Joint Estonian-Latvian Computer Science Theory Days

30.09 – 03.10.2010 / Rakari, Latvia

Between Qualification and Certification:

Specifying and Verifying Model Transformations in an Embedded Code Generator

Andres Toom[1,2]

Joint work with Marc Pantel[2]

1. Institute of Cybernetics at Tallinn University of Technology, Estonia
2. IRIT-ENSEEIHT, Université de Toulouse, France

Outline

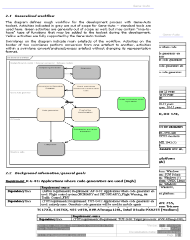
- Model Driven Engineering (MDE)
- Tool Qualification
  - DO-178B(C)
- Gene-Auto Embedded Code Generator
  - Proven Development – Certified Elementary Tool Development With The Coq Proof Assistant
    - N. Izerrouken, M. Pantel, X. Thirioux (IRIT)
  - Structural Model Transformation Specification and Validation with MOF and OCL
    - Joint work with M. Pantel (IRIT)

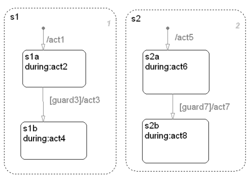
MDE

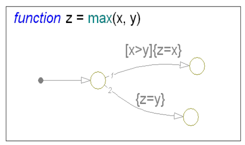
Model Driven Engineering (MDE)

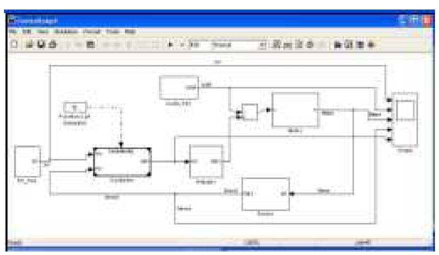
- MDE promotes using models in each phase of software development
- Main idea:
  - The abstract platform independent model (PIM) is transformed to the target specific model (code) through a series of successive refinements
- Well known examples
  - Unified Modeling Language (UML) - OMG
  - Meta-Object Facility (MOF) - OMG
  - Eclipse Modeling Framework (EMF)
    - Ecore, EMOF

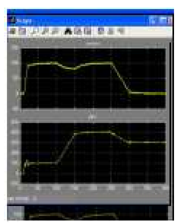
Software Development V-Cycle

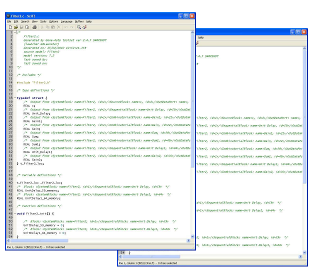


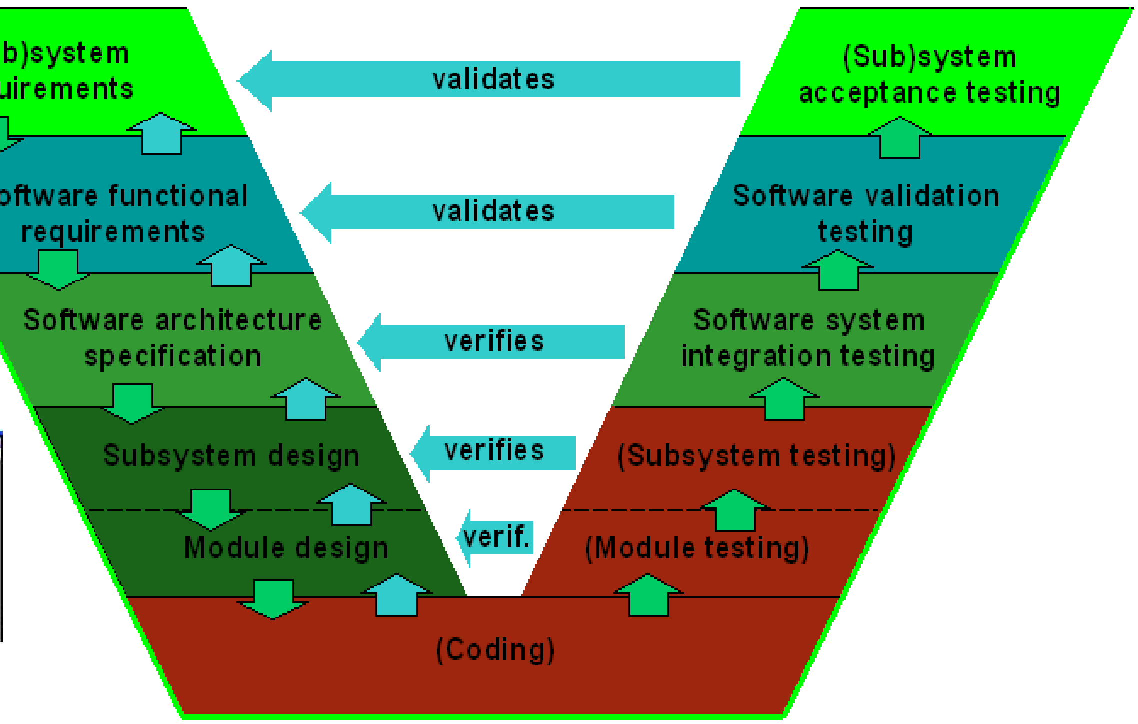












Some challenges

- Functionality and complexity increase of embedded systems
- Increased safety-criticality, high-integrity
- Required software longevity (maintainable up to 80 years)
- Product quality and certification (DO-178/ED-12, ISO 26262, …)
- Need to reduce development cycles and prototype loops to achieve cost-efficiency

Tool Qualification

DO-178

- DO-178B(C)/ED-12B(C) - Software Considerations In Airborne Systems And Equipment Certification
- Main software related civil avionics standard in the US, Europe and many other countries
- Has motivated also standards in other domains, e.g. the new automotive ISO 26262 standard

DO-178 (contd)

- Short history of the DO-178/ED-12
  - 1982 DO-178/ED-12
  - 1985 DO-178A/ED-12A
  - 1992 DO-178B/ED-12B
  - 2010? DO-178C/ED-12C
    - SG4: Model Based Design and Verification
    - SG5: Object-Oriented Technology
    - SG6: Formal Methods

DO-178 (contd)

- DO-178/ED-12 objectives vs. level of criticality

Level	Failure condition	Objectives	With independen
A	Catastrophic	66	25
B	Hazardous	65	14
C	Major	57	2
D	Minor	28	2
E	No effect	0	0

DO178 and testing

- Test coverage criteria for code structure
  - Level C - statement coverage
  - Level B - decision coverage
  - Level A - modified condition/decision coverage (MC/DC)
- Example

```
if a and (b or c) then
   x = y;
```

  - Statement coverage - 1 test needed
  - Decision coverage - 2 tests needed
  - Modified condition/decision coverage (MC/DC) - 4 tests needed

DO178B(C) and Formal Methods

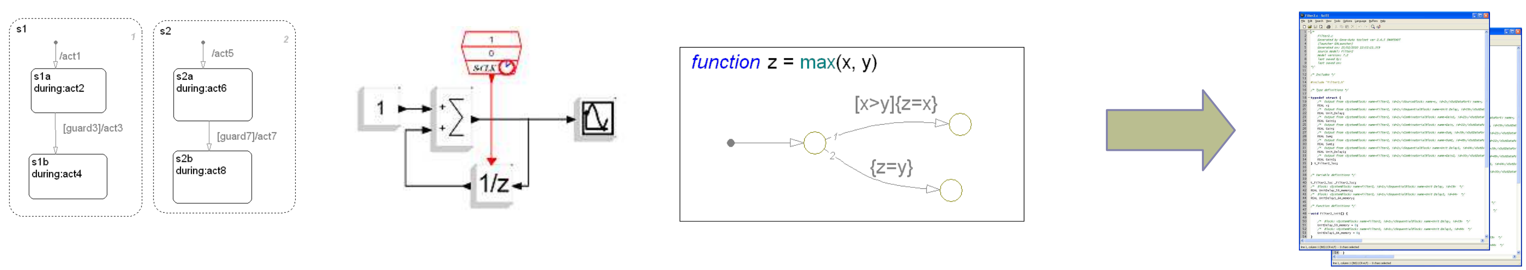
- DO178B states
  - A formal method can be used as an alternative method
  - An alternative method should be shown to satisfy the objectives of this document
- DO178C
  - Does not relax this basic requirement, but provides more guidelines to using formal methods in a qualified context

Gene-Auto

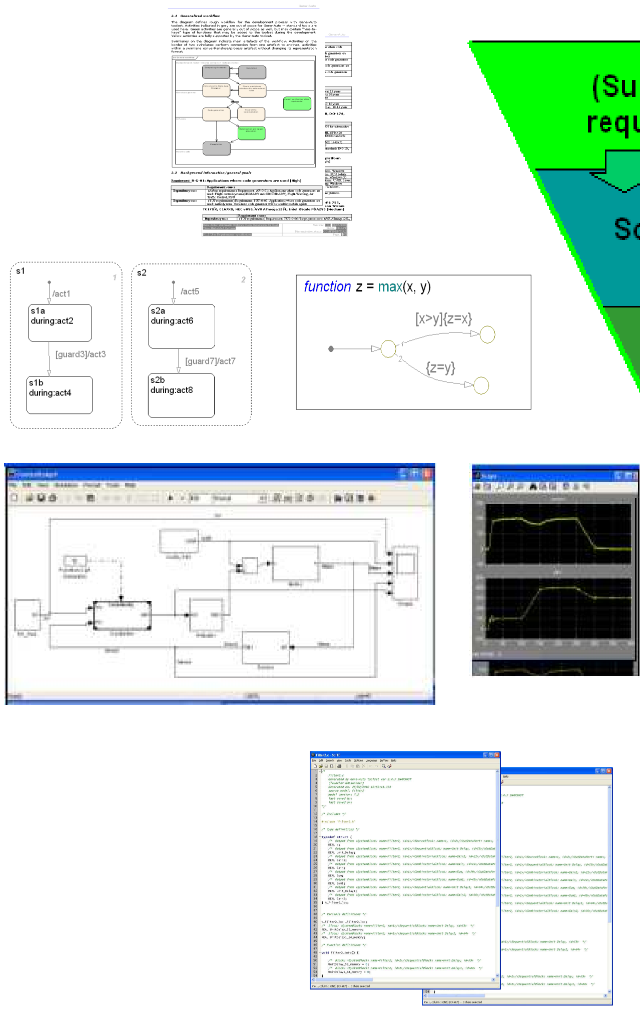
Embedded Code Generator

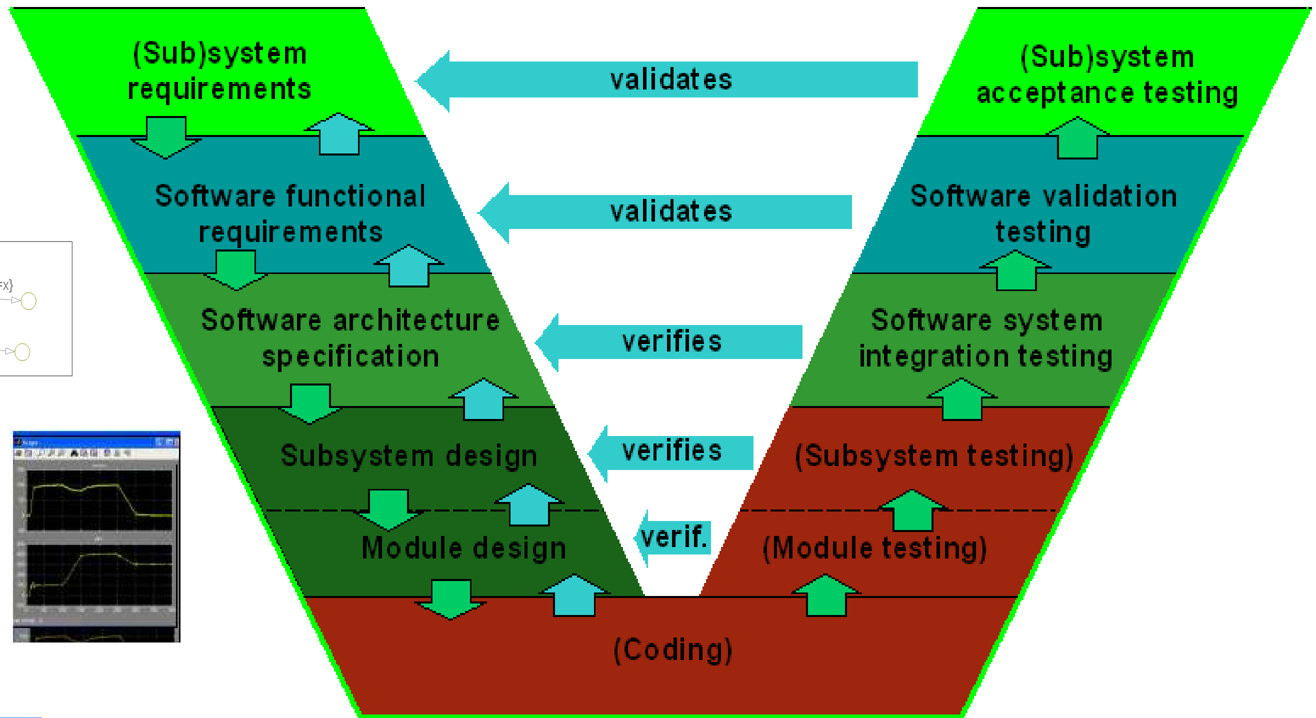
Main objectives

- Translation of functional specifications of application software from high-level graphical modelling formalisms (Simulink/Stateflow/Scicos) to imperative sequential code
  - C, Ada
- DO-178B(C)/ED-12B(C) qualifiable toolset for safety-critical embedded systems
- Open source and customisable architecture
- Evaluate formal methods based code generator qualification vs. traditional testing based approach



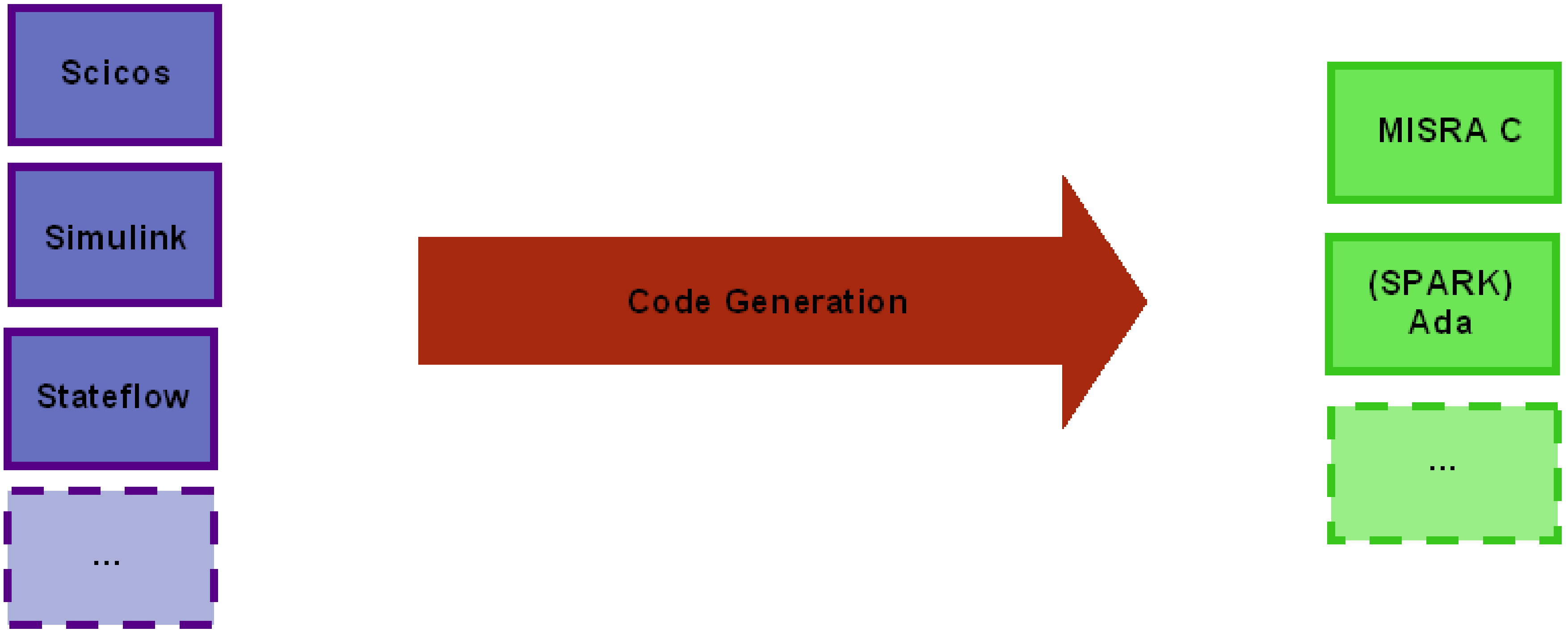
Main scope



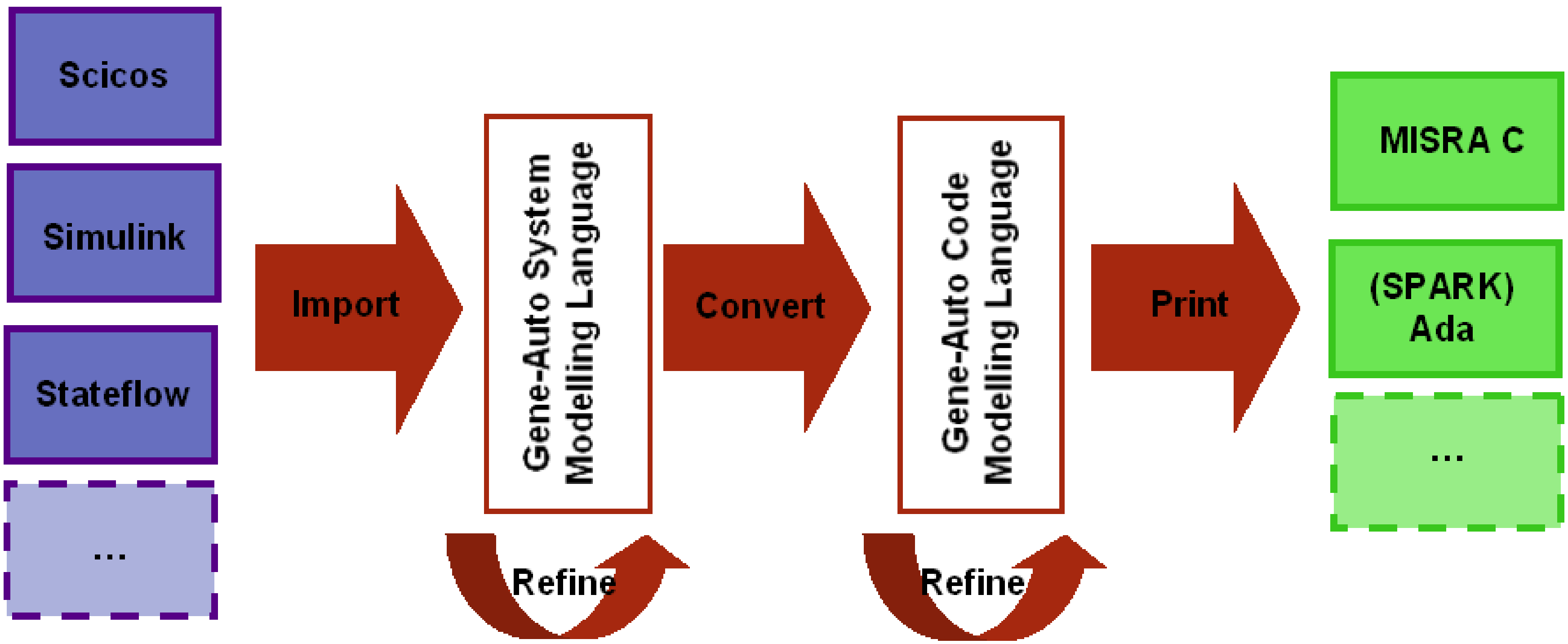


GENE-AUTO

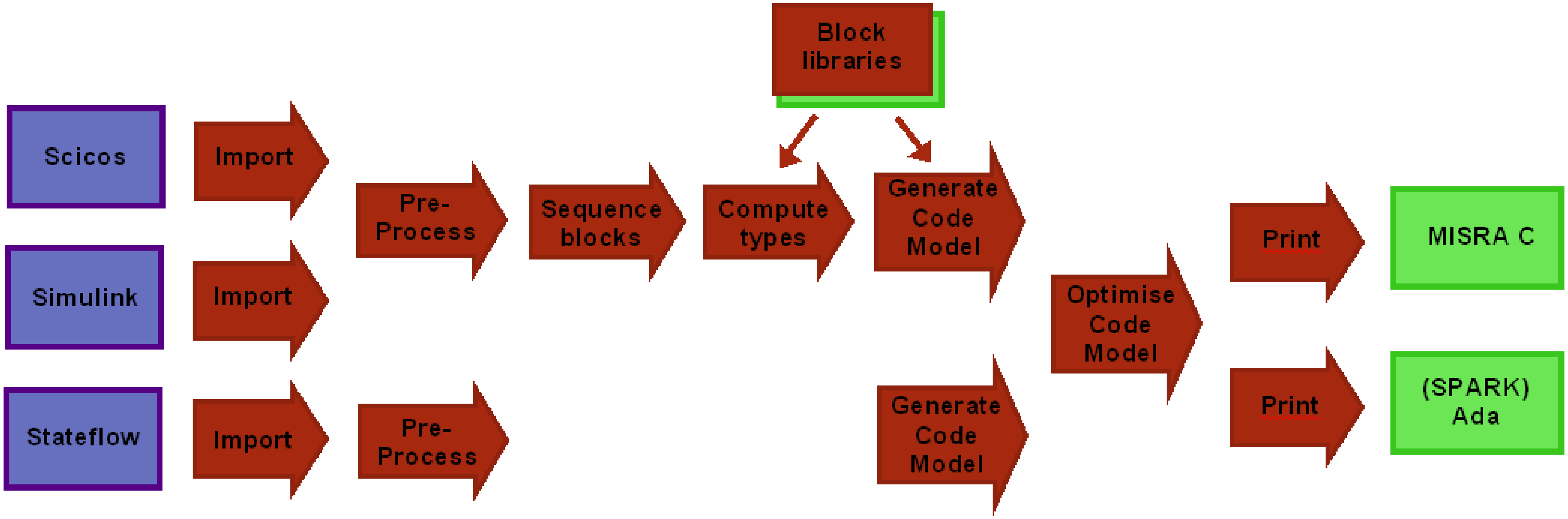
Toolset architecture and approach



Toolset architecture and approach (contd)



Toolset architecture and approach (contd)



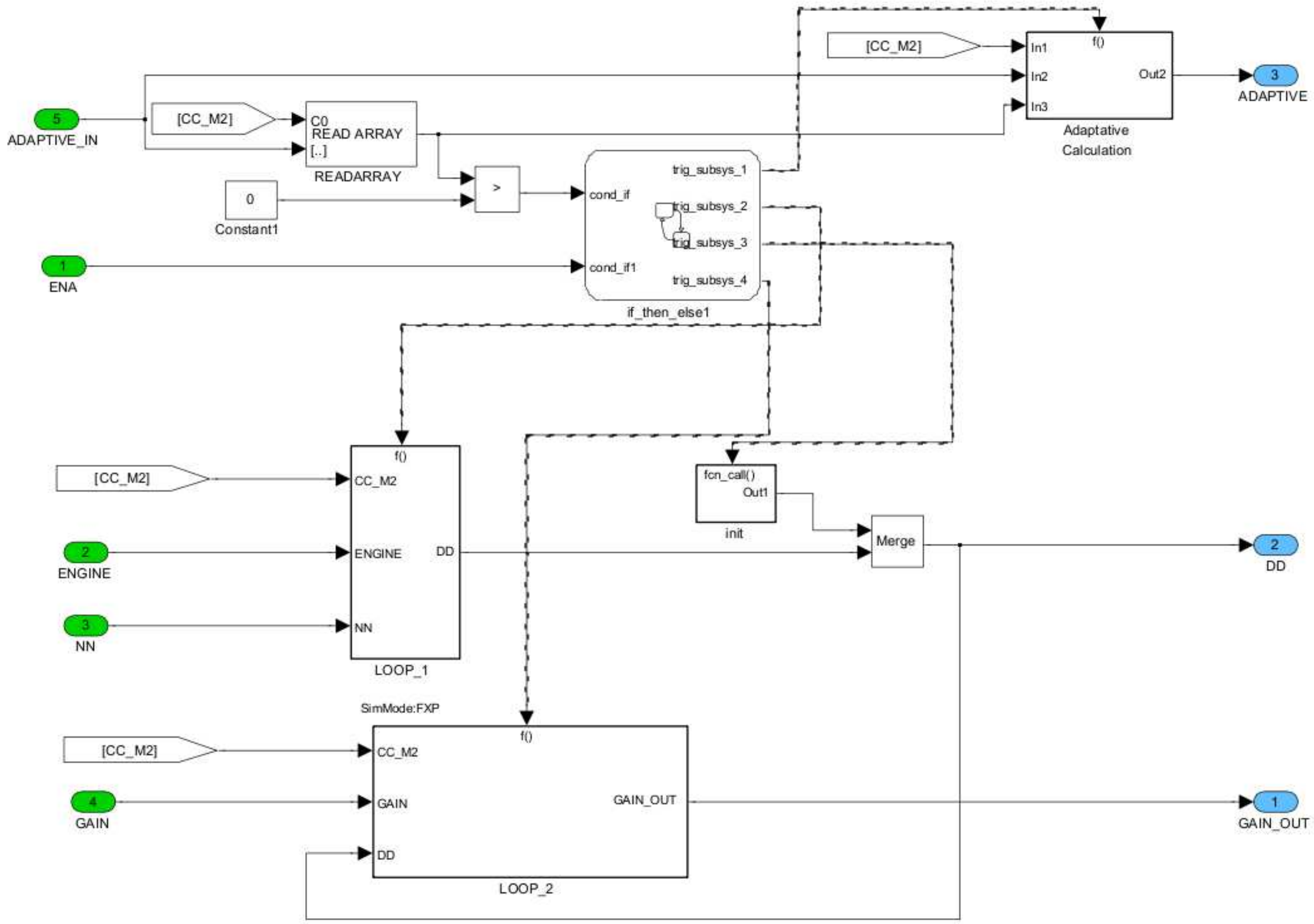
Proven Development – Certified Elementary Tool Development With The Coq Proof Assistant

N. Izerrouken, M. Pantel, X. Thirioux (IRIT)

Proven development with Coq

- In the early phase of Gene-Auto a comparative study of several formal techniques was conducted. Including:
  - Proven development, certified development (B-method), proof carrying code, translation validation, model checking, static analysis
- Proven development with a proof assistant and program extraction was identified as the most appropriate technique currently available for this task
- Initial case study
  - Sequencing of Simulink block diagrams

Sequencing of Simulink block diagrams



Fragment of an automotive powertrain controller (Continental)

Coq

- An interactive theorem prover (proof assistant)
- Capable of extracting a certified program from the constructive proof of its formal specification
- Considered to have nearly industrial strength
- Not DO-178 qualified
- Part of the Coq kernel formally verified. Verification of a larger subset and other components (e.g. extractor) ongoing
- Has been used to program and prove the correctness of a compiler from a large subset of the C programming language to PowerPC assembly code
  - X. Leroy, Formal verification of a realistic compiler (CompCert)

Summary of the proven development experiment

- Block sequencer tool (Coq + OCaml extraction)
  - ca 4500 LOC of specification and proofs
  - ca 130 proved theorems
  - Izerrouken, N., Pantel, M., Thirioux, X., Machine-Checked Sequencer for Critical Embedded Code Generator, ICFEM'09

- Block diagram typer (Coq + OCaml extraction)
  - Prototype. Similar approach as above
  - Type checking (limited subset of blocks)
  - Type inference in the forward and backward direction (in progress)

Summary of the proven development (contd)

- The technology is quite difficult for common software engineers
- It is quite difficult to give an accurate estimation of the development time based on the early user requirements
- Qualified Coq proof checker and program extractor are currently only available preliminary academic prototypes
- Development of scalable algorithms is not trivial

Structural Model Transformation Specification and Validation with MOF and OCL

Joint work with M. Pantel (IRIT)

Domain Specific Model Transformations

- Very application and user specific
- Often quite simple structural refinements
- Need to be easily maintainable and customisable

Domain Specific Model Transformations (contd)

- Gene-Auto Functional Model Preprocessor
  - Input of the tool is a raw imported GASystemModel
  - Output is refined GASystemModel
  - The tool performs following main transformations (model refinements):
    - Masked Subsystems that have corresponding library Blocks are replaced by library Blocks
    - Stateflow Subsystems are replaced with Stateflow Blocks and their port datatypes are determined
    - All remaining elementary Blocks in the input model are matched with corresponding library Blocks
    - Virtual Subsystems are flattened
    - Execution priorities of concurrent Blocks are computed based on their graphical position

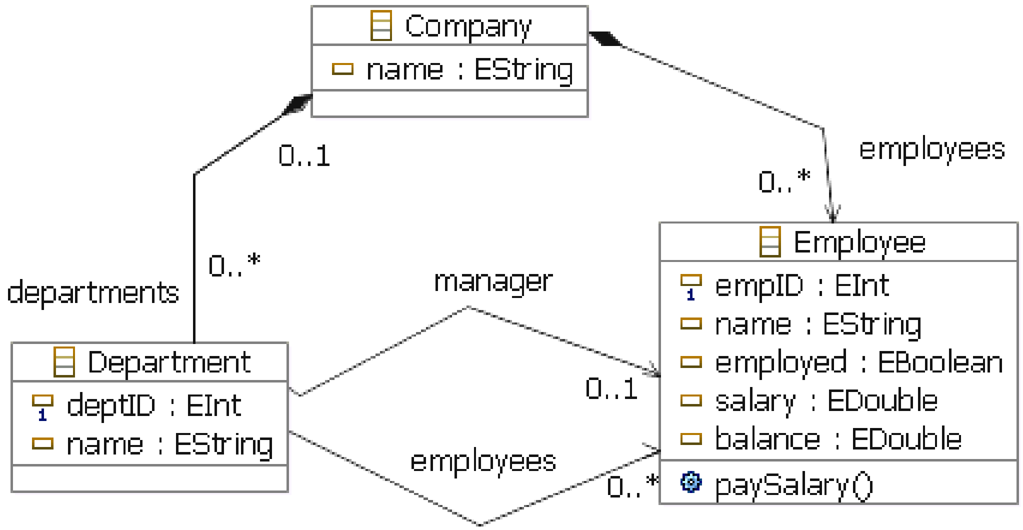
Translation Validation

- 1998 A. Pnueli et al.
- *A posteriori* verification of individual transformation instances
- Transformation is followed by a verification phase
- The transformation engine can give additional hints to the verifier

MOF and OCL

- Meta-Object Facility (MOF) - *OMG standard*
  - Four-layered meta-modeling architecture (M0..M3)
  - Most prominent example: Unified Modeling Language (UML) meta-model (M2 model)
- Object Constraints Language (OCL) - *OMG standard*
  - A declarative constraint and object query language.
  - Initially conceived as a formal specification language extension to the UML
  - Can now be used with any MOF meta-model

MOF - OCL Example
Company (Meta)Model



MOF - OCL Example (contd)

```
-- Employee class in the meta-model
class Employee
   empID       : EInt
   name        : EString
   employed    : EBoolean
   salary      : EDouble
   balance     : Edouble

-- OCL constraints
context Employee
   inv  : empID > 0
   inv  : employed implies salary > 0
context Employee::paySalary()
   pre         : employed and salary > 0
   post        : balance = balance@pre + salary
```

Model Transformation Constraints with MOF - OCL

```
-- Source
context GASystemModel
inv: getAllElements()->isUnique(id)
inv: …

-- Target
context GASystemModel
inv: getAllElements()->isUnique(id)
inv: getAllElements()
        ->select(oclIsKindOf(SystemBlock)
               and isVirtual)
           ->isEmpty()
inv: …
```

Model Transformation Constraints (contd)

- Need to model the *whole* transformation

```
-- System-to-System transformation
class SM2SM
   src : GASystemModel
   tgt : GASystemModel
```

Example: Gene-Auto Tool Requirement

TR-FMPB-022 ***← Elementary tool operational requirement***

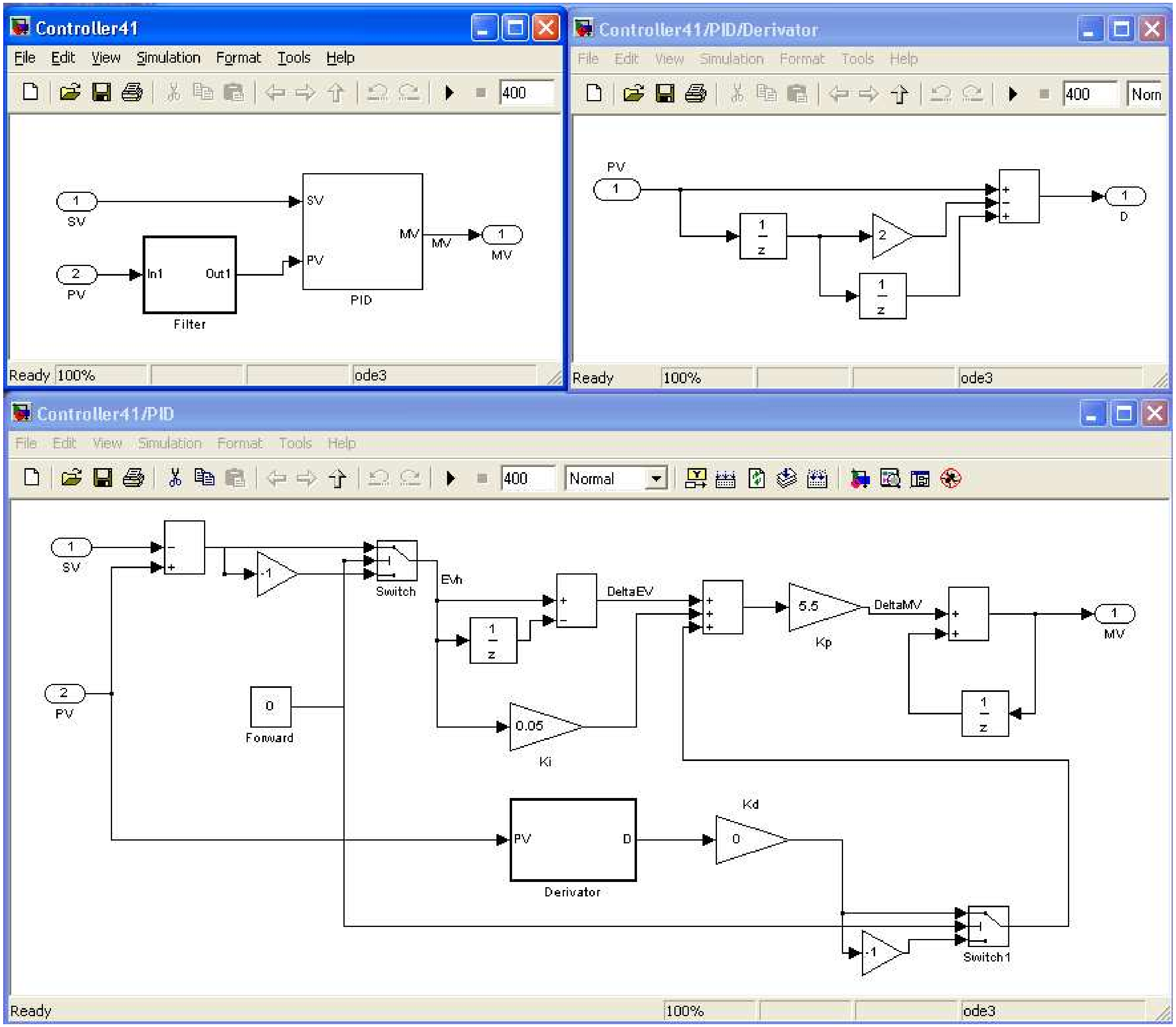
The tool must expand virtual (sub)systems

For each block of type SystemBlock in the source model:

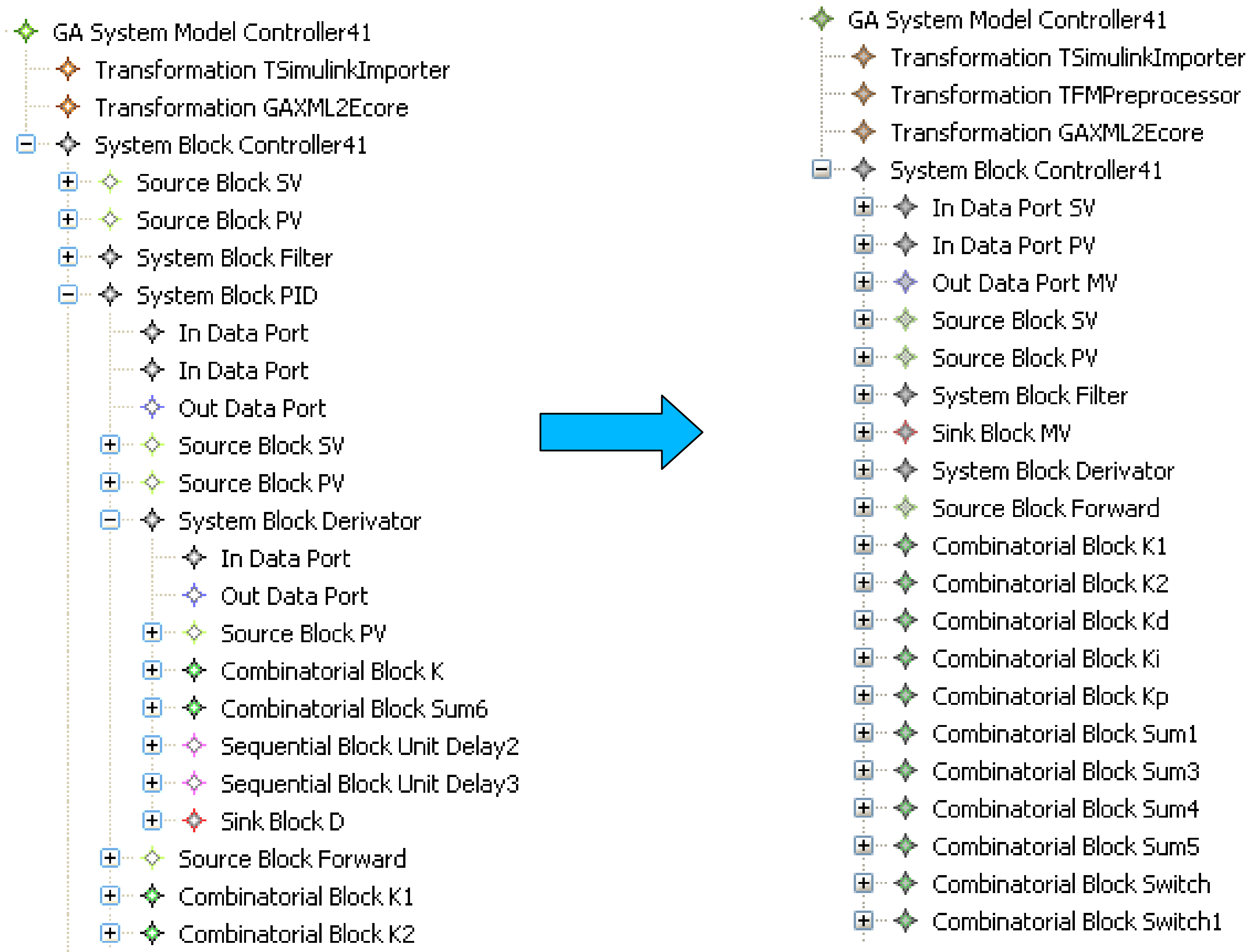
- If it is an atomic system, the tool processes the contents of the system in order to replace any inner virtual systems, but does not flatten the current system
- If it is a virtual system, the tool replaces the SystemBlock block by its content

Traced toolset operational requirement → ***GR-SL-B008***

Example model



Example (contd)



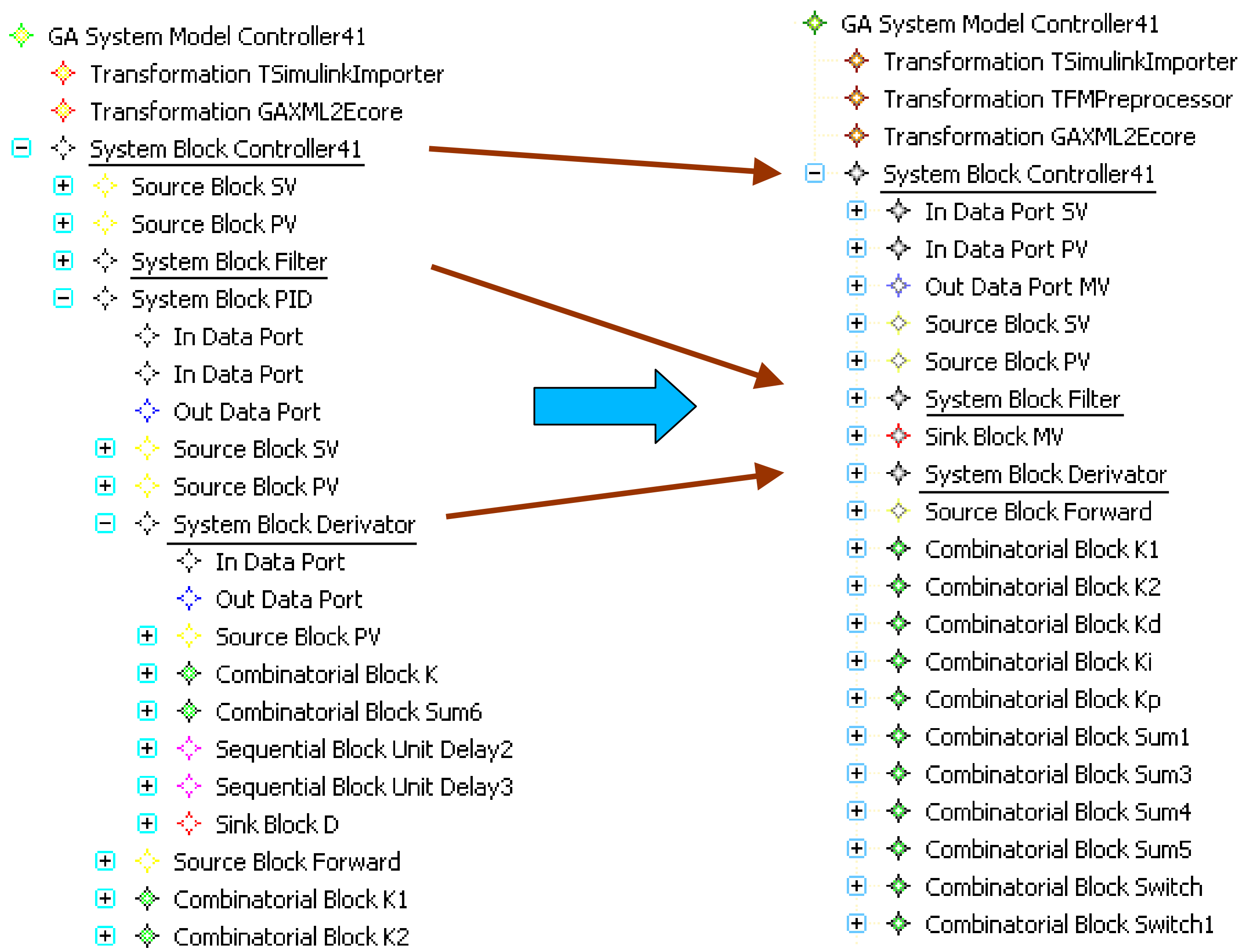
flattenedChildBlocks()

```
context SM2SM

def: flattenedChildBlocks()
    blocks->
        select(
              -- Primitive child blocks
              (not oclIsKindOf(SystemBlock)) or
             -- Atomic SubSystems
              (oclIsKindOf(SystemBlock)
                       and not isVirtual))

        -- Add children of non-atomic SubSystems
        ->union(
            self.blocks
                ->select(oclIsKindOf(SystemBlock)
                             and isVirtual)
                ->collect(flattenedChildBlocks()))
```

Hints from the Code Generator



Tool Requirement in MOF - OCL (using hints from the code generator)

```
context SM2SM

-- TR-FMPB-022:
inv:
    -- All non-atomic Systems in the src have a
    -- corresponding Sys2Sys link
    self.src.getAllElements()
        ->select(oclIsKindOf(SystemBlock))
        ->select(isVirtual = false)
             =
        self.links.sys2sys.src

    and

    -- Each System was correctly flattened
    self.links.sys2sys
        ->forAll(
            tgt.blocks = src.flattenedChildBlocks())
```

Summary of Translation Validation with MOF - OCL

- Fits well the classical MDE software development process
  - Natural language requirements can be well formalised in OCL by a domain specialist
  - If the requirements evolve, then the rules can be easily adjusted. Changes have only local effect
  - Implementers have precise requirements and need not have deep domain knowledge
- Rather light and stable tool-chain
- Consistency of the set of rules is only up to the specifier
  - Semantic analysis can be performed as a separate stage, by a different person/team – separation of concerns
- Completeness (model coverage) analysis is complicated
- Need to validate each transformation instance

Thank you!