Integrated Formal Approach for a Qualified Critical Code Generator

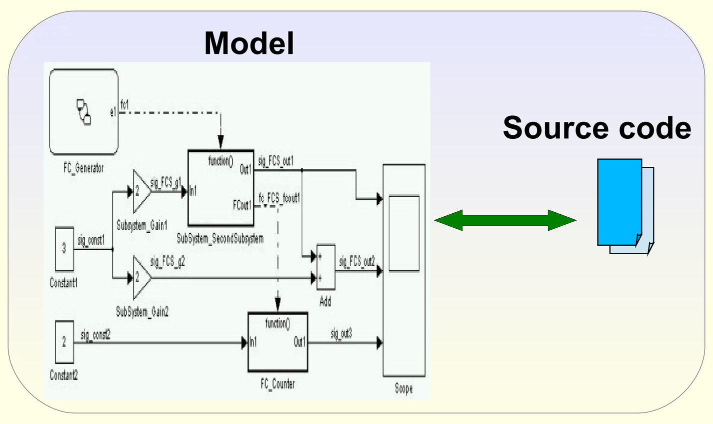
N. Izerrouken[1,2](nizerrou@N7.fr), M. Pantel[2], X. Thirioux[2] and O. Ssi Yan Kai[1]

[1]Continental Automotive, Innovation Center, Toulouse, France

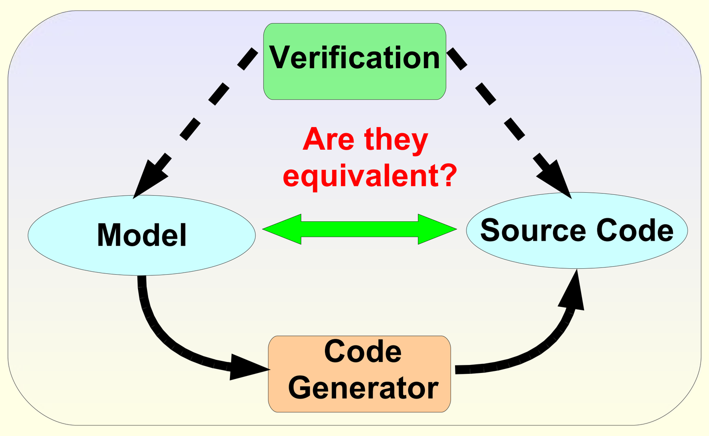
[2]Institut de Recherche en Informatique de Toulouse, Institut National Polytechnique de Toulouse, France

Problem Statement

- Code generators reduce development time and verification costs (compliance of source code w.r.t. model-based design)
- Industry is aware that critical systems require more rigorous verification than classical testing => formal specification & verification

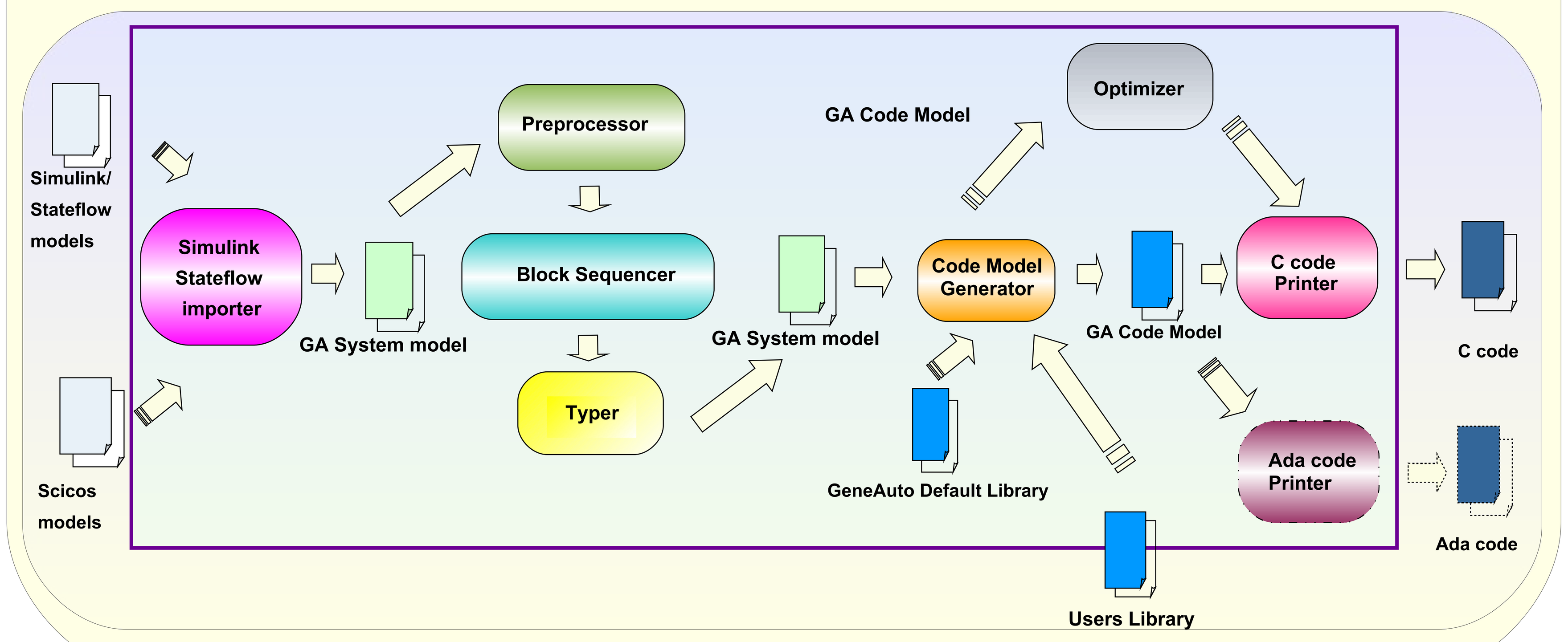


- Formal verification of:
  - Input model
  - Generated source code
  - Compliance generated code/model
- Complex analysis of verification failures => Verification of the Code generator itself



Context: GeneAuto

- GeneAuto : Automatic code Generator for critical embedded systems dedicated to transportation domain.
- GeneAuto is split into elementary tools
  - Easier to specify, verify and validate
  - Several implementations can be provided



Main goals

- Reduction of industrial unit testing costs
- Qualification of GeneAuto using DO178B/ED-12B recommendations
- Pragmatic integration of formal technologies into develpment tools for safety critical systems

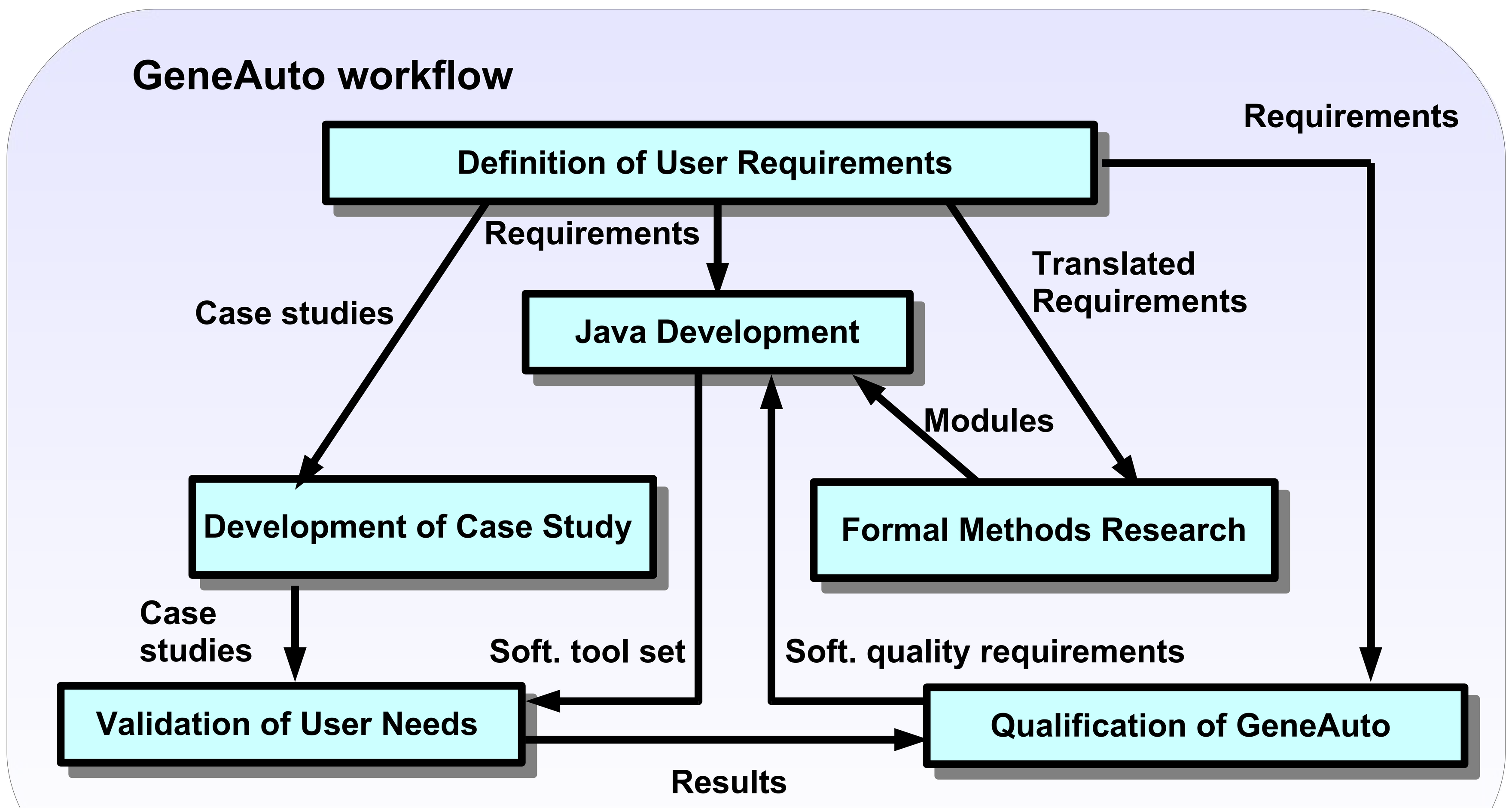
Development Process

Choices

- Specification, verification and validation using the Coq proof assistant
- Integration of the formal elementary tools to the GeneAuto tool chain
- Qualification of the development process of GeneAuto containing classical Java and formal elementary tools (Coq/OCaml)

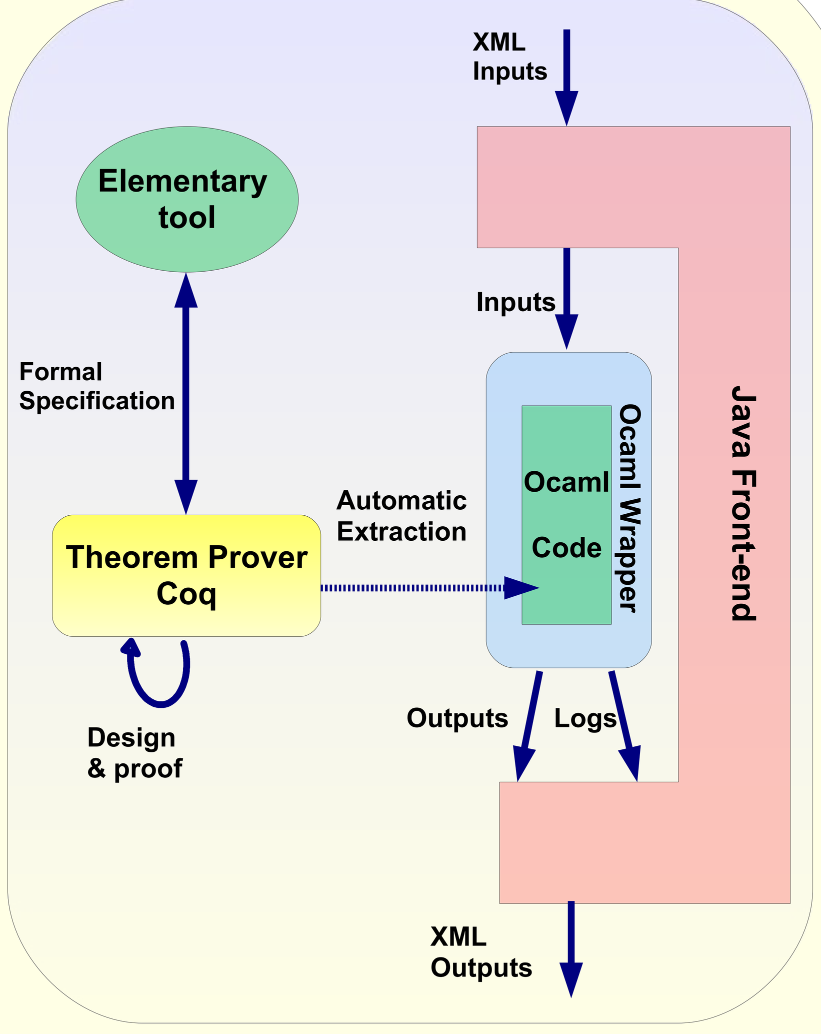
For each elementary tool

- Translation of user/tool requirements from natural to formal language (complex task, human proof reading)
- Formal specification of the tool requirements and design
- Formal verification of specified properties (correctness of Block Sequencer,Typer, etc.)



Integration of Formal Methods

- Each elementary tool
  - is developed and verified in Coq;
  - is verified and extracted in OCaml: extracted code preserves the properties proved in Coq.
- Java front-end
  - reads input XML models;
  - executes extracted Ocaml Wrapper;
  - writes output XML models.
- Ocaml Wrapper
  - reads input models;
  - executes the extracted OCaml code (sequencer, typer, etc.);
  - writes the output result (execution order, types, etc.).



Qualification Concerns

Example of translation of requirements

- From natural language

```
F6.1 Sort blocks based on data-flow constraints.
F6.3 Sort blocks with partial ordering according to priority from the input
model.
F6.4 Sort blocks that are still partially ordered according to their graphical
position in the input model.
```

- To Coq language

```
Definition correct_execution_order_dataflow
(m : ModelType) (s : SequencedModelType) : Prop :=
  forall (d : nat), (O < d) /\ (d <= m.signalsNumber) ->
   ((s.signalKind = DataSignal) ->
   (~ (isControlled s.src m)) ->
   (~ (isControlled s.dst m)) ->
   (s.dst.blockKind = CombinatorialBlock) ->
   (s.src.blockKind = CombinatorialBlock) ->
   ((s.sequencedBlocks d.src) = (Position _)) ->
   ((s.sequencedBlocks d.dst) = (Position _)) ->
     let (Position posSrc) = (s.sequencedBlocks d.src) in
     let (Position posDst) = (s.sequencedBlocks d.dst) in
       posSrc < posDst.
```

Qualification process

- Qualification of the development process of Java components
  - Detailed documented development process using DO178B/ED-12B
  - Validation process done through testing and cross-reading
- Qualification of the formal elementary tools
  - Coq proof checker partially verified
  - Coq extractor generates Ocaml code structurally similar to Coq specification
  - Removal of unit & integration test phase from the formally developed elementary tools in DO178B/ED-12B

Main Results

- Mixing classical and formal development
- Development of correct-by-construction components
  - ~4500 lines of Coq code and more than 130 proved theorems for the Block Sequencer
- Block Sequencer case study successfully integrated into GeneAuto
- Application to Real-size systems from transportation domains

Case Study	Satellite Orbit Control	"Knock" reduction Software	Airplane Flight Control System	Satellite Agile Control System	Sensor Networks
Model blocks	1085	5793	2800	1931	1108
Depth	8	9	7	6	7

- Runtime cost is comparable with similar tools (eg. Mathworks RTW)
- Qualification of the development process
  - Classical development
  - Formal components (Block Sequencer case study)