KRATES
inseneribüroo • engineering bureau

AdaCore
QGen

Qualifiable code generation backend for TASTE

ESA Contract No. 4000118510/16/NL/CBi

IB Krates OÜ, Estonia

Andres Toom

IB Krates OÜ / AdaCore Estonia OÜ

TEC-ED & TEC-SW Final Presentation Day

11 December 2018 ESA/ESTEC

Outline

- Introduction
  - Project team
  - Background technologies
  - Objectives
- Code generation
  - SDL to C / Ada (SPARK)
  - VDM-SL to Ada (SPARK)
  - SDL + VDM-SL to C / Ada (SPARK)
- Design by Contract workflows
- TASTE / AADL and Simulink Round-Trip
- QGen / SSI roadmap

Introduction

Project team

- Contractor IB Krates OÜ
  - An Estonian systems integration and software development SME.
  - In close collaboration with AdaCore, leading vendor of Ada compiler and development tools.
  - Together created the QGen open source qualifiable code generator for embedded high-integrity and safety-critical domains.
  - ESA project (2014-2015): Integrating the QGen (GMS-P) Automatic Code Generator with the Space Component Model (Integrating QGen in TASTE).
- AdaCore Estonia OÜ
  - In 2018 the code generation product line was acquired by AdaCore and a new entity Estonia OÜ was created for strengthening the QGen-related developments.

Background

- Follow up of the ESA-PECS project
  - Integrating QGen (GMS-P) with the Space Component Model (SCM)
  - Provided integration of the QGen code generator into the AADL-Simulink workflow of TASTE
- Existing technologies and tools
  - Component-based modelling (TASTE toolset)
  - Architectural modelling (AADL, ASN.1, …)
  - Functional modelling (Simulink, SDL, ...)
  - Code generators (Ocarina, Simulink Coder, QGen, ...)
  - Explicitly coded components (C, Ada, ...)
  - ITU Specification and Description Language (SDL)
  - OpenGEODE tool
  - Vienna Development Method (VDM)
  - Overture tool
- What is added
  - QGen for code generation from SDL and VDM-SL
  - Additional component-based modelling workflows using the above

What is QGen? (1/2)

Trusted Code Generator

- **From Simulink® & Stateflow® to Ada SPARK** (Ada subset) **or MISRA C**
- **Customizable** code generation
- Qualification for DO-178C at Tool Qualification Level 1 ongoing
- **Consistency** of the generated code and the Simulink® simulation

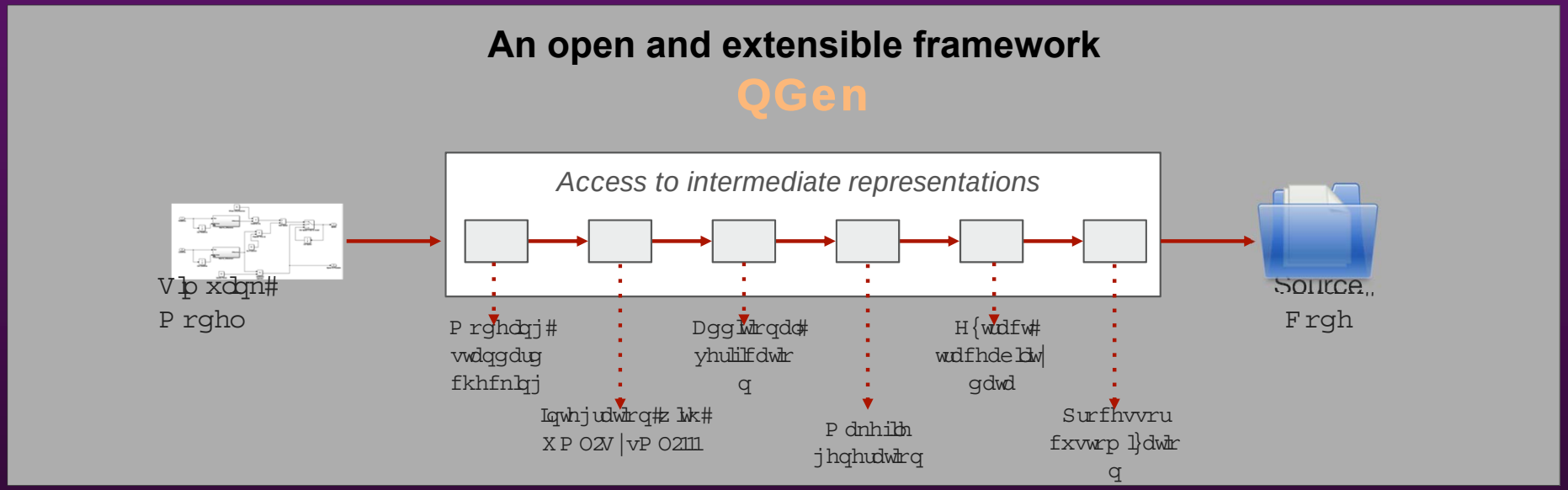
Model Verifier

- Formal static model verifier for runtime errors and functional properties
- Aiming for DO-178C at Tool Qualification Level 5

Integrated Model-Based Development Toolset

- **Model-level debugger**
- Processor-In-the-Loop testing

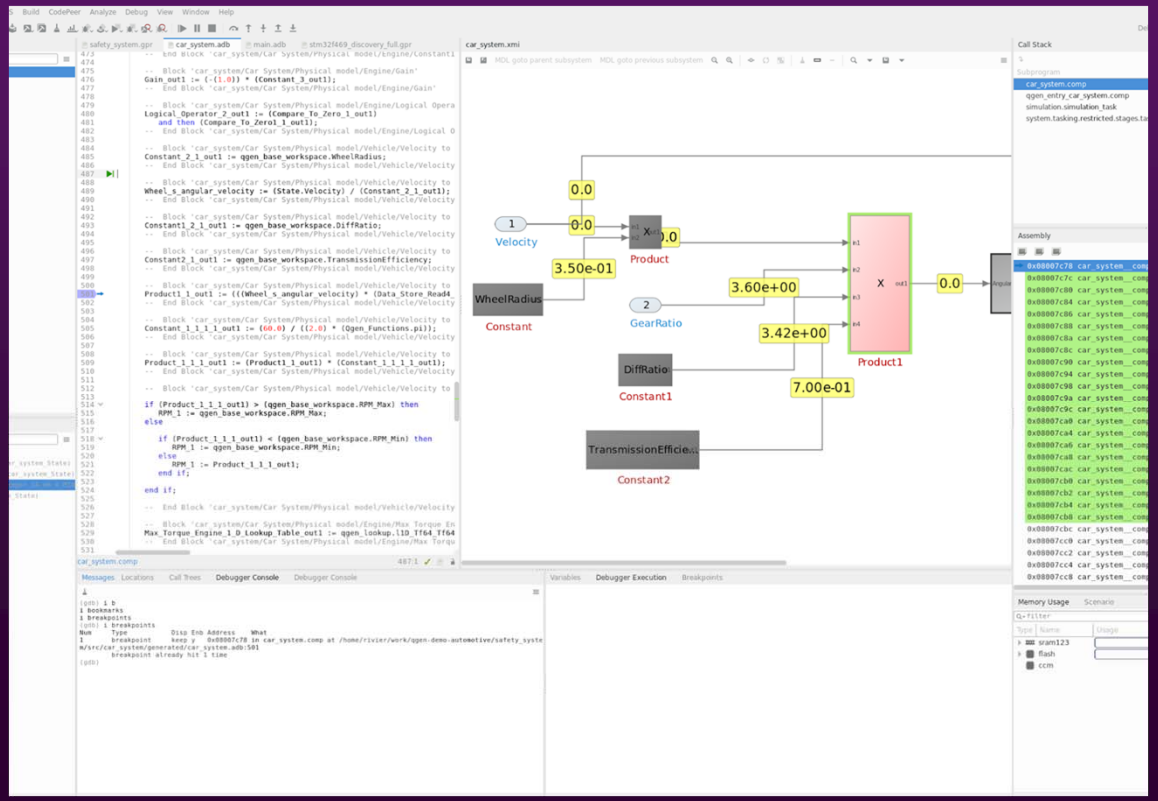
What is QGen? (2/2)



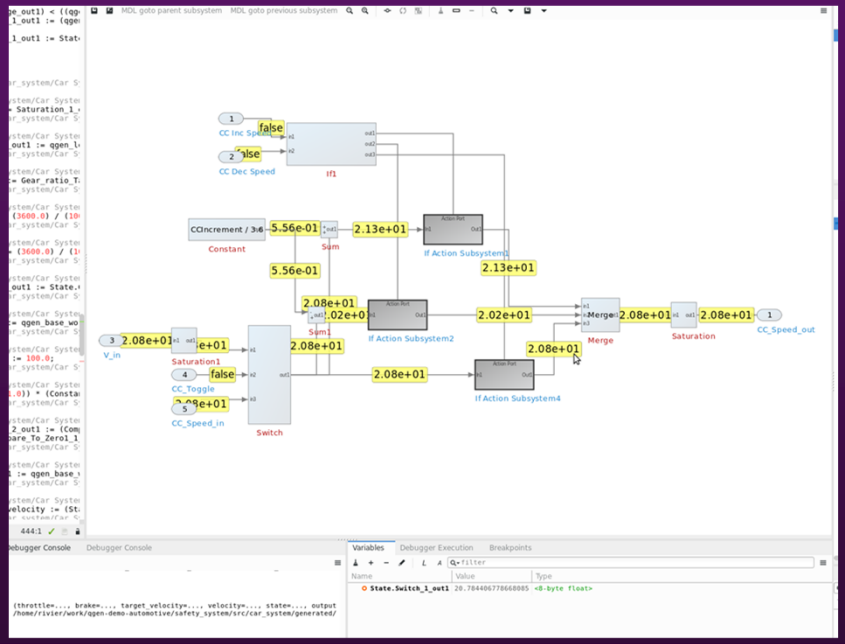
- "The gcc for modeling languages"
- Designed to accept multiple languages in input/output, including in-house DSLs
- A single code generation style/strategy for all of your modeling languages
- XMI-based model import/export at different abstraction levels

QGen Debugger – Unique Model Debugging Capability

- Bridge the gap between control engineering and software engineering
- Integrate code generated from Simulink models with code written manually
- Analyse model behavior by stepping through source code
- Display side-by-side synchronized view of Model, Source Code and Object Code
- Execute code on Host, Local Emulator or even your Embedded Target
- Currently, limited Stateflow support (planned for 2019)

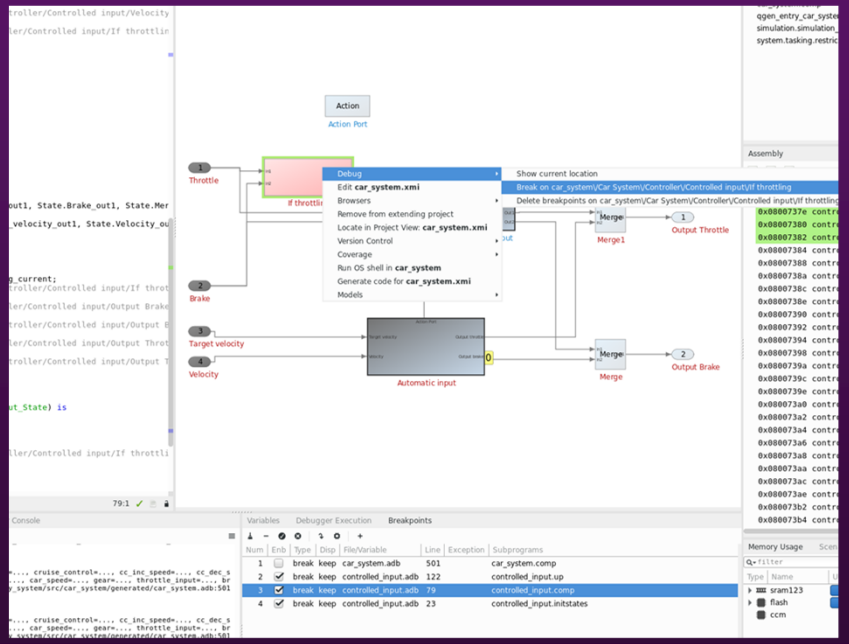


QGen Debugger Features



Display signal values dynamically

- Examine values in Variables view
- **Set persistent values**
- Log values to file



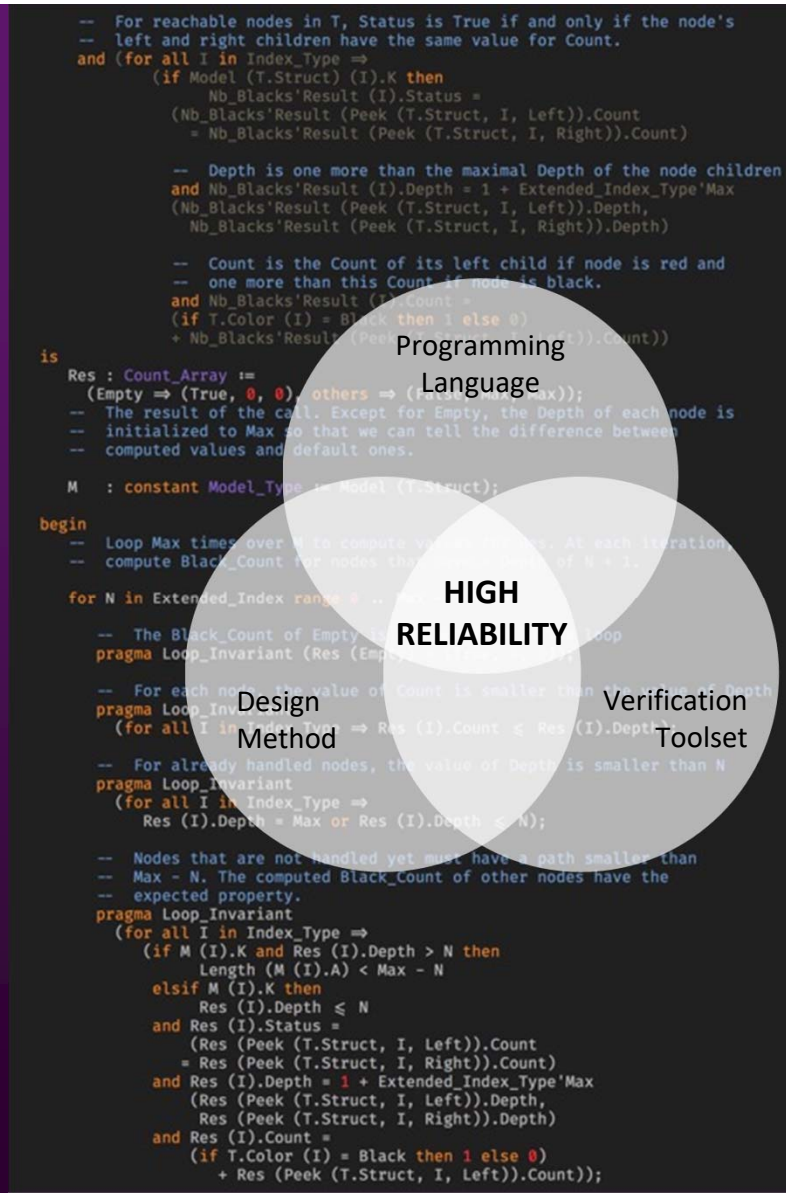
Insert breakpoints on blocks or model references

- Blocks highlighting
- Easily switch between code and model or debug side-by-side

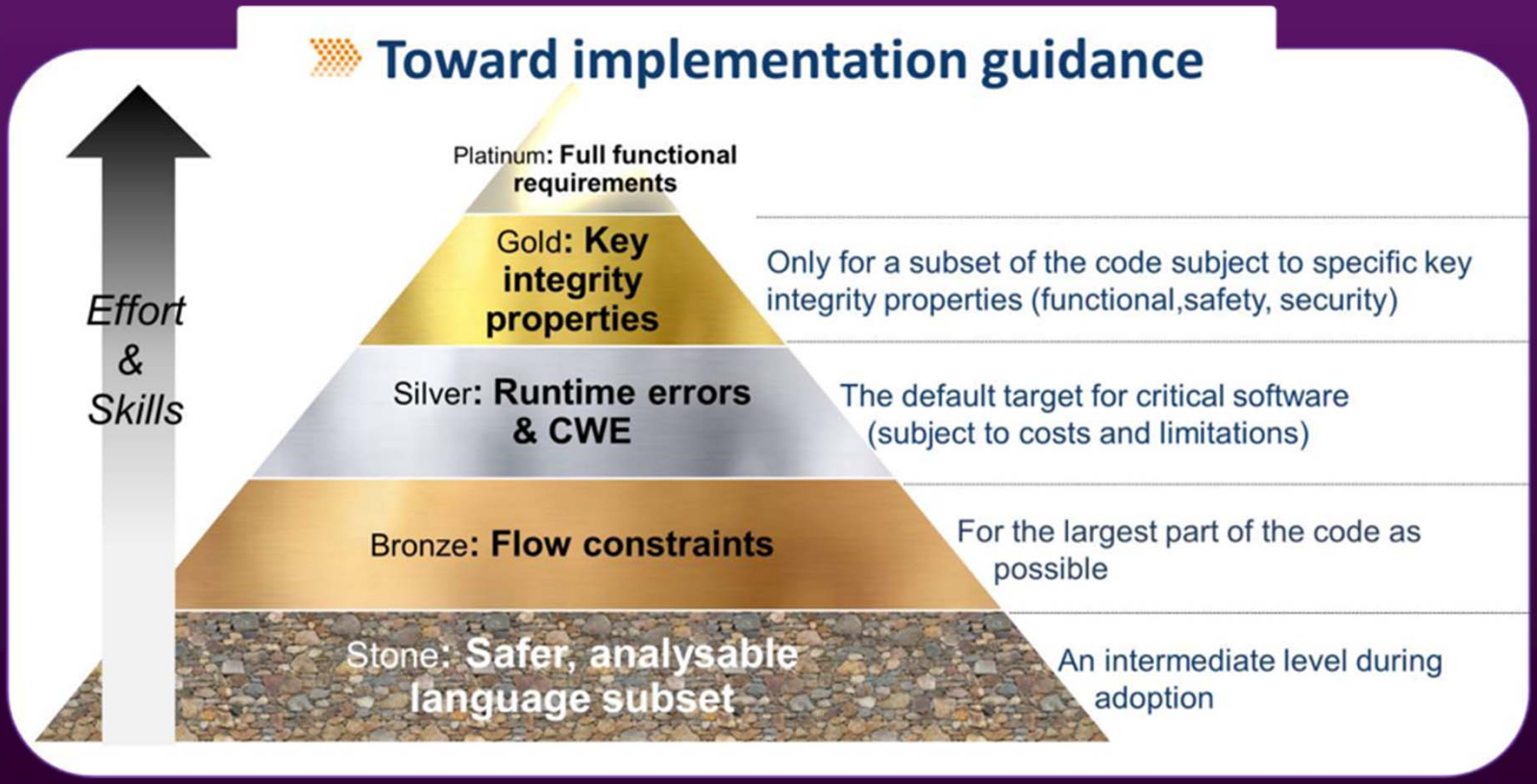
SPARK 2014

A programming language that includes a specification language

- *Specification* and *implementation* defined for each program module — in the same language.
- Answers specific problems for embedded systems developers
  - closes gap between formal specifications and code
  - fewer coding errors
- Program your *specification* and your *proofs!*



Practical Application of Formal Methods with SPARK



QGen for TASTE & Simulink

- ESA-PECS Project (IB Krates, 2013-2015)
  - "Integrating the QGen Automatic Code Generator with the Space Component Model"
  - Improved code generation from Simulink®/Stateflow® in TASTE
  - Code generation driven by a single build script, less manual steps, fully repeatable
  - Direct use of native data types defined in ASN.1
  - Less buffers required. Cleaner glue code
  - Code generation with formal verification support
  - CodePeer and SPARK integration
  - Dedicated support for on-target regression testing
  - Comparison against stored simulation results. No need for external IO or software
  - Comparative study of the DO-178C and ECSS based qualification

KRATES
inseneriburoo • engineering bureau

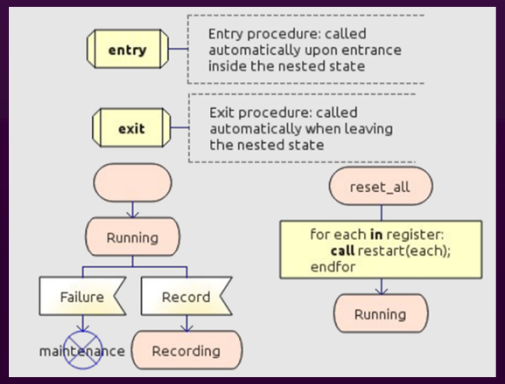
Objectives of the Current Project

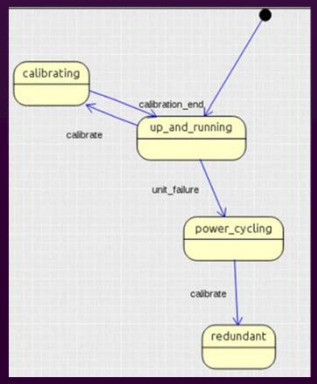
- O1 (Main objective)
  - Provide a universal and qualifiable code generation backend for the two main models for behavioural specifications in TASTE: SDL and VDM based on the QGen toolset.
- O2 (Supports O1)
  - Support the simulation and debugging of SDL + VDM-SL models from the TASTE/OpenGEODE environment based on the QGen code generation backend.
- O3 (Extends O1)
  - Develop an approach for specifying high-level formal properties (contracts) for a component in VDM
  - transforming and propagating those contracts to detailed design models (e.g. Simulink) and/or generated program code (e.g. SPARK Ada).
  - using these contracts for automatic consistency checking between the high-level specification, design and implementation.
- O4 (General enhancement)
  - Improve the mapping between architectural model in AADL and architecture elements expressed in Simulink models.
- O5 (Validation and verification)

Code Generation from SDL

SDL - Specification and Description Language

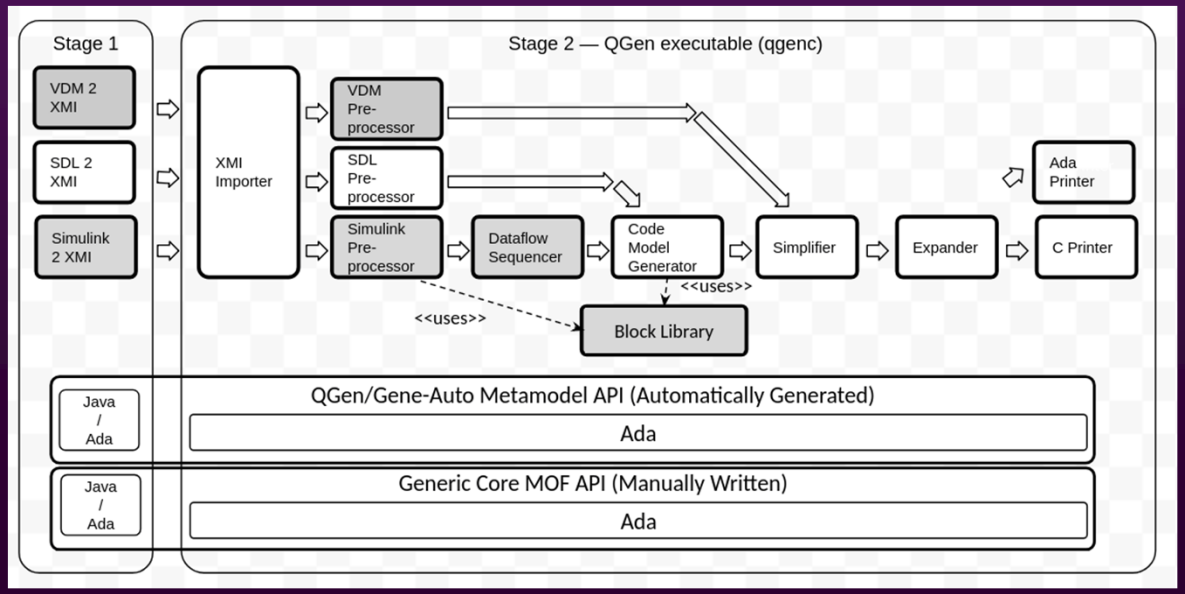
- ITU-T standardised (Z.100 .. Z.106) formal language for the specification of the behaviour of reactive systems, such as real-time systems
- Complete and unambiguous formal semantics making it suitable for high-level design as well as low-level design and automatic code generation
- Supports the ASN.1 language for the specification of datatypes
- Tools
  - PragmaDev Studio (RTDS) -- Commercial
  - OpenGEODE -- Open Source. Integrated in TASTE





Code generation from SDL with QGen

- Ada and C code generation from a subset of SDL (with ASN.1 datatypes)
- Pivot language: Extended Gene-Auto/QGen metamodel (Ecore)
  - Allows for standardised XMI-based model exchange with external tools



QGen SDL Converter — Frontend (Java)

- 1st stage: ANTLR 4-based importer
  - SDL importer is based on the OpenGEODE ANTLR grammar (sdl92.g)
  - ASN.1 data definitions are imported based on the XML data structure produced by the ASN1SCC tool
- 2nd stage: the ANTLR tree is converted to the Gene-Auto/QGen Ecore metamodel.
  - Uses an Eclipse Modeling Framework (EMF) provided core features.
  - Model is serialized as an XMI file

QGen SDL Converter — Backend (Ada)

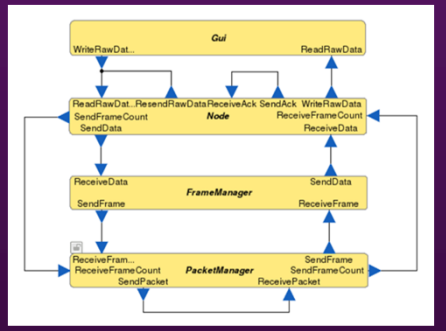
- Code generation with QGenc
  - Input: Model XMI
  - Some preprocessing of SDL models to be structurally closer to Stateflow models
  - Type inference extensions for sequence types, octet types
  - Backend support for SDL operators/functions: append, length, write, writeln
  - Improved support for custom types and slice operations
  - New expansion and postprocessing steps. Mainly related to sequences
  - Output: Ada or C code

SDL Features and Limitations

- Supported features
  - Most of the features that are supported in OpenGEODE
- Unsupported features
  - SDL type (process) instances, processes with formal parameters
  - Use clauses to non-ASN.1 external modules
  - Parameterized ASN.1 types, IA5String ASN.1 type
  - Continuous signals with explicit priority value
  - Informal text
  - Non-deterministic choice (any)
- Other limitations
  - The same variable cannot be passed simultaneously to an in and out/in-out parameter or multiple out/in-out parameters when Ada code is generated
  - ASN.1 set type definition is supported, but the ASN1SCC set implementation is not (Ada Functional Sets are supported instead)
  - QGen does not detect all the same errors as OpenGEODE does. It is assumed that the model was checked before the code generation step.

Case Study 1 - TUT Nanosatellite Case Study

- Tallinn University of Technology (TUT) Nanosatellite project.
  - TCTM protocol for ground – satellite communication
  - Based on AX.25 (amateur radio)
  - Modeled by the master student Dan Rodionov
- Results
  - Ada code was generated using OpenGEODE
  - C and Ada code was generated with QGen
  - Functional verification of the generated code wrt. OpenGEODE simulation
  - Some model-level issues were detected during the process and fixed
    - E.g. missing initialization of local and out variables
  - Identical behaviour between OpenGEODE and QGen generated code was achieved after that
  - The performance of QGen generated code was somewhat lower from OpenGEODE's



Case Study 2 - OpenGEODE Regression Testsuite

- Case study 1 (TUT Nanosat) fully supported
- Case study 2 (OpenGeode regression testsuite)
  - Ada code generation: 71% passes OpenGEODE testsuite
  - C code generation: 64% passes OpenGEODE testsuite

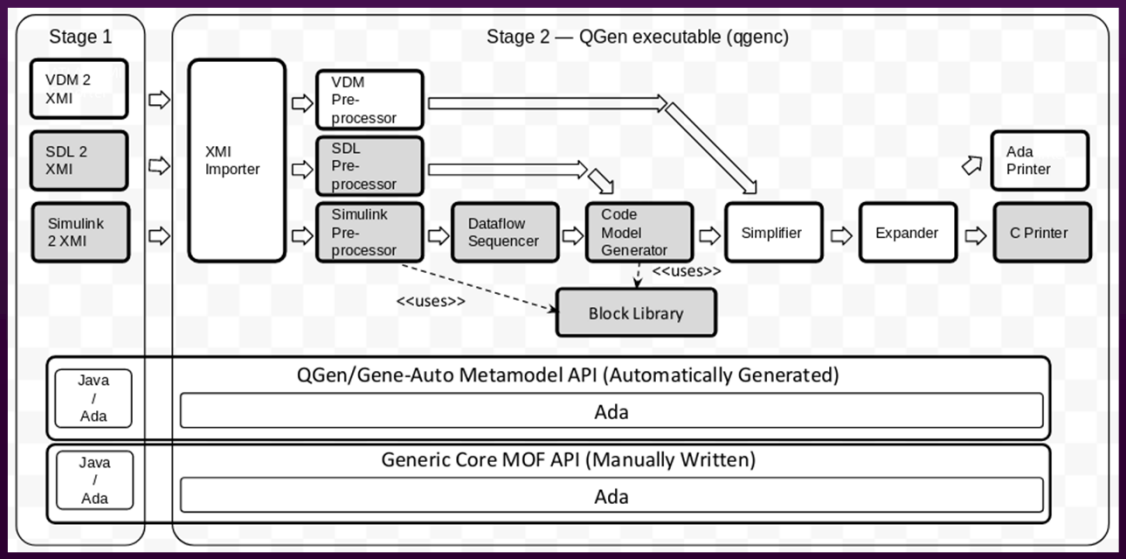
Code Generation from VDM-SL

VDM - Vienna Development Method

- VDM is a formal method that has a long history for the development of computer-based systems.
  - The specification of the first PL/1 and Ada compilers
- ISO standardized syntax and semantics.
- The core concepts of VDM are formalised in the VDM Specification Language (VDM-SL) and extended by object-oriented features in VDM++ and real-time concepts in VDM-RT.
- The VDM-SL language features include the specification of
  - Data types, functions and operations on data
  - Powerful and easy to use collections (sets, sequences, mappings)
  - Functions and operations can be defined *implicitly* through contracts and/or *explicitly*
- Tools: VDMTools (commercial), Overture (open source).
- VDM is one of the functional languages supported by TASTE.

Code generation from VDM-SL with QGen

- Ada SPARK code generation from a subset of VDM-SL (with VDM-SL native and ASN.1 datatypes)
- Pivot language: Extended Gene-Auto/QGen metamodel (Ecore)
  - Allows for standardised XMI-based model exchange with external tools



VDM-SL to SPARK Converter — Frontend (Java)

- Relies heavily on the Overture toolset
- Imports information from the intermediate Abstract Syntax Tree (AST)
- Uses the AST visitor pattern provided by Overture for creating QGen/Gene-Auto metamodel objects
- The QGen/Gene-Auto model is serialized to an XMI file

VDM-SL to SPARK Converter — Backend (Ada)

- Input: (Code Model) XMI
- Preprocessing of VDM-SL models
- Generic code model expansion and post-processing
- Output: Ada SPARK code

Code Generation from VDM-SL (1/2)

- Code generator implemented
  - Case study 1: (finnuc) supported
  - Case study 2: (AlarmSL) supported with reduced model
    - Sequence definitions must be given in ASN.1
    - Record pattern matching is not supported (can be avoided)
    - Implicit functions are not supported
    - Type invariants are not generated
- Supported features (next slide)

Code Generation from VDM-SL (2/2)

- Supported VDM-SL features
  - Basic VDM-SL types: real, int, nat, nat1, bool. Composite types (records)
  - Unions of quote types (enumerations)
  - ASN.1 type definitions for the same subset
  - Token types with literal values.
  - Flat and multi-module specifications
  - Most common unary and binary operations: +, -, *, …
  - State and value definitions
  - Assignments, Let statements
  - Operations and explicit function definitions
  - Pre- and postconditions
  - Map and Set types. Based on Ada Functional Containers. Most operators supported
  - Sequence types. Based on ASN.1 / ASN1SCC. Limited support

Code Generation from SDL + VDM-SL

Code Generation from SDL + VDM-SL

- In the TASTE modelling philosophy it is possible to create heterogeneous models that has components implemented in SDL and other languages that provide a certain C language interface for binding the respective Provided and Required Interfaces (PI/RI) of the components.
- The VDM2SPARK code generator supports this pattern.
- To ensure that the data types at the interfaces are compatible they must be defined using a common ASN.1 specification for both models.
- When the switch --taste-interface is passed to the VDM2SPARK tool, then all the formal parameters of Ada functions or procedures generated from VDM-SL operations (but not from functions) have an additional export with C conventions pragma and the argument passing mode will be changed from IN to IN OUT.
- This effectively means that values can be passed to these subroutines by reference (pointer) removing memory overhead that would be otherwise caused by interfacing.
- The C name of the subroutine will have an additional prefix “PI_” due to comply with the TASTE naming conventions (PI - Provided Interface).
- The function names don’t have to be compatible. It is possible to use lightweight glue code for binding the interfaces.

Code Generation from SDL + VDM-SL (Example)

- ASN.1 definitions must be converted to VDM-SL

File DataView.asn

```
TASTE-BasicTypes DEFINITIONS ::=
BEGIN

T-UInt32 ::= INTEGER (0..4294967295)

TASTE-Peek-id ::= INTEGER (0..4294967295)

FixedIntList ::= SEQUENCE (SIZE(3)) OF TASTE-Peek-id

MyEnum ::= ENUMERATED {one, two, three, four, five}

MySimpleSeq ::= SEQUENCE { a INTEGER (0..255), b BOOLEAN, c MyEnum }

END
```

File dataview.vdmsl

```
module TASTE_BasicTypes
exports all
definitions
types
   TASTE_Peek_id = int
   inv x ==
      x >= 0 and x <= 4294967295;

   FixedIntList = seq of TASTE_Peek_id
   inv x ==
      len x = 3;

   MyEnum = <one> | <two> | <three> | <four> | <five>;

   MySimpleSeq_a = int
   inv x ==
      x >= 0 and x <= 255;
   MySimpleSeq ::
      a : MySimpleSeq_a
      b : bool
      c : MyEnum
   inv mk_MySimpleSeq(a, b, c) ==
      a >= 0 and a <= 255;

values

end TASTE_BasicTypes
```

```
$ asn1.exe -customStgAstVersion 4 -customStg vdmsl.stg:dataview.vdmsl DataView.asn
```

Code Generation from SDL + VDM-SL (Example)

- An example VDM-SL specification that uses these types

File compute.vdmsl

```
module Compute
imports from TASTE_BasicTypes all, from IO all
exports all
operations
        VDM : TASTE_BasicTypes`FixedIntList ==> TASTE_BasicTypes`FixedIntList
        VDM  (inp)
          == return inp;

        VDM2 : TASTE_BasicTypes`TASTE_Peek_id ==> TASTE_BasicTypes`MySimpleSeq
        VDM2  (inp)
          == return mk_TASTE_BasicTypes`MySimpleSeq(inp, false, <two>)
        pre inp >= 0 and inp <= 255
        post RESULT.a > 0;

        Run : TASTE_BasicTypes`TASTE_Peek_id ==> ()
        Run (x) ==
        (
          IO`println("VDM Run: Started");
          IO`print("Calling vdm2 with arg ");
          IO`println(x);
          IO`println(VDM2(x));
          IO`println("VDM Run: Ended");
        );
end Compute
```

Code Generation from SDL + VDM-SL (Example)

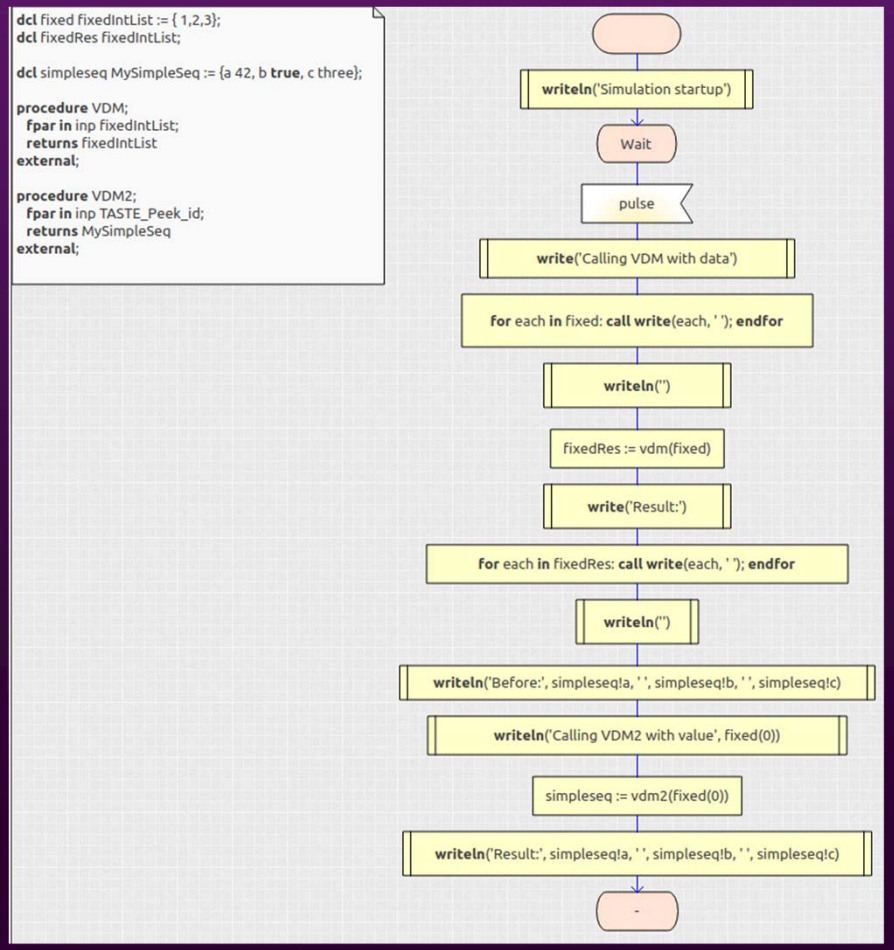
- Generated Ada SPARK specification

Generated compute.ads (with DataView.asn and --taste-interface)

```
with adaasn1rtl; use adaasn1rtl;
with TASTE_BasicTypes; use TASTE_BasicTypes;
with Interfaces; use Interfaces;

package Compute is

   function VDM
      (inp : in out asn1SccFixedIntList)
      return asn1SccFixedIntList;
   pragma Export (C, VDM, "PI_VDM");

   function VDM2
      (inp : in out asn1SccTASTE_Peek_id)
      return asn1SccMySimpleSeq
   with
   Pre =>
   (((inp) >= (0)) and then ((inp) <= (255))),
   Post =>
   ((VDM2'Result.a) > (0));
   pragma Export (C, VDM2, "PI_VDM2");

   procedure Run (x : in out asn1SccTASTE_Peek_id;
   pragma Export (C, Run, "PI_Run");

end Compute;
```

Code Generation from SDL + VDM-SL (Example)

- SDL model



Code Generation from SDL + VDM-SL (Example)

- The following simple glue code in the C language is used to combine the code generated from SDL and VDM-SL models

glue.h

```
/* Glue code for binding code generated from SDL, VDM-SL and ASN.1 */

#ifndef GLUE_H
#define GLUE_H

/* Data type definitions generated by ASN1SCC */
#include "dataview-uniq.h"

/* External functions */
/* Exported from the VDM-SL model compute.vdmsl */
/* Generated by vdm2spark (Ada SPARK code with C interface) */
extern asn1SccFixedIntList PI_VDM (asn1SccFixedIntList* const inp);
extern asn1SccMySimpleSeq PI_VDM2 (asn1SccT_UInt32* const inp);

/* Prototypes of local functions */
/* Imported by the SDL model orchestrator.pr */
/* The code from the SDL model is senerated by qgen-sdl
   (Ada SPARK code with C interface) */
asn1SccFixedIntList VDM (asn1SccFixedIntList* const inp);
asn1SccMySimpleSeq VDM2 (asn1SccT_UInt32* const inp);

#endif
```

glue.c

```
/* Glue code for binding code generated from SDL, VDM-SL and ASN.1 */

#include "glue.h"

/* Definitions of local functions */
asn1SccFixedIntList VDM (asn1SccFixedIntList* const inp) {
  return PI_VDM(inp);
}

asn1SccMySimpleSeq VDM2 (asn1SccT_UInt32* const inp) {
  return PI_VDM2(inp);
}
```

Simulation and Debugging Support

QGen backend for simulation and debugging

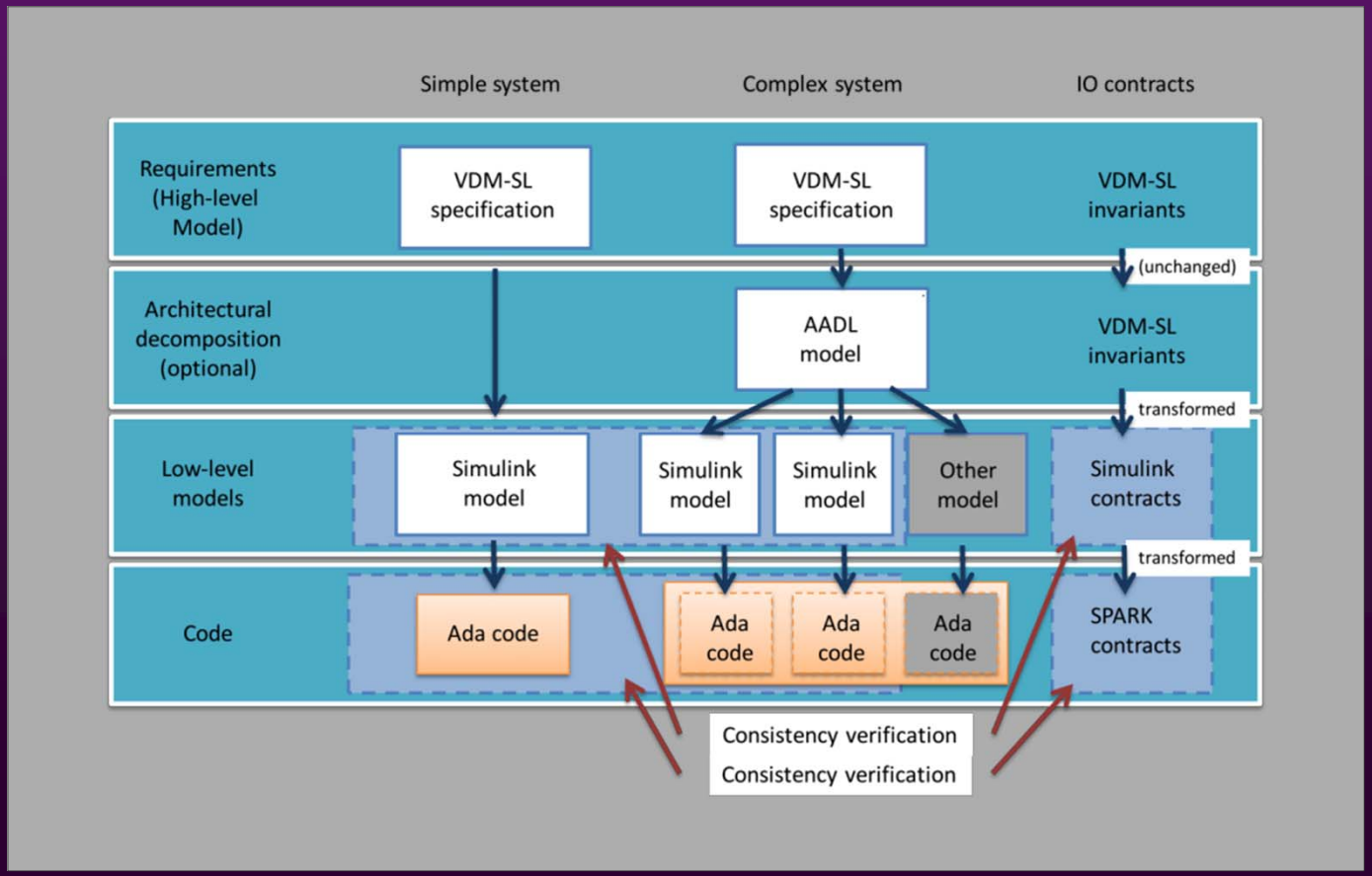
- The scope of this objective was reduced
- It is currently possible to launch QGen via OpenGEODE for code generation from SDL models
- The interface for reading/writing run-time values from OpenGEODE is implemented
- The interface for reading/writing the chart state from OpenGEODE is *not* yet implemented

Design by Contract Workflows

Transformation of VDM-SL contracts for consistency verification (1/3)

- The Design-by-Contract (DbC) is a well-known paradigm and methodology.
- Now emerging also in MBSE - specify and verify contracts at the model level.
- A variant of the DbC is proposed using VDM-SL and Simulink.
  Potentially, also AADL (TASTE).
- Potential use cases:
  - Algorithm design is performed in Simulink. This may be practical for instance in control applications where one can benefit from dedicated blocks such as Integrators, Lookup tables etc.
  - The creation of test data. Simulink is a handy tool for composing and visualizing varions test vectors and comparing the outputs provided by designed subsystem with expected ones.
  - Verification and validation using the Simulink, QGen or SPARK toolsets.
- A more general parallel QGen-related initiative is Software-to-System Integrity. (SSI) – translation and verification of SysML models with formal contracts.

Transformation of VDM-SL contracts for consistency verification (2/3)

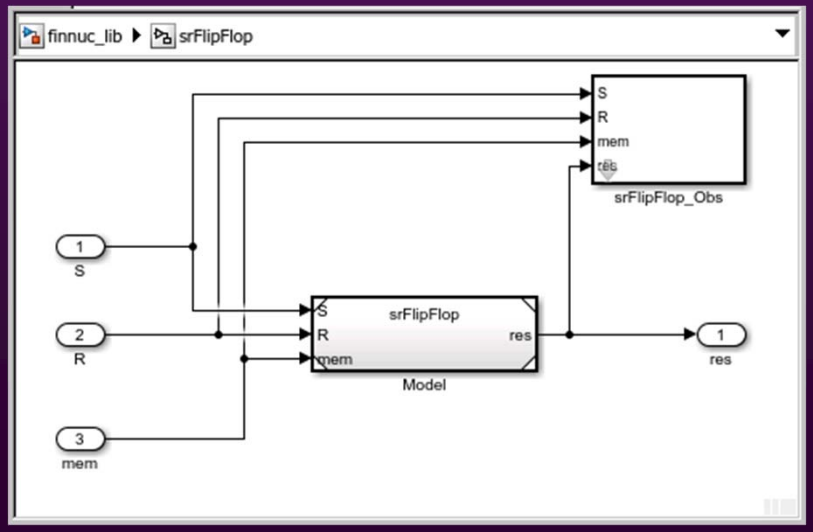


Transformation of VDM-SL contracts for consistency verification (3/3)

- VDM-SL to Ada SPARK
  - Core functionality:
    - VDM-SL preconditions and postconditions would be translated to Ada preconditions and postconditions
    - VDM-SL invariants can be translated to Ada type invariants or assertions *(not yet implemented)*
- Extension to Simulink
  - From a VDM-SL model generate a skeleton of a Simulink model with contracts on IO
  - Contracts are implemented as Ada-based S-function (executable black box) blocks
  - The VDM-SL model can modified and the S-function blocks regenerated
- Possible future extension to AADL (and TASTE)
  - The VDM-SL contracts for the components' IO are embedded as AADL properties in the AADL model
  - Semantic verification of the contracts performed either at Simulink or code level.
  - (Ongoing work in the scope of the QGen Software-to-System Integrity (SSI) initiative)

Expressing contracts in Simulink

- Contracts for Simulink to SPARK conversion
  - Contracts are encapsulated in subsystems (further referred as observer subsystems) and explicitly marked with mask type QGenContract. Such an observer can be attached to any signal in model.



VDM-SL to Simulink Mapping (1/3)

- VDM-SL module
  - Module with name <modulename> shall be converted to Simulink libraries as follows:
    - Specification library named <modulename>_lib. This library always exists
    - Implementation library named <modulename>_impl_lib. This library is generated only when there are any functions or operations with explicit body.
- Types
  - Shall be converted to datatype definitions according to the VDM-SL to Ada SPARK mappings and made available to Simulink as matlab script <modulename>.m

VDM-SL to Simulink Mapping (2/3)

- Functions/operations
  - Each function or operation (sub-program) named <fname> shall be converted to:
    - A SPARK function or procedure <fname> in case the sub-program has an explicit body
    - An S-function block <fname> allowing to invoke the generated SPARK sub-program from Simulink, stored in the Implementation library.
    - A Simulink model <fname> containing
      - Input ports corresponding to the sub-program input arguments,
      - Output port corresponding to the return value
      - Output ports corresponding to output arguments (if any)
    - Reference block pointing to the <fname> S-function in the implementation library
  - Subsystem <fname> in the specification library containing
    - Input ports corresponding to the sub-program input arguments,
    - Output port corresponding to the return value
    - Output ports corresponding to output arguments (if any)
    - ModelReference block pointing to the <fname> model
    - Subsystems corresponding to pre-and postconditions

VDM-SL to Simulink Mapping (3/3)

- Pre- and postconditions
  - Each pre- and postcondition in VDM shall be converted to an observer block in the Specification library
    - The name of the observer shall be <fname>_obs[_pre|_post]
    - Type of the observer shall be defined by mask type either QGenPrecondition or QGen_Postcondition
    - The original VDM text shall be stored in mask parameter QGenContractVDM
    - Generated SPARK code shall be stored in mask parameter QGenContractSPARK
    - Subsystem shall have an input port for each variable the contract used
    - It shall contain an S-function executing the generated SPARK code and returning true/false to an Assertion block.
  - In case a function does not have postcondition, but has explicit definition the explicit definition shall be used for postcondition. The simulink structured shall be generated as above, except that
    - The executed S-function is the one generated in the implementation library
    - Output of the S-function shall be compared with output of the subsystem where postcondition is applied and this comparison attached to the Assert block

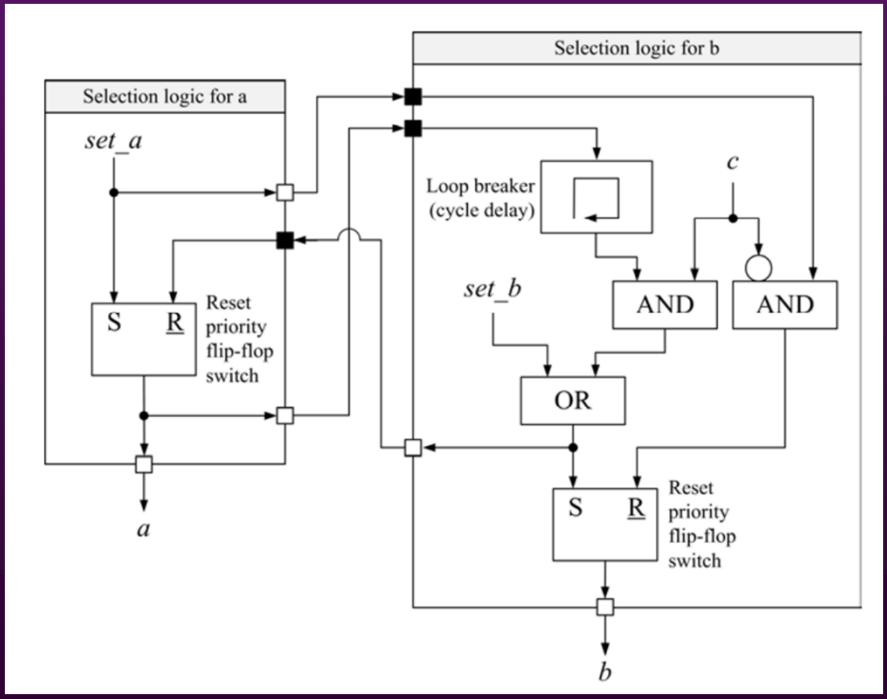
Case Study: Mutually Exclusive State Switches

- We use a simple model from [1] implementing two subsystems with mutually exclusive states.
- The paper presents number of contracts on the specification and shows, that the given design violates one of the requirements.
- Here we
  - model the same specification in VDM-SL
  - perform simulation in Simulink demonstrating the violation of the contract
  - and also prove the violation using GNATProve

[1] Pakonen, A; Tahvonen, T; Hartikainen, M; Pihlanko, M, “Practical applications of model checking in the Finnish nuclear industry”, Proceedings of the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies, NPIC & HMIT 2017, 11 - 15 June, 2017, San Francisco, CA, USA.

Case Study: Mutually Exclusive State Switches

- The original model from [1]



[1] Pakonen, A; Tahvonen, T; Hartikainen, M; Pihlanko, M, “Practical applications of model checking in the Finnish nuclear industry”, Proceedings of the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies, NPIC & HMIT 2017, 11 - 15 June, 2017, San Francisco, CA, USA.

Case Study: Mutually Exclusive State Switches

r1: “The set_a command shall lead to a, if b is not already selected”,
or as a formal LTL property p1: $G((set\_a \wedge \neg b) \rightarrow a)$.

r2: “The set_a command shall reset b, if c is not active”,
or p2: $G((set\_a \wedge \neg c) \rightarrow b)$.

r3: “The set_b command shall lead to b and reset a, if set_a is not active”,
or p3: $G((set\_b \wedge \neg set\_a) \rightarrow (b \wedge \neg a))$.

r4: “If a has been selected, the signal c shall change the selection to b”,
or p4: $G((Ya \wedge c) \rightarrow b)$.

r5: “Only one mode (a or b) shall be active at the same time”,
or p5: $G(\neg(a \wedge b))$

[1] Pakonen, A; Tahvonen, T; Hartikainen, M; Pihlanko, M, “Practical applications of model checking in the Finnish nuclear industry”, Proceedings of the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies, NPIC & HMIT 2017, 11 - 15 June, 2017, San Francisco, CA, USA.

Case Study: Mutually Exclusive State Switches

- VDM-SL specification for the state and types

```
state SetABState of
  --  memory of the flip-flop block in
  --  selection logic of output A
  selAFlipFlop : FFMem
  --  memory of the flip-flop block in
  --  selection logic of output B
  selBFlipFlop : FFMem
  --  memory of the loop breaker in
  --  selection logic of output B
  selBMem      : bool

  --  make sure the outputs are in opposite states
  --  initially
  init s == s = mk_SetABState (
    mk_FFMem (true, false),
    mk_FFMem (false, true),
    true)
end
```

```
types
  --  datatype for fliplop output and memory
  FFMem::   q          : bool
            qCompl     : bool;

  --  datatypes outputs
  --  (purely technical to name tuple elements)
  SelBRes ::  setB  : bool
              b     : bool;
  SelABRes :: a : bool
              b     : bool;
values
```

Case Study: Mutually Exclusive State Switches

- VDM-SL specification for the elementary functions and operations

```
functions

  -- srFlipFlop returns the new value of
  -- flipflop memory based on Set, Reset and
  -- previous outputs
  srFlipFlop : bool * bool * FFMem -> FFMem
  srFlipFlop (S, R, mem) ==
   if S then
    (if not R then mk_FFMem (true, false)
     else mk_FFMem (false, false))
   else
    (if not R then mem
     else mk_FFMem (false, true));
```

```
operations

  -- procedure implementing selection logic for A
  selectA : bool * bool ==> bool
  selectA (setA, resetA) ==
   (
    selAFlipFlop := srFlipFlop (setA, resetA, selAFlipFlop);
    return selAFlipFlop.q;
   );

   -- procedure implementing selection logic for B
   selectB : bool * bool * bool ==> SelBRes
   selectB (setB, resetB, C) ==
    (
      let s = setB or (selBMem and C) in
       (selBFlipFlop := srFlipFlop
         (s, not C and resetB, selBFlipFlop);
        return mk_SelBRes(s, selBFlipFlop.q);
       )
    );
```

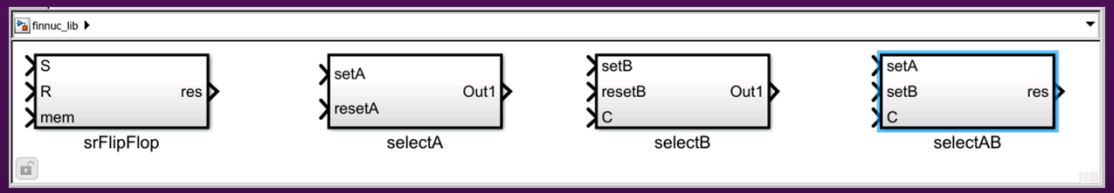
Case Study: Mutually Exclusive State Switches

- VDM-SL specification for the integral logic

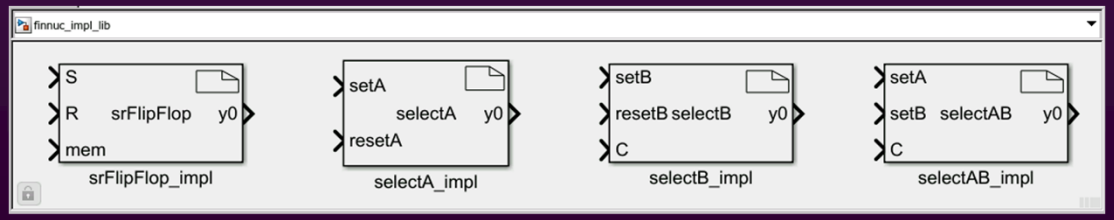
```
-- procedure implementig the integral logic
selectAB : bool * bool * bool ==> SelABRes
selectAB (setA, setB, C) ==
 (
   let selB = selectB (setB, setA, C) in
   (
    selBMem := selectA (setA, selB.setB);
    return mk_SelABRes (selBMem, selB.b);
   )
 )
 post
 (
   --  R1
   (setA and not RESULT.b => RESULT.a) and
   --  R2
   (setA and not C => not RESULT.b) and
   --  R3
   ((setB and not setA) => (RESULT.b and not RESULT.a)) and
   --  R4
   (selBMem~ and C => RESULT.b) and
   --  R5
   (not (RESULT.a and RESULT.b))
 );
```

Case Study: Mutually Exclusive State Switches

- Simulink model
  - Specification library

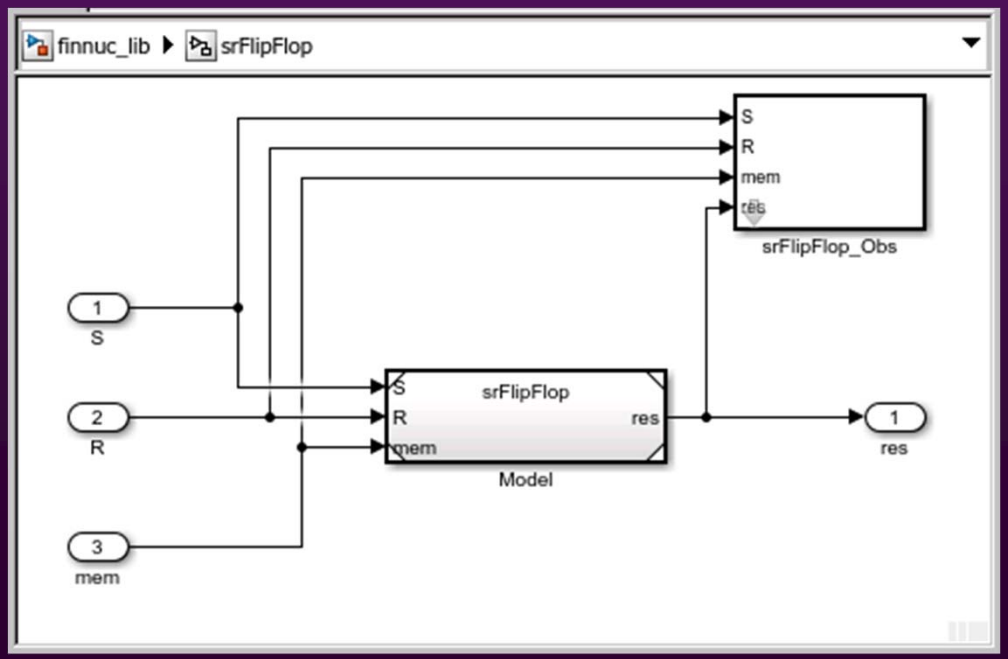


  - Implementation library



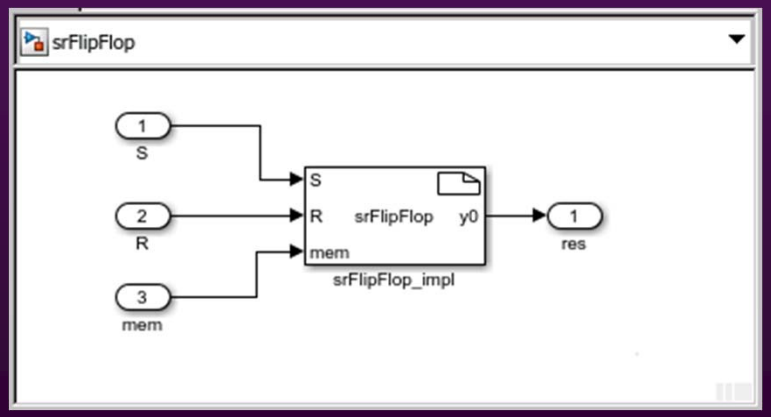
Case Study: Mutually Exclusive State Switches

- Simulink model
  - Contents of the generated srFlipFlop block in the specification library



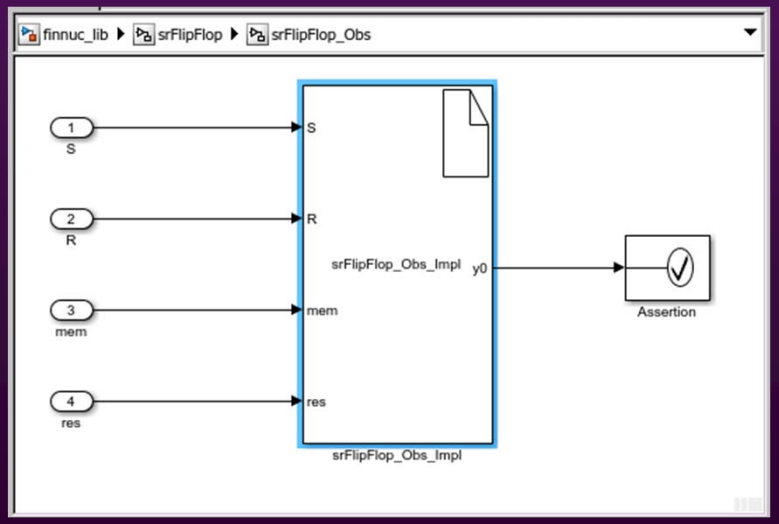
Case Study: Mutually Exclusive State Switches

- Simulink model
  - Implementation of the srFlipFlop function



Case Study: Mutually Exclusive State Switches

- Simulink model
  - The observer for srFlipFlop block



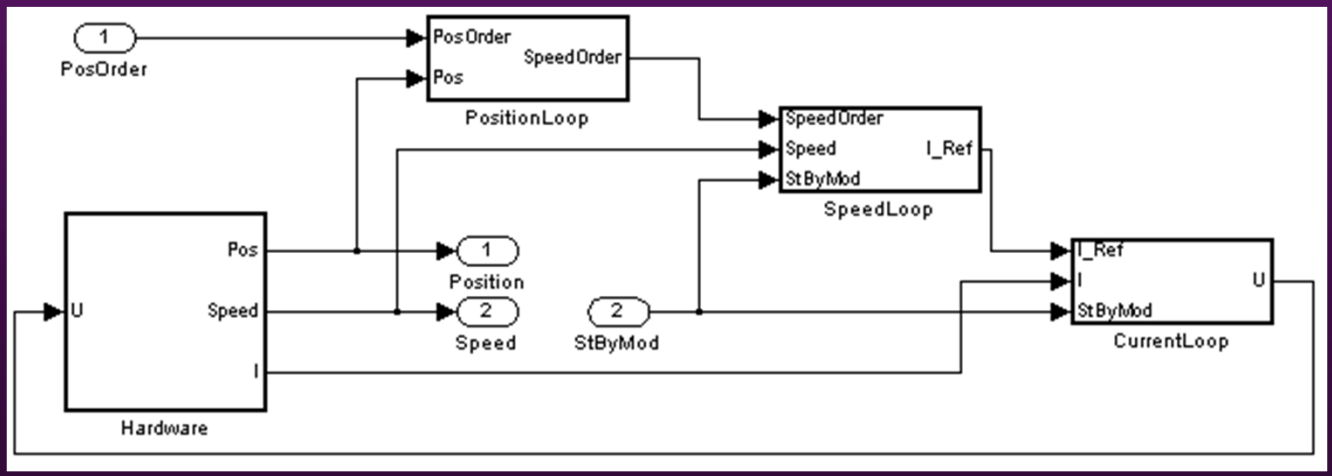
TASTE/AADL and Simulink Round-Trip

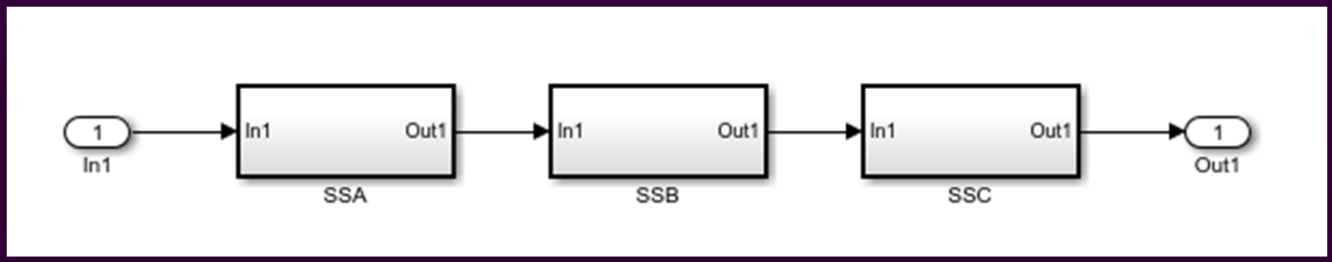
TASTE/AADL and Simulink Round-Trip

- Proposal to improve the mapping of multi-component Simulink models to TASTE/AADL
- Potentially useful when:
  a. The original model comes from a system engineer who already has divided the controller part of an application into different sub-systems and verified the functional behaviour.
     - In this case the top-level diagram could be converted from Simulink to AADL
  b. Architecture is already modelled as an AADL model, however, verification and/or validation would benefit from the modelling tools in Simulink.
     - In this case the AADL model could be converted to a Simulink diagram, where each component is either a subsystem (for the components with matched Simulink models) or an S-function (for components with generated code).
- Improved language mappings between AADL and Simulink have been defined. Tool support currently out of scope.

TASTE/AADL and Simulink Round-Trip

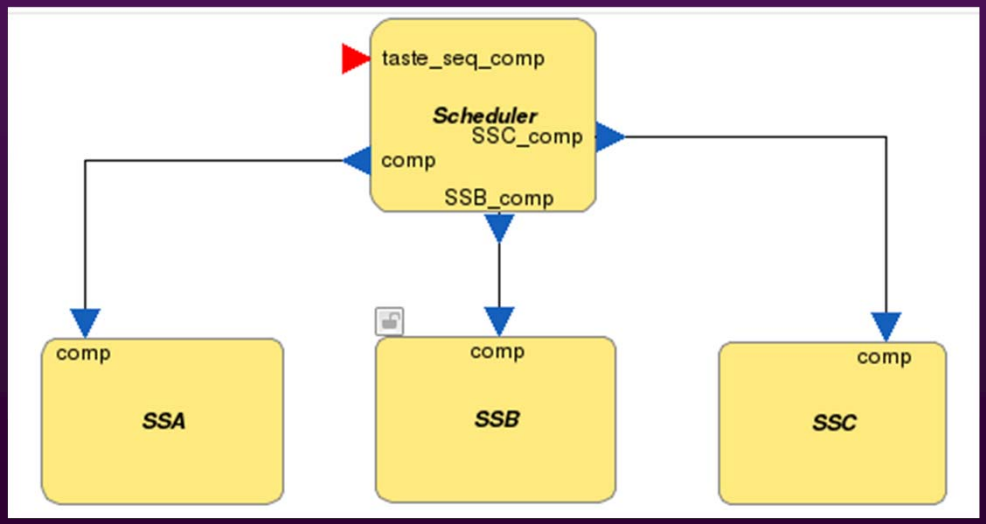
- Motivation
  - Multi-component (and possibly also multi-rate) Simulink models





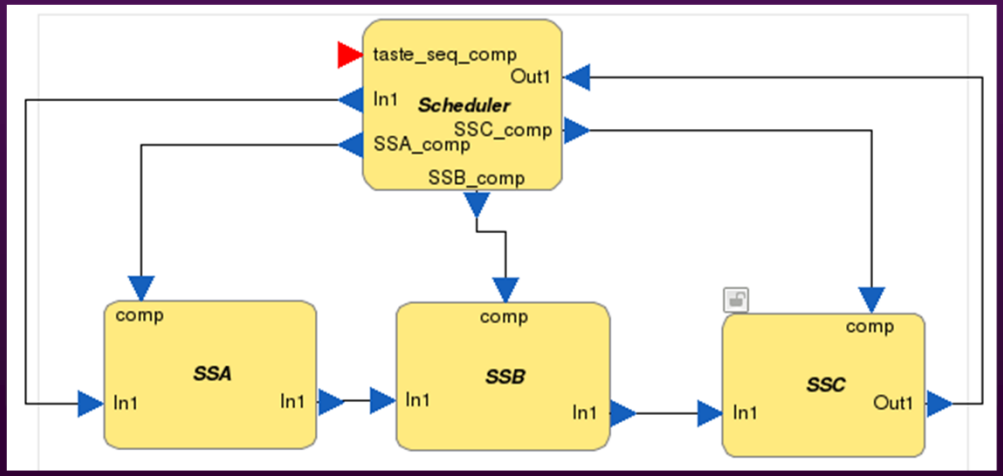
TASTE/AADL and Simulink Round-Trip

- Option 1: Data flow and control flow through a central scheduler
  - This is the current approach for integrating subsystems imported from Simulink
  - Data flow and dependencies between the subsystems are embedded in the scheduler's code



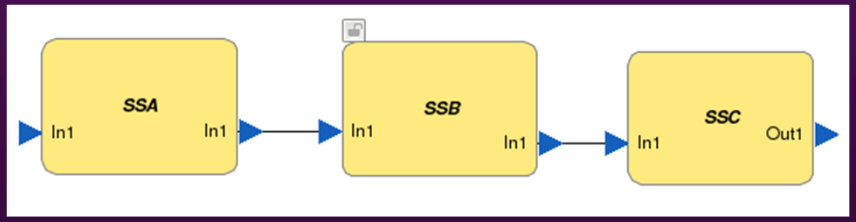
TASTE/AADL and Simulink Round-Trip

- Option 2: Explicit data flow, control flow in a central scheduler
  - Additional sporadic interfaces for passing data.
  - Intermediate data buffers are required.



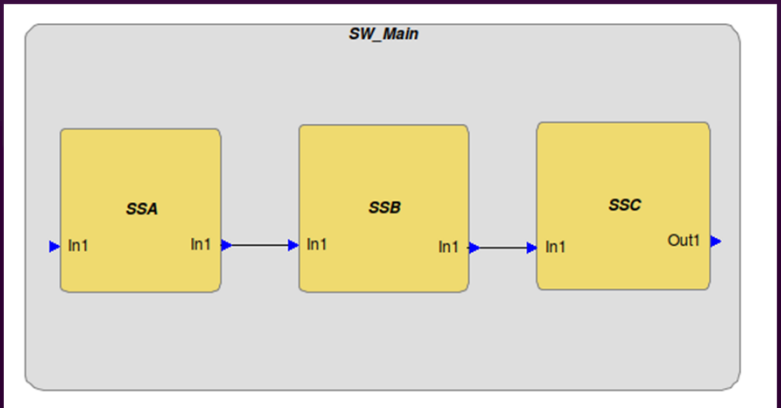
TASTE/AADL and Simulink Round-Trip

- Option 3: A set of synchronous functions
  - Truly synchronous semantics similar to Simulink
  - Simple and intuitive. However, intermediate data and event buffers are required



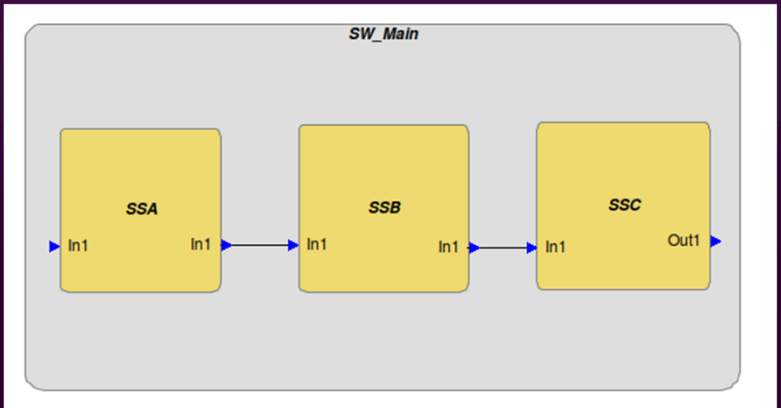
TASTE/AADL and Simulink Round-Trip

- Option 4: A set of synchronous functions with dedicated container (1/2)
  - The functions corresponding to individual Simulink subsystems are placed in a specific container and this container is marked to have synchronous execution model
  - Allows one to present the data flow explicitly in the model without the need for composing the scheduler manually
  - Changes required in the semantics of interface view elements:
    - a new type for provided and required interfaces denoting that no code shall be generated for the interface
    - container markup to define the synchronously executed cluster



TASTE/AADL and Simulink Round-Trip

- Option 4: A set of synchronous functions with dedicated container (2/2)
  - Proposed code generation workflow (*Not yet implemented*)
    - The AADL model is parsed using Ocarina. The converter picks up functions where the source language is QGen_Ada or QGen_C and creates QGen SystemBlock objects for each of them
    - QGen *system model* data structure is created using the QGen metamodel API
    - The same API is used for converting the block diagram to a Simulink diagram
    - TASTE buildsupport invokes QGen on the top-level Simulink subsystem. All contained functions in the AADL model are contained in this diagram and processed by QGen

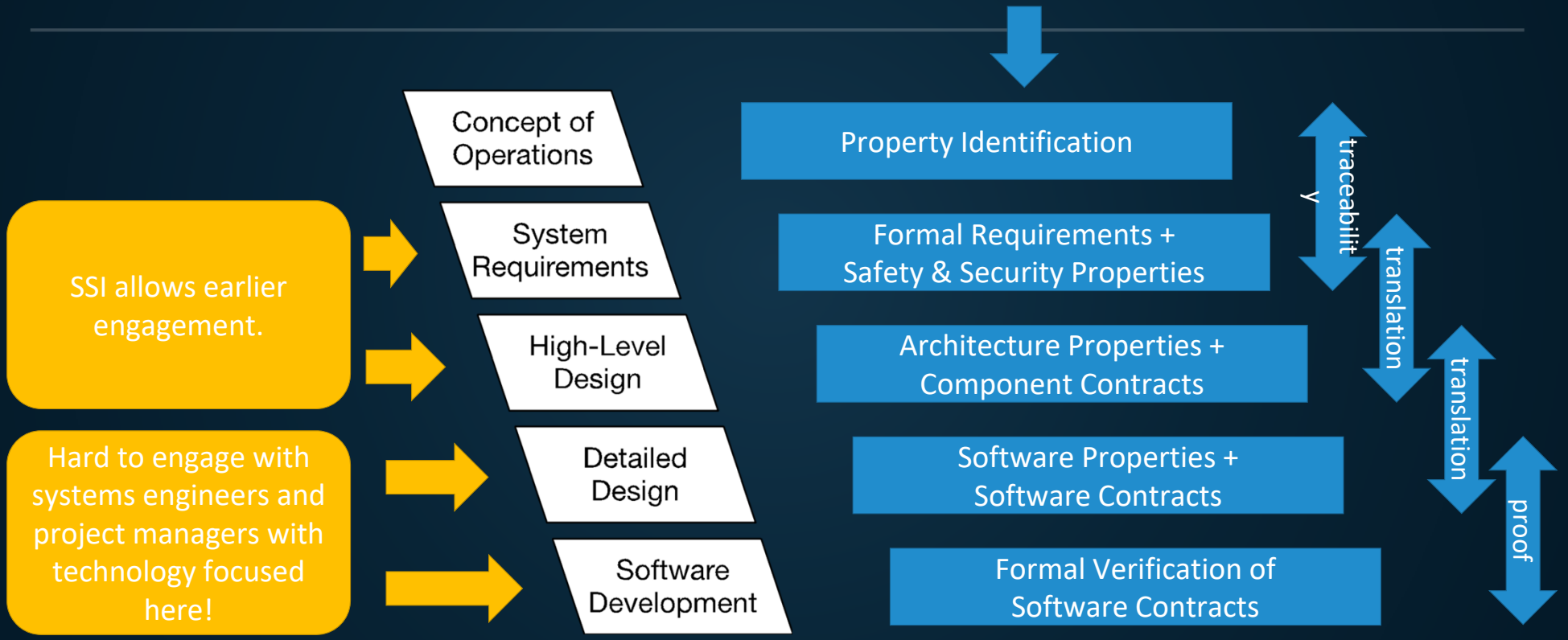


QGen / SSI Roadmap

QGen Roadmap 2019-2020

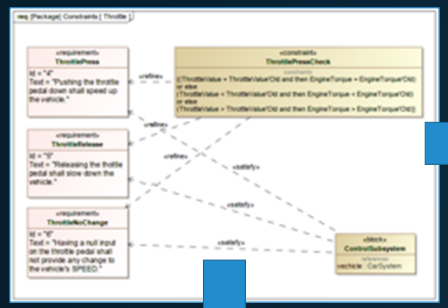
- Support for upcoming Simulink versions
- Improved support for modular and incremental code generation
- Enhanced Code Performance and Quality
- Enhanced CodePeer and SPARK integration
- Enhanced QGen Build Performance
- Improved Ada S-Function integration in Simulink
- Completion of QGen TQL1 qualification

SSI


SSI allows earlier engagement.
Hard to engage with systems engineers and project managers with technology focused here!
Concept of Operations
System Requirements
High-Level Design
Detailed Design
Software Development
Property Identification
Formal Requirements + Safety & Security Properties
Architecture Properties + Component Contracts
Software Properties + Software Contracts
Formal Verification of Software Contracts
traceability
translation
translation
proof

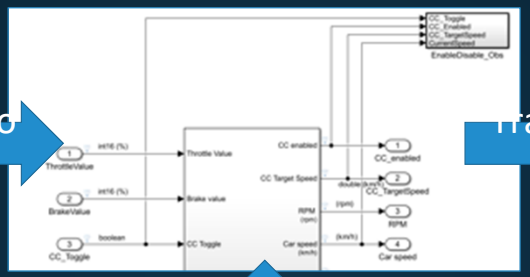
SSI

SysML Requirements Diagram



Translation

Simulink Synchronous Observer



Translation

SPARK

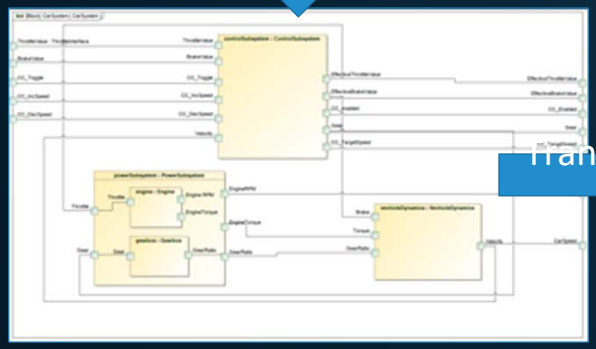
```
package car_system is

   procedure initStates (State : in out car_system_State);

   procedure initOutputs (State : in out car_system_State);

   procedure comp
     (Throttle_Value : Integer_16_m0_0_M100_0;
      Brake_value : Integer_16_m0_0_M100_0;
      Cruise_Control : Boolean;
      Acc_Speed : Boolean;
      Dec_Speed : Boolean;
      CC_enabled : out Boolean;
      CC_Target_Speed : out Long_Float_m30_0_M100_0;
      RPM : out Long_Float;
      Car_speed : out Long_Float;
      Gear : out Integer_8;
      Throttle_input : out Integer_16_m0_0_M100_0;
      Brake_input : out Integer_16_m0_0_M100_0;
      State : in out car_system_State);

   procedure up (State : in out car_system_State);
```

Manual Refinement

QGen Verifier

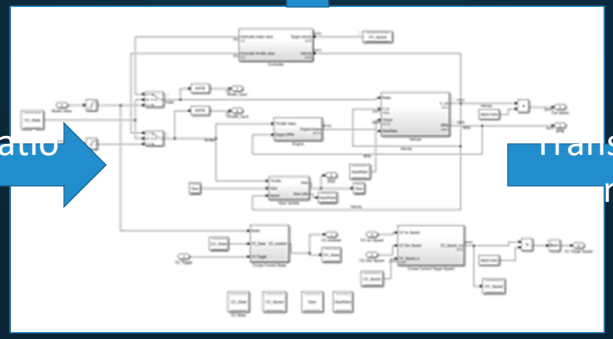
SPARK Tools

SysML Internal Block Diagram



Translation

Simulink Subsystem



Translation

SPARK Code

```
package body car_system is

   procedure initStates (State : in out car_system_State) is
   begin
      --  Start of function
      --  Block 'car_system/Car System/Physical model/Vehicle/Discrete-Time Integrator'
      State.Discrete_Time_Integrator_in_memory := 0.0E+00;
      State.Discrete_Time_Integrator_out_memory := (qgen_base_workspace.V_init) / (3.6);
      --  End Block 'car_system/Car System/Physical model/Vehicle/Discrete-Time Integrator'

      --  Block 'car_system/Car System/Controller/If1'
      Controlled_input.initStates (State.Controlled_input_memory);
      --  End Block 'car_system/Car System/Controller/If1'

      --  Block 'car_system/Car System/Controller/Merge throttle'
      State.Merge_throttle_out1 := 0;
      --  End Block 'car_system/Car System/Controller/Merge throttle'

      --  Block 'car_system/Car System/Controller/Merge brake'
      State.Merge_brake_out1 := 0;
      --  End Block 'car_system/Car System/Controller/Merge brake'

      --  Block 'car_system/Car System/Cruise Control Target Speed/Merge'
      State.Merge_out1 := 0.0;
      --  End Block 'car_system/Car System/Cruise Control Target Speed/Merge'

      --  Block 'car_system/Car System/CC State'
      State.CC_State := False;
```

Tools availability and licensing

- QGen
  - Available from AdaCore as a commercial open-source product
    - Please visit http://www.adacore.com/qgen
  - Evaluation version will be integrated into the TASTE VM
  - GPL license
- QGen SDL frontend
  - Available from IB Krates or AdaCore
  - and also will be integrated into the TASTE VM
  - EPL license
- QGen VDM-SL frontend
  - Relies of the VDM importer from Overture - GPL license and
  - QGen/Gene-Auto API that is based on the Eclipse Modeling Framework (EMF) - EPL license
  - EPL licensed code and GPL licensed code may only be combined under certain conditions.
  - Negotiations on modifying the Overture license are ongoing with the Overture community. Tool availability is currently pending.

AdaCore

Thank you!

Contacts:

www.krates.ee	**info@krates.ee**
www.adacore.com / qgen	**sales@adacore.com**