Certification and qualification concerns in the development of safety critical systems

Ricardo Bedin-França, Jean-Charles Dalbin, Denis Favre-Félix, Pierre-Loïc Garoche, Marc Pantel, Frédéric Pothon, Virginie Wiels, . . .

IRIT - ACADIE
ONERA - DTIM
AIRBUS Operations

JTRES 2010 — Thursday August the 20th 2010

Plan

Plan

ACADIE

Safe MDE concerns

- Main purpose: Safety critical systems
- Main approach: formal specification and verification
- Problems: expressiveness, decidability, completeness, consistency

Safe MDE concerns II

- Proposals: Raise abstraction
  - Higher level programming languages and frameworks
  - Domain specific (modeling) languages
    - easy to access for end users
    - with a simple formal embedding
    - with automatic verification tools
    - with usefull validation and verification results
    - that are accepted by certification authorities
- Needs:
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

ACADIE

Safe MDE concerns II

- Proposals: Raise abstraction
  - Higher level programming languages and frameworks
  - Domain specific (modeling) languages
    - easy to access for end users
    - with a simple formal embedding
    - with automatic verification tools
    - with usefull validation and verification results
    - that are accepted by certification authorities
- Needs:
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

ACADIE

Related past projects

- RNTL COTRE: Transformation to verification languages
- ACI FIACRE: Intermediate verification language
- ITEA GeneAuto: Qualified Simulink/Stateflow to C code generator
- ITEA ES_PASS: Static analysis for Product insurance
- ITEA SPICES: AADL behavioral annex
- ANR OpenEmbedd: AADL to FIACRE verification chain (Kermeta based)
- CNES (French Space Agency) AutoJava: profiled UML to RTSJ code generator

ACADIE

Related current projects

- FUI TOPCASED: Metamodels semantics, Model animators, Verification chains based on model transformations
- ANR SPaCIFY: GeneAuto + AADL = Synoptic <-> Polychrony (Kermeta based)
- ANR iTemis: SOA/SCA verification
- FRAE quarteFt: model transformation based on Java/TOM for AADL to FIACRE
- ITEA2 OPEES: Formal methods and Certification authorities
- JTI ARTEMISE CESAR: V & V view for safety critical components.

ACADIE

Plan

A bit of wording

- Requirement: What the end user expects from a system
  - High level: focus on end users needs (user provided)
    - Translate profiled UML to RTSJ; C to PowerPC
    - Generate test inputs and expected outputs from a system specification
    - Prove the absence of runtime errors
    - Compute a precise estimation of WCET
    - Schedule activities
  - Low level: focus on technical solutions (developer provided)
    - Relies on abstract interpretation for properties estimation
    - on graph coloring for register allocation
    - on linear programming for task scheduling
    - Generates a C function for each Simulink atomic sub-system
    - a RTSJ class for each UML class
  - Traceability links between various requirements, design and implementation choices

ACADIE

A bit of wording

- Requirement: What the end user expects from a system
  - High level: focus on end users needs (user provided)
    - Translate profiled UML to RTSJ; C to PowerPC
    - Generate test inputs and expected outputs from a system specification
    - Prove the absence of runtime errors
    - Compute a precise estimation of WCET
    - Schedule activities
  - Low level: focus on technical solutions (developer provided)
    - Relies on abstract interpretation for properties estimation
    - on graph coloring for register allocation
    - on linear programming for task scheduling
    - Generates a C function for each Simulink atomic sub-system
    - a RTSJ class for each UML class
  - Traceability links between various requirements, design and implementation choices

ACADIE

A bit of wording

- Requirement: What the end user expects from a system
  - High level: focus on end users needs (user provided)
    - Translate profiled UML to RTSJ; C to PowerPC
    - Generate test inputs and expected outputs from a system specification
    - Prove the absence of runtime errors
    - Compute a precise estimation of WCET
    - Schedule activities
  - Low level: focus on technical solutions (developer provided)
    - Relies on abstract interpretation for properties estimation
    - on graph coloring for register allocation
    - on linear programming for task scheduling
    - Generates a C function for each Simulink atomic sub-system
    - a RTSJ class for each UML class
  - Traceability links between various requirements, design and implementation choices

ACADIE

A bit of wording II

- Verification: System fulfills its requirements explicit specification
- Validation: System fulfills its requirements implicit human needs
- Certification: System (and its development) follows standards (DO-178, IEC-61508, ISO-26262, ...)
- Qualification: Tools for system development follows standards
- Certification and qualification: System context related

A bit of wording II

- Verification: System fulfills its requirements explicit specification
- Validation: System fulfills its requirements implicit human needs
- Certification: System (and its development) follows standards (DO-178, IEC-61508, ISO-26262, . . .)
- Qualification: Tools for system development follows standards
- Certification and qualification: System context related

A bit of wording II

- Verification: System fulfills its requirements explicit specification
- Validation: System fulfills its requirements implicit human needs
- Certification: System (and its development) follows standards (DO-178, IEC-61508, ISO-26262, ...)
- Qualification: Tools for system development follows standards
- Certification and qualification: System context related

DO-178B/ED-12B standards: Certification

- Software in aeronautics: Design Assurance Level (A down to E)
- Most constraining standards up to now
  accepted by other standards (automotive, space, ...)
- Main concern: Safety of passengers
- Main purpose: Provide confidence in the system and its development
- Key issue: Choose the strategy and technologies that will minimize risks (no restriction)
- Process and test-centered approach
  - Definition of a precise process (development/verification)
  - MCDC test coverage
    truth-table lines of sub-expressions in conditions
  - Asymmetry with independence argument: several implementation by different teams, with different tools, ...

DO-178B/ED-12B standards: Certification

- Software in aeronautics: Design Assurance Level (A down to E)
- Most constraining standards up to now
  accepted by other standards (automotive, space, ...)
- Main concern: Safety of passengers
- Main purpose: Provide confidence in the system and its development
- Key issue: Choose the strategy and technologies that will minimize risks (no restriction)
- Process and test-centered approach
  - Definition of a precise process (development/verification)
  - MCDC test coverage
    truth-table lines of sub-expressions in conditions
  - Asymmetry with independence argument: several implementation by different teams, with different tools, ...

DO-178B/ED-12B standards: Certification

- Software in aeronautics: Design Assurance Level (A down to E)
- Most constraining standards up to now
  accepted by other standards (automotive, space, ...)
- Main concern: Safety of passengers
- Main purpose: Provide confidence in the system and its development
- Key issue: Choose the strategy and technologies that will minimize risks (no restriction)
- Process and test-centered approach
  - Definition of a precise process (development/verification)
  - MCDC test coverage
    truth-table lines of sub-expressions in conditions
  - Asymmetry with independence argument: several implementation by different teams, with different tools, ...

ACADIE

DO-178B/ED-12B standards: Qualification

- Development tools: Tools whose output is part of airborne software and thus can introduce errors (same constraints as the developed system).
- Verification tools: Tools that cannot introduce errors, but may fail to detect them (much softer constraints: black box V & V).
- No proof of error absence category

DO-178C/ED-12C standards: Qualification

- Introduce detailed Tool Qualification Level (1 downto 5)
- Criteria 1: A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
- Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
  - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
  - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
- Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
- Still no proof of error absence category (might be TQL-2 for DAL A).

ACADIE

DO-178C/ED-12C standards: Qualification

- Introduce detailed Tool Qualification Level (1 downto 5)
- Criteria 1: A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
- Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
  - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
  - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
- Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
- Still no proof of error absence category (might be TQL-2 for DAL A).

ACADIE

DO-178C/ED-12C standards: Qualification

- Introduce detailed Tool Qualification Level (1 downto 5)
  - Criteria 1: A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

ACADIE

DO-178C/ED-12C standards: Qualification

- Introduce detailed Tool Qualification Level (1 downto 5)
  - Criteria 1: A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
- Still no proof of error absence category (might be TQL-2 for DAL A).

ACADIE

Common documents

- Phase 1: Cooperative process definition:
  - Plan for software aspects of certification (PSAC)
  - Development plan (SDP)
  - Verification plan (SVP)
  - Configuration management plan (SCMP)
  - Quality assurance plan (SQAP)
  - Tool qualification plan

Common documents (qualification case)

- Phase 2: Process application verification
  - User requirements
  - Tool architecture (elementary tools and their assembly)
  - Tool requirements: Can be refined user requirements or derived requirements (linked to technology choices, should be avoided or strongly justified)
  - Development and verification results (each elementary tools)
  - Traceability links
  - Verification results (user level)

Some comments

- Standards were designed for systems not tools: Adaptation required
- MCDC not mandatory for tools, but similar arguments might be required
- Traceability of all artefacts in the development, relate requirements, design and implementation choices
- Purpose is to provide confidence
- Both cooperative and coercive approach
- Any verification technology can be used, from proofreading to automatic proof if confidence is given
- Choose the strategy and technologies that will best reduce risks

Some comments

- Standards were designed for systems not tools: Adaptation required
- MCDC not mandatory for tools, but similar arguments might be required
- Traceability of all artefacts in the development, relate requirements, design and implementation choices
- Purpose is to provide confidence
- Both cooperative and coercive approach
- Any verification technology can be used, from proofreading to automatic proof if confidence is given
- Choose the strategy and technologies that will best reduce risks

Some comments II

- Must be applied as soon as possible (cost reduction)
- Small is beautiful (simplicity is the key)
- Certification authorities need to understand the technologies
- Cross-experiments are mandatory (classical w.r.t. formal methods)

Some comments II

- Must be applied as soon as possible (cost reduction)
- Small is beautiful (simplicity is the key)
- Certification authorities need to understand the technologies
- Cross-experiments are mandatory (classical w.r.t. formal methods)

Plan

Transformation verification technologies

- Verification subject:
  - Transformation: done once, no verification at use, white box, very high cost
  - Transformation application: done at each use, black box, easier, complex error management
- Classical technologies:
  - Document independant proofreading (requirements, specification, implementation)
  - Test
    - Unit, Integration, Functional, Deployment level
    - Requirement based test coverage
    - Source code test coverage
    - Structural coverage, Decision coverage, Multiple Condition Decision Coverage (MCDC)

ACADIE

Transformation verification technologies II

- Formal technologies (require formal specification):
  - Automated test generation
  - Model checking (abstraction of the system)
  - Static analysis (abstraction of the language)
  - Automated proof
  - Assisted (human in the loop) proof
- Transformation case
  - Transformation specification: Structural/Behavioral
  - Proof of transformation correctness
  - Links with certification/qualification

Classical development and verification process

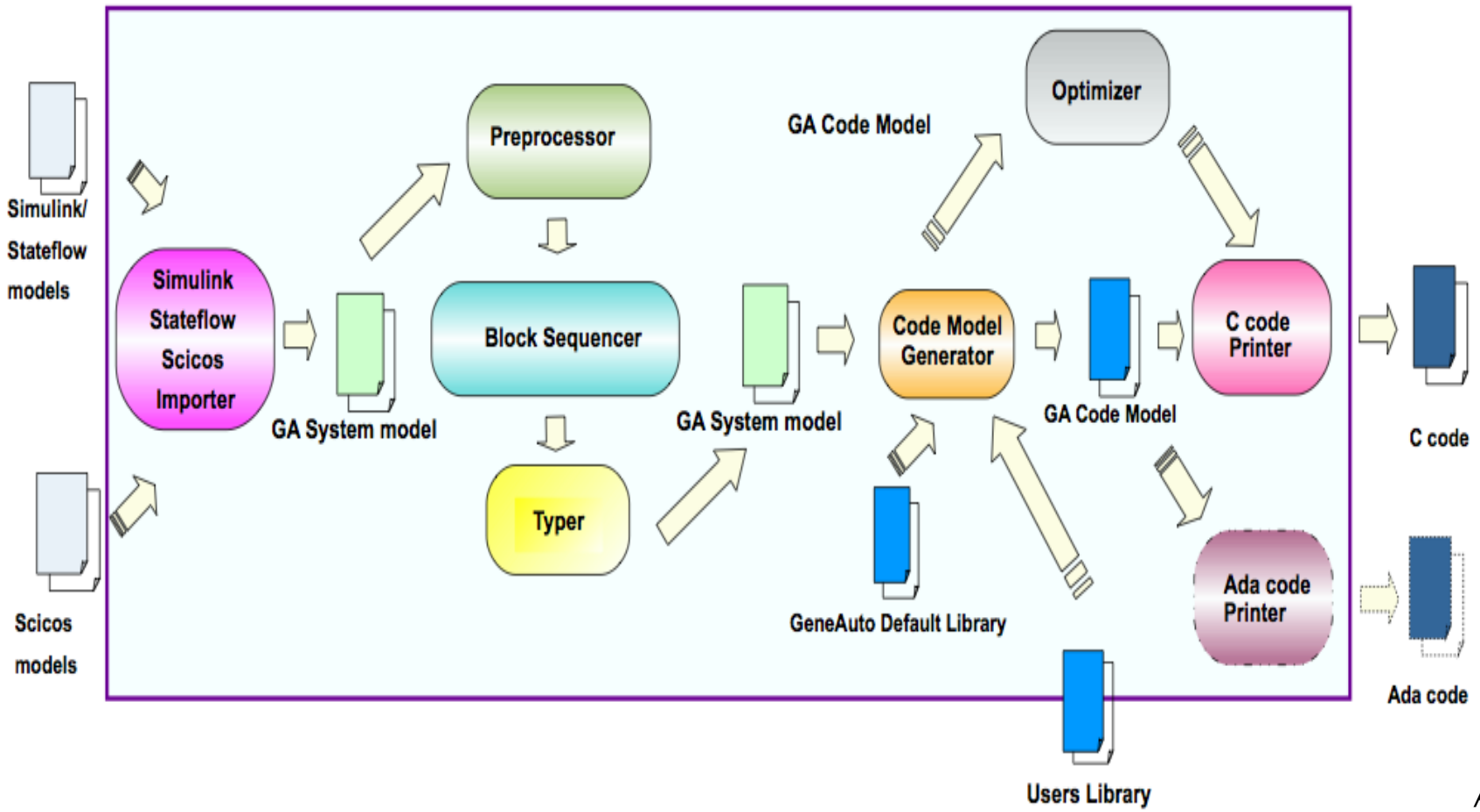
- Tool development, verification and qualification plans
- User requirements
- Tool requirements (human proofreading)
- Test plan (requirements based coverage, code coverage verification)
- Implementation and test application

GeneAuto experiment: Proof assistant based

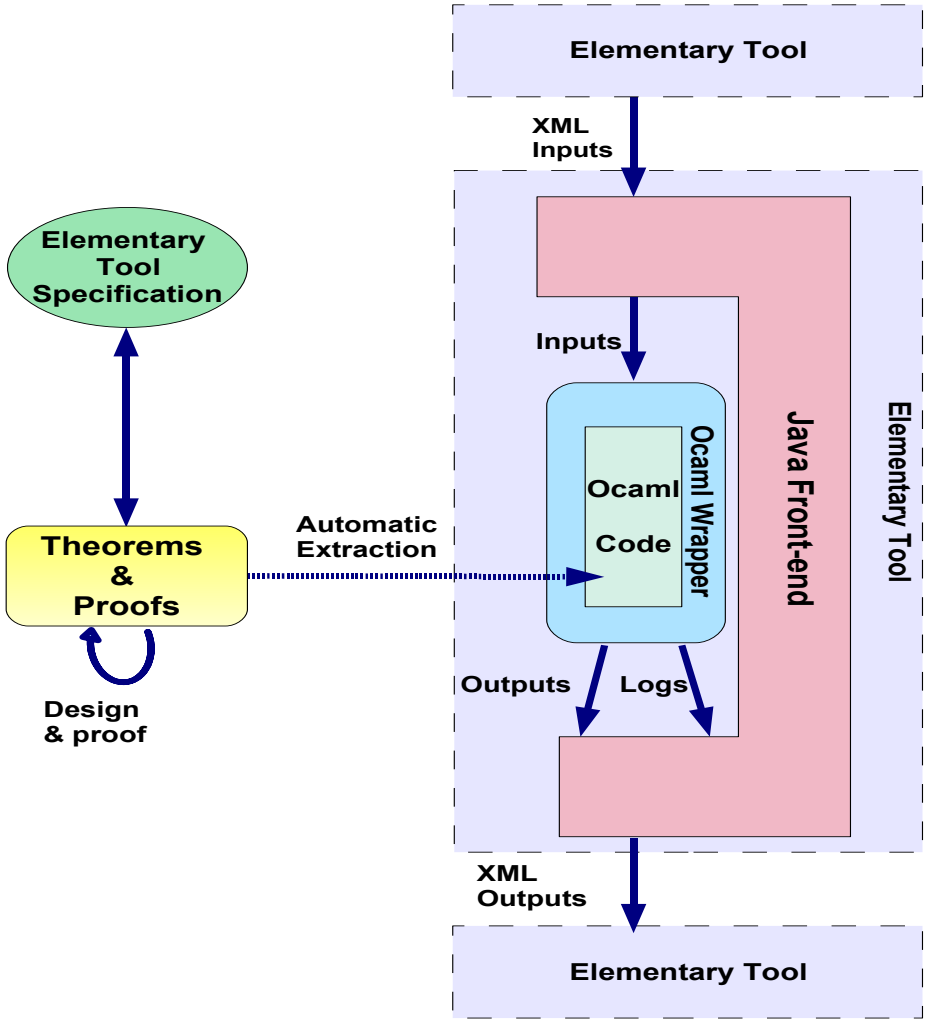
- Derived from the classical process, validated by french certification bodies
- Formal specification using Coq of tool requirements, implementation and correctness
- Proofreading verification of requirements specification
- Automated verification of specification correctness
- Extraction of OCaML source implementation
- Proofreading verification of extracted OCaML source
- Integration of OCaML implementation with Java/XML implementation (communication through simple text files with regular grammars)
- Proofreading verification of OCaML/Java wrappers (simple regular grammar parsing)
- Test-based verification of user requirements conformance

GeneAuto Code Generator Architecture

Split into independent modules (easier V & V and qualification)



Integration



An example: User requirements R-CG-040

- F6 – Determine execution order of functional model blocks
  The execution order generated by the ACG must be as close as possible to that in Simulink and it shall be possible to visualise the execution order
  Same scheduling as Simulink is required to ensure that generated code conforms to Simulink simulations.
- Refinement
  - F6.1 Sort blocks based on data-flow constraints
  - F6.2 Refine the order according to control 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

The same one in Coq

```
Definition correct_execution_order_dataflow
(m: ModelType) (s: SequencedModelType) : Prop :=
 forall (d:nat), (0<d) /\ (d <= m.signalsNumber) ->
 ((s.signalKind = DataSignal) ->
 (~ (isControlled s.src m)) ->
 (~ (isControlled s.dst m)) ->
 (s.src.BlockKind = CombinatorialBlock) ->
 (s.dst.BlockKind = CombinatorialBlock) ->
  let (Position posSrc) = (s.sequencedBlocks d.src) in
  let (Position posDst) = (s.sequencedBlocks d.dst) in
   posSrc < posDst.
```

ACADIE

Open questions ?

What are:

- User requirement for a transformation/verification ?
- Tool requirement for a transformation/verification ?
- Formal specification for a transformation/verification ?
- Test coverage for a transformation/verification ?
- Test oracle for a transformation/verification ?
- Qualification constraint for transformation/verification languages ?
- Best strategy for tool verification (once vs at each use) ?

ACADIE

GeneAuto feedbacks

- From the certification perspective: Very good but...
  - Still some work on qualification of the proof assistant tools
    - Proof verifier
    - Program extractor
  - Complex management of input/output
- From the developer perspective:
  - High dependence to the technologies
  - Very high cost to use the technology
  - Not easy to subcontract
  - Scalability not ensured
  - Bad separation between semantics-based verification and requirements-based specification
  - Hard to assess development time
- On the use of Java: How to provide confidence in the libraries ?

ACADIE

Going further: CompCert use experiment

- CompCert: C to PowerPC optimising code generator developed at INRIA by Xavier Leroy
- PhD thesis at Airbus: Improve certified code efficiency
  - Metrics: WCET, Code and memory size, Cache and memory accesses
  - Improvements of the various phases from models to embedded binary code
  - New verification process using formal methods
  - First CompCert experiments: -12% WCET, -25% code size, -72% cache read, -65% cadre write
  - Design of a CompCert dedicated verification process
  - Feed static analysis results (Astrée, frama-C) from C to binary through CompCert (improve WCET precision)
  - Improve SCADE block scheduling to reduce memory accesses (signal liveness)
  - Design of a whole development cycle verification process

ACADIE

Going further: CompCert use experiment

- CompCert: C to PowerPC optimising code generator developed at INRIA by Xavier Leroy
- PhD thesis at Airbus: Improve certified code efficiency
  - Metrics: WCET, Code and memory size, Cache and memory accesses
  - Improvements of the various phases from models to embedded binary code
  - New verification process using formal methods
  - First CompCert experiments: -12% WCET, -25% code size, -72% cache read, -65% cadre write
  - Design of a CompCert dedicated verification process
  - Feed static analysis results (Astrée, frama-C) from C to binary through CompCert (improve WCET precision)
  - Improve SCADE block scheduling to reduce memory accesses (signal liveness)
  - Design of a whole development cycle verification process

ACADIE

Proposal: Mixed approach

- Separate specification verification from implementation verification
- Define explicitly semantics traceability link metamodel
- Specify transformation as properties of links
- Implementation verification (mostly syntactic)
  - Implementation must generate both target and links
  - Implementation verification checks properties on generated links links
- Specification verification: Prove the semantics equivalence between source and target in a trace link
- Soon to be started PhD thesis at Airbus

Early feedbacks

- Separation of concerns:
  - Industrial partners: Specification, Implementation, Implementation verification (mainly syntactic)
  - Academic partners: Specification verification (semantics)
- Very good subcontracting capabilities
- Almost no technology constraints on the industrial partner (classical technologies)
- Good scalability
- Easy to analyse syntactic error reports
- Enables to modify generated code and links
- Parallel work between syntactic and semantics concerns

ACADIE

Work in progress

- Positive first experiments on simple use cases from GeneAuto
- But requires some grayboxing (expose parts of the internals)
  - Flattening of statecharts
  - Either very complex specification (doing the flattening)
  - Or express the fixpoint nature of implementation (in the specification)
- Require full scale experiments
- Require exchange with certification authorities
- Require qualified syntactic verification tool (OCL-like, but simpler)
- Require explicit relations between syntactic and semantics work
- Require explicit description of semantics in metamodels

ACADIE

Plan

Static analysis tools

- Several kind of tools
  - Qualitative and quantitative properties
  - Fixed or user defined properties
  - Semantic abstraction or Proof technologies
- Common aspects: Common pre-qualification
  - Product (source of binary code) reader: fully common ?
  - Configuration (properties, ...) reader: partly common
  - Result writer and browser: partly common ?
- Split the verification tool in a sequence of elementary activities
  - Common ones (pre-qualification could be shared)
  - Technology specific ones
  - Easier to specify, to validate and to verify
  - Can be physical or virtual (produce intermediate results even in a single tool)

ACADIE

Required activities

- Specify user requirements
- Specify tool architecture (elementary tools and their assembly)
- Specify tool level requirements (elementary tools and their assembly)
- Specify functional test cases and results
- Choose verification strategy:
  - Tool verification or Result verification
  - Integration and unit tests (eventually with test generators and oracles)
  - Proof reading of tool source or test results
  - Formal verification of the verification tool itself (i.e. Coq in Coq, Compcert in Coq, . . .)

Abstraction kind

- Translate to non standard semantics
- Compute recursive equations
- Compute fixpoint of equations
  - Fixpoint algorithm
  - Abstract domains and operators
  - Widening, narrowing
- Check that properties are satisfied on the abstract values
- Produce user friendly feedback (related to product and its standard semantics)

ACADIE

Deductive kind

- Produce proof obligations (weakest precondition, verification condition, . . .)
- Check the satisfaction of proof obligations
  - Proof term rewriting to simpler language
  - Split to different sub-languages (pure logic, arithmetic, . . .)
  - Apply heuristics to produce a proof term
  - Check the correctness of the proof term
  - Produce failure feedback or proof certificate (related to product and its standard semantics)
- Produce user friendly feedback

Potential strategy: Common parts

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer:
  - Cross-reading
  - Introduce dual reader/writer: check composition is identity
  - Asymmetric implementation: Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified:
  - Formal tool requirements: foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification: properties of the trace links given as tool requirements
  - Semantic verification: validation of the technology
- User-friendly feedback: Code generation based on trace

Potential strategy: Common parts

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer:
  - Cross-reading
  - Introduce dual reader/writer: check composition is identity
  - Asymmetric implementation: Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified:
  - Formal tool requirements: foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification: properties of the trace links given as tool requirements
  - Semantic verification: validation of the technology
- User-friendly feedback: Code generation based on trace

ACADIE

Potential strategy: Common parts

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer:
  - Cross-reading
  - Introduce dual reader/writer: check composition is identity
  - Asymmetric implementation: Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified:
  - Formal tool requirements: foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification: properties of the trace links given as tool requirements
  - Semantic verification: validation of the technology
- User-friendly feedback: Code generation based on trace

Potential strategy: Common parts

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer:
  - Cross-reading
  - Introduce dual reader/writer: check composition is identity
  - Asymmetric implementation: Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified:
  - Formal tool requirements: foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification: properties of the trace links given as tool requirements
  - Semantic verification: validation of the technology
- User-friendly feedback: Code generation based on trace

Potential strategy: Abstraction kind

- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification: monotony at the equations-level
  - Semantic verification: soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification: soundness of the abstraction

Potential strategy: Abstraction kind

- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification: monotony at the equations-level
  - Semantic verification: soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification: soundness of the abstraction

Potential strategy: Abstraction kind

- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification: monotony at the equations-level
  - Semantic verification: soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification: soundness of the abstraction

ACADIE

Potential strategy: Abstraction kind

- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification: monotony at the equations-level
  - Semantic verification: soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification: soundness of the abstraction

ACADIE

Potential strategy: Deductive kind

- Proof obligation computation is a kind of code generation
  - Semantic verification: correctness of the axiomatic semantics
- Satisfaction of the proof obligations:
  - No verification on proof certificate generation
  - Verification of the certificate itself (much simpler than some heuristic-based automatic prover)
  - Term rewriting can be considered as code generation (endogenous)
  - Curry-Howard type checking can be verified in a similar way
  - Rely on Coq In Coq, Isabelle in Isabelle, . . .

ACADIE

What about validation of the technologies ?

- Mainly scientific work and a lot of publications
- Brings confidence but paperwork is not enough
- Mechanized is better but still not enough
- Functional user level tests still mandatory currently
- Mixed system verification experiments (both tests and static analysis)
- Reverse analysis of existing systems

ACADIE

Synthesis

- Technical exchange with certification authorities mandatory
- Cross experiments and reverse engineering experiments mandatory
- Verification strategy must be designed early to choose the right architecture and trace information
- Semi-formal (even formal) requirements must be written as soon as possible

ACADIE