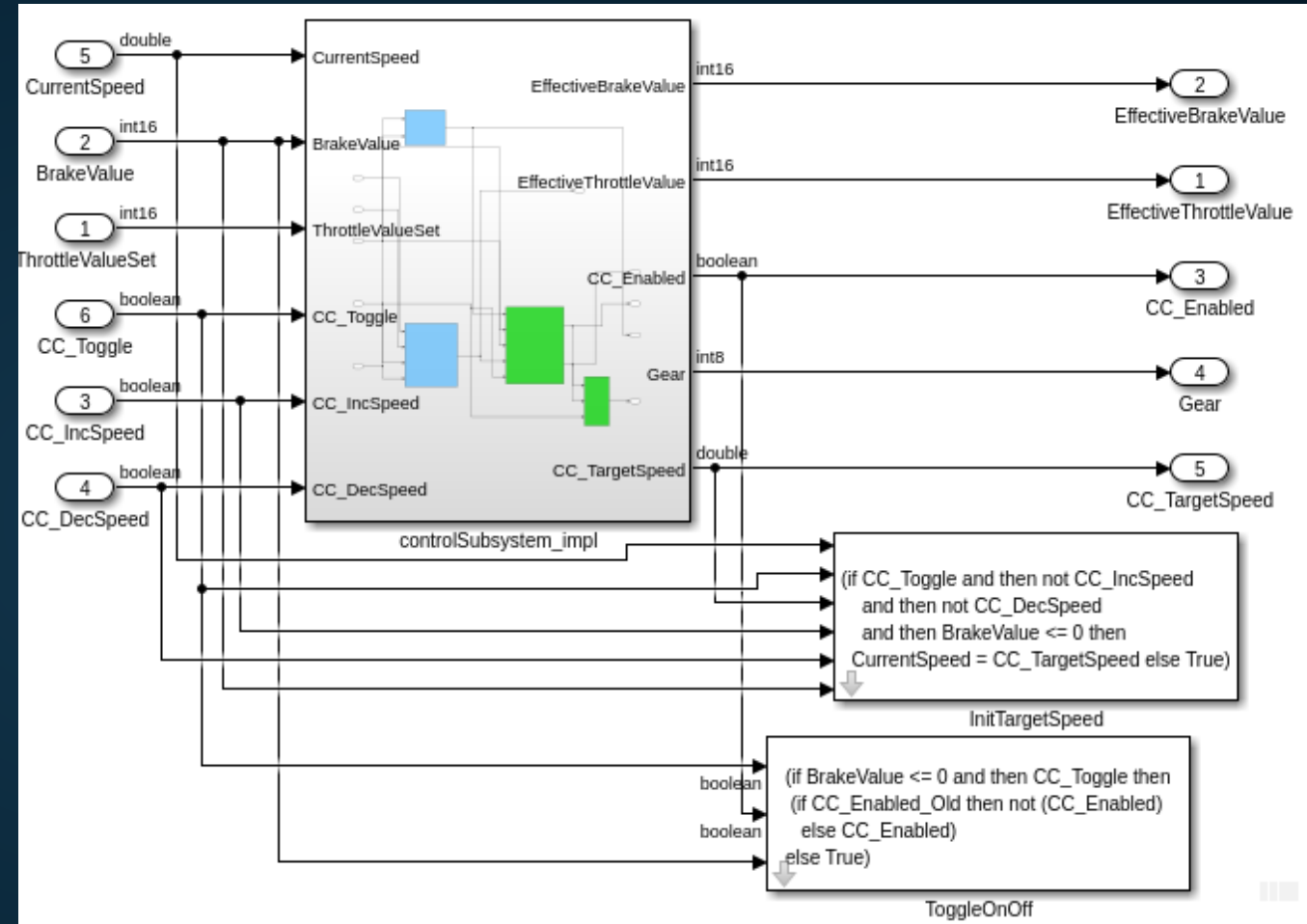
Internal Structure → Simulink

- Convert blocks from IBD to Simulink
- Provide skeletons / containers for
 - control algorithms
 - plant model



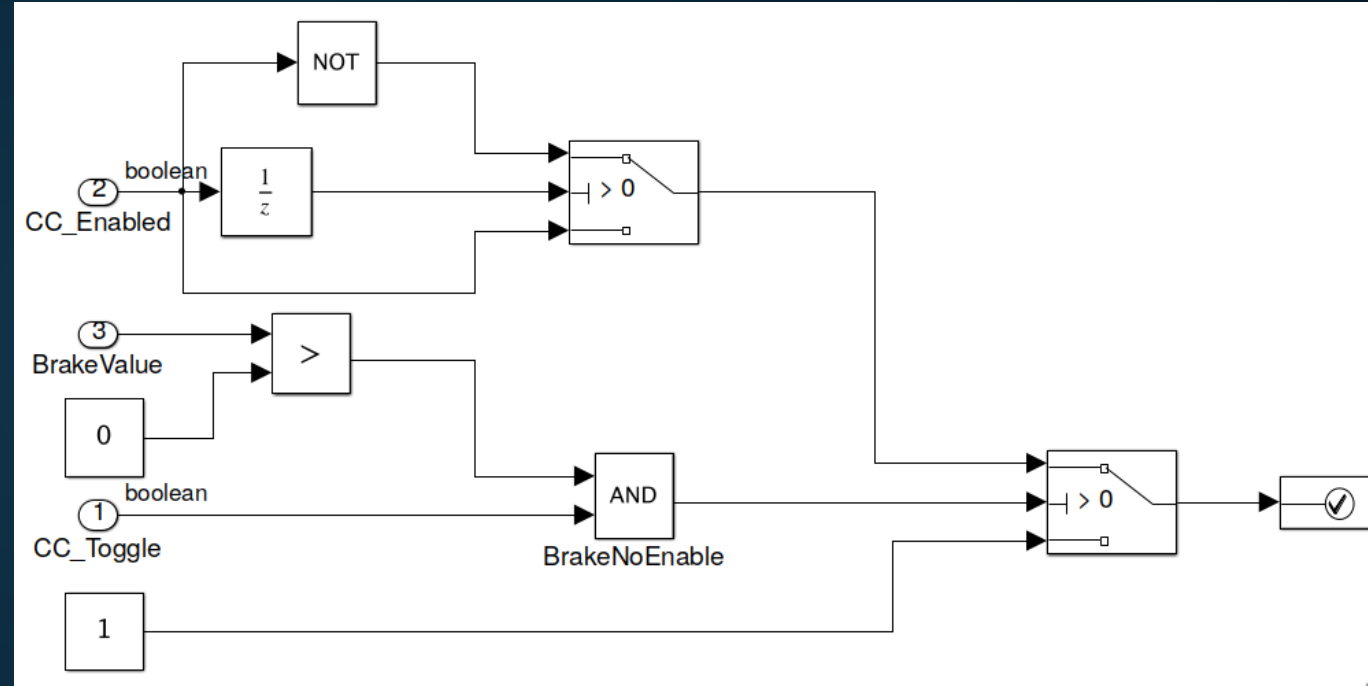
Requirements → Simulink Observers

- The requirements formalized by constraints are inserted in Simulink as synchronous observers
- Block mask tells the code generator that subsystem contents should be handled as a post-condition



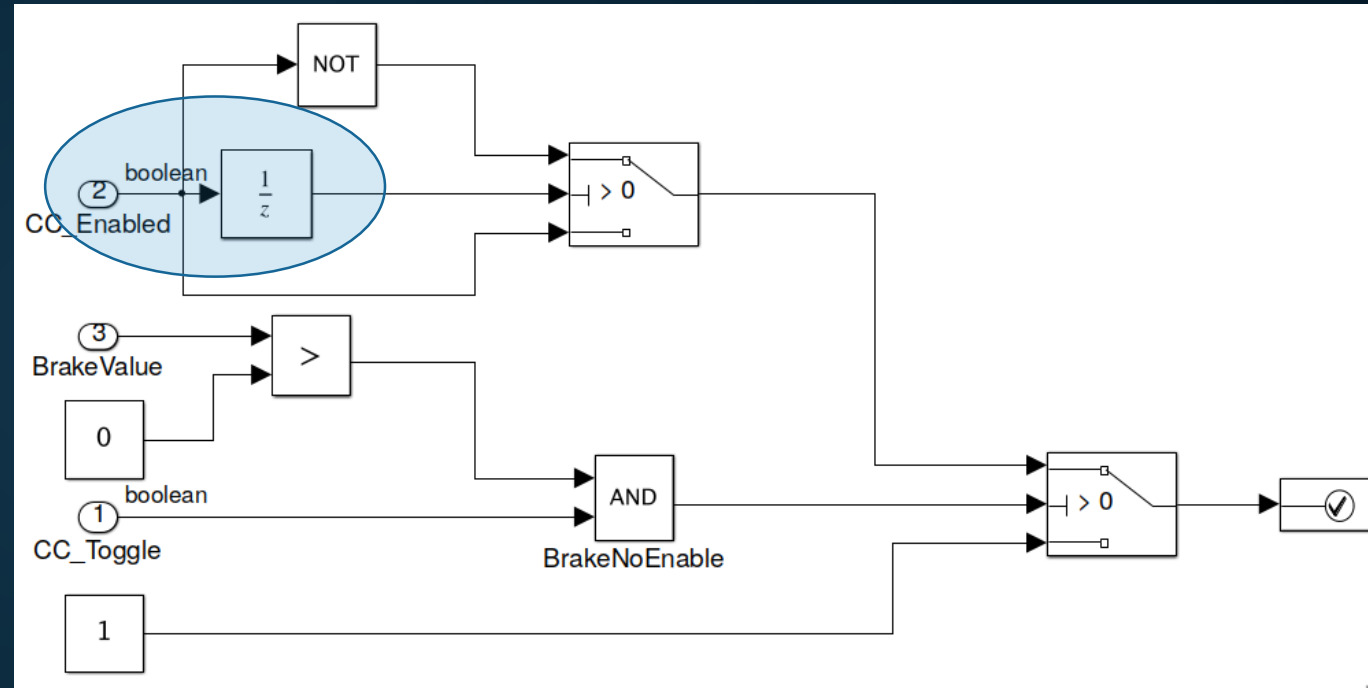
Observer Contents

- A QGen observer is a subsystem that
 - takes signals from functional part of the model as input
 - compares signal values with
 - each other
 - an oracle defined by constraints in SysML
 - raises an exception when comparison fails



Reasoning About Time

- A simplified way of inserting the time in constraints is to refer to previous computation steps
- Here the modeler has a choice to either
 - insert the memory buffer explicitly and refer to this
 - rely on 'Old mechanism in Ada
- To mimic the 'Old behavior in Simulink we use the UnitDelay block



Contract in generated code

- Each observer block is converted to a check function

```
package ToggleOnOff is

  function check
    (CC_Toggle : Boolean;
     CC_Enabled : Boolean;
     BrakeValue : Integer_16;
     CC_Enabled_Old : Boolean)
    return Boolean
  is (if BrakeValue > 0 and then CC_Toggle then
      (if CC_Enabled_Old then not CC_Enabled
       else CC_Enabled)
      else True);

end ToggleOnOff;
```

Contract in generated code

- The check function is called from pre- or postcondition of a functional subsystem
- Internal memory blocks in observers are replaced with 'Old actuals

```
package controlSubsystem is

  procedure initState (State : in out controlSubsystem_State);

  procedure initOutputs (State : in out controlSubsystem_State);

  procedure comp
    (ThrottleValueSet : Integer_16;
     BrakeValue : Integer_16;
     CC_IncSpeed : Boolean;
     CC_DecSpeed : Boolean;
     CurrentSpeed : Long_Float;
     CC_Toggle : Boolean;
     EffectiveThrottleValue : out Integer_16;
     EffectiveBrakeValue : out Integer_16;
     CC_Enabled : out Boolean;
     Gear : out Integer_8;
     CC_TargetSpeed : out Long_Float;
     State : in out controlSubsystem_State)
  with
    Post =>
      (ToggleOnOff.check
       (CC_Toggle, CC_Enabled, BrakeValue,
        CC_Enabled'Old))
      and
      (InitTargetSpeed.check
       (CurrentSpeed, CC_Toggle, CC_TargetSpeed,
        CC_IncSpeed, CC_DecSpeed, BrakeValue));

  procedure up (State : in out controlSubsystem_State);

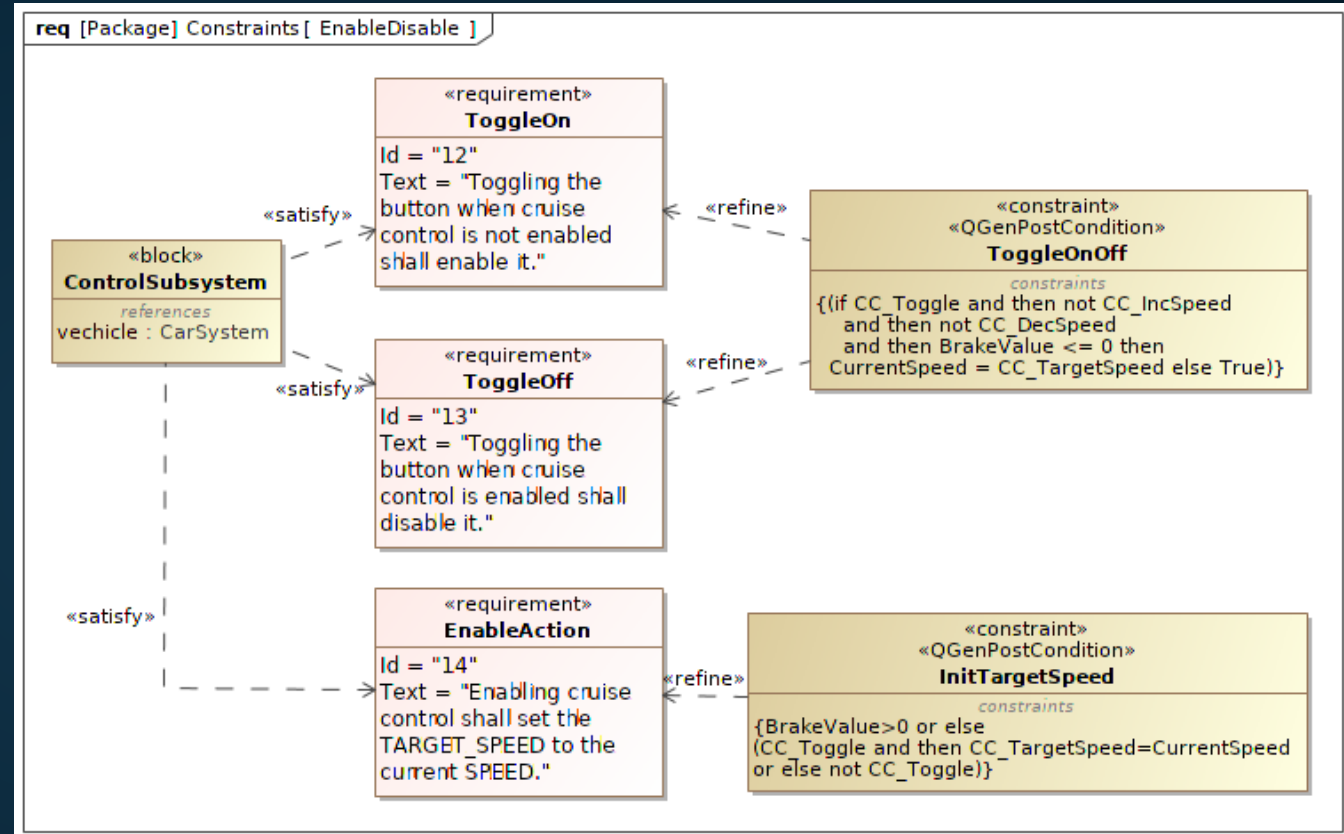
end controlSubsystem;
```

Formalizing requirements

- Parametric diagrams
 - Good for physical phenomena – the „plant model“
 - May need „creative interpretation“ while translating to software constraints
- Activity diagrams/state models
 - Potential candidates for draft algorithm design or test oracle
 - Equivalence proofs not trivial (if possible at all) after refinements in subsequent design steps
- **Constraint blocks**
 - Good form for representing axiomatic definitions of properties and their relationships
 - Easy to carry forward to the next levels and backpropagate changes

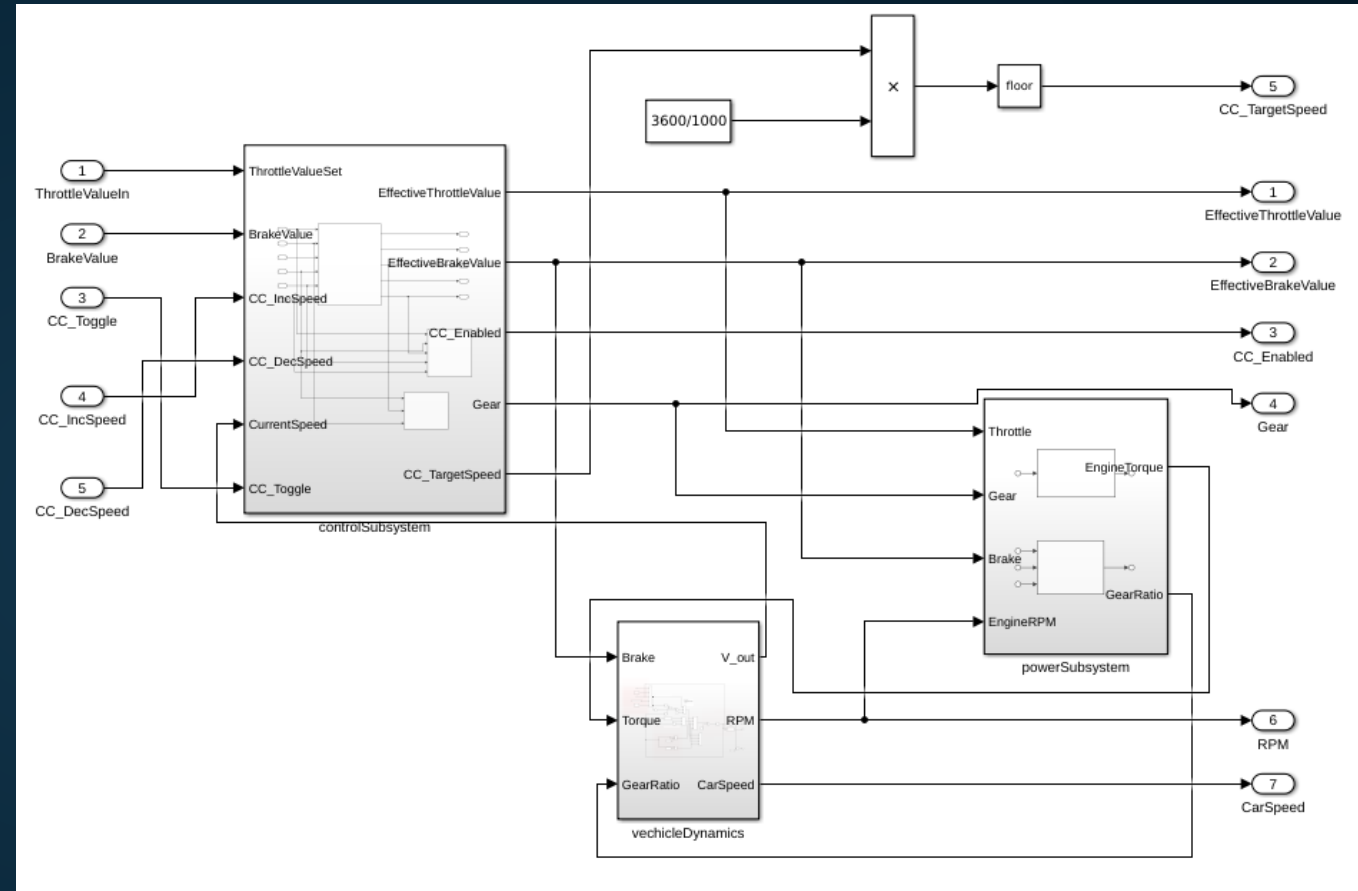
Why SPARK in SysML?

- Looking for axiomatic specifications potentially with late binding
- OCL seems too strictly defined for this purpose (e.g. pre and postconditions bound to behaviors) => using a different language rather than loosening the constraints
- The current converter is easily extensible to support OCL or some other expression language



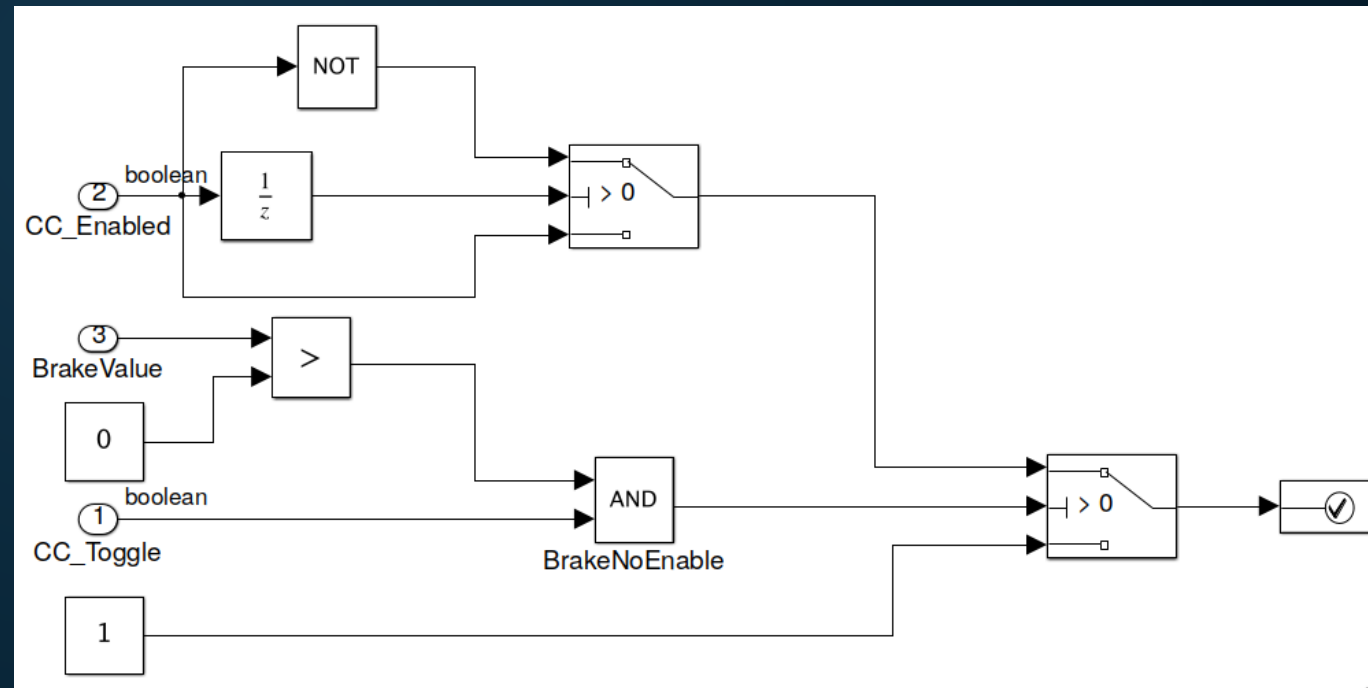
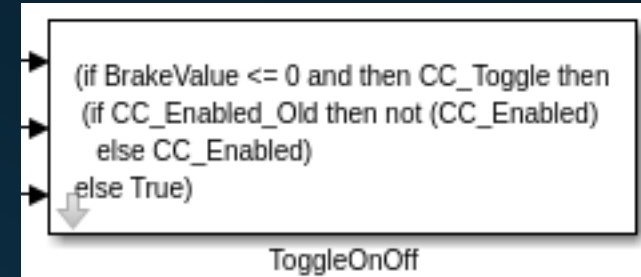
The Role of Simulink

- An appropriate tool for algorithm design
- More natural choice for a control engineer than activity or parametric diagrams
- Qualifiable automated workflow from Simulink to code already exists (QGen)



Observers in Simulink

- SPARK expression would be sufficient for code generation and simulation (using a s-function)
- Difficult to validate and modify in Simulink
- Block diagram simplifies contract refinement at simulation time



Questions / challenges / next steps

- Relation between parametric diagrams and constraints?
- Good workflow for binding the constraint expression with block properties?
- Composability and validation of the constraints
 - First formalization in SysML where the only validation mechanism is review
 - Easy to validate in Simulink or source code but this is too late for systems engineer
 - Achieving completeness assumes iterations between system design and algorithm design
- Support for automatic proof
 - Need for additional hints about code to successfully prove postconditions

Thank you!

AdaCore