INPT/IRIT-ACADIE/LAAS-OLC
Safe MDE concerns
Model transformation and verification
Certification and Qualification concerns

Marc Pantel

IRIT – Institut de Recherche en Informatique de Toulouse
ACADIE team

CNRS-LAAS – Laboratoire d'Architecture et d'Analyse des Systèmes
OLC team

2009/09/23 Aerospace Valley GeneAuto meeting

Safe MDE concerns

- Main purpose: Safety critical systems
- Main approach: formal specification and verification
- Problems: expressiveness, decidability, completeness, consistency
- Proposals: 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
- Needs:
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

Involvment in OPEES

- WP3: Certification/Qualification aspects
  - Model Driven Engineering
  - Formal Specification and Verification (GeneAuto/ES_PASS followup)
- WP4: Tool integration use cases
  - AADL/FIACRE/TINA verification toolchain
  - GeneAuto generation toolchain (Simulink/Stateflow to MISRA C)

Some definitions

- Validation: Implementation satisfies the end user needs
- Verification: Implementation satisfies its specification
- Validation of a verified implementation: Specification satisfies the end user needs
- Certification: Products satisfies standards (DO-178, IEC-61508, ISO-26262, . . .)

Some definitions II

- Qualification (DO-178 definition): Tools used to develop product must satisfies standards
  - Automated code generation (Model transformation): Same constraints as the generated code
  - (Model or) Code verification:
    - Absence of error (test removal): Almost the same constraints as the generated code
    - Error detection (test required): Low criticity system constraints
- User requirements: Specification of the end user needs (Simulink to C, UML to Java, Stateflow to UML State machine, . . .)
- Tool requirements: Specification of the tool (a block to a function, a class for a class, . . .), should refine the previous one

Verification technologies

- Verification subject:
  - Transformation: done once, no verification at use, white box, very complex
  - Transformation application: done at each use, black box, easier, complex error mangagement
- Classical technologies:
  - Document (cross-)reading (requirements, specification, implementation)
  - Test
    - Requirement based test coverage
    - Source code test coverage
    - Structural coverage, Decision coverage, Multiple Condition Decision Coverage

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

Development and verification process

- Classical process:
  - User requirements
  - Tool requirements (cross-reading)
  - Test plan (requirements based coverage, code coverage verification)
  - Implementation and test application
  - Open question: Qualification of the development tools (transformation language) ?
  - Open question: What are a transformation tool requirements, specification, implementation ?

Development and verification process II

- Introduction of formal methods:
  - GeneAuto experiment: derived from the classical process, validated by french certification bodies
  - Formal specification of tool requirements, implementation and correctness (experiments using Coq during GeneAuto)
  - Cross-reading verication of requirements specification
  - Automated verification of specification correctness
  - Extraction of OCaML source implementation
  - Cross-reading verication of extracted OCaML source
  - Integration of OCaML implementation with Java/XML implementation (communication through simple text files with regular grammars)
  - Cross-reading verication of OCaML/Java wrappers (simple regular grammar parsing)
  - Test-based verification of user requirements conformance
  - Open question: Qualification of the development tools (specification verifier) ?

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) ?

Using GeneAuto with UML/SysML models

Starting points:

- Stateflow strongly inspired by UML2/Statecharts
- Question from Rockwell Collins to avoid using closed model editors
- UML2/Activity provide diagrams for object-flows (similar data-flows)
- SysML improves these aspects to represent continuous systems
- SysML adapts UML2 to system engineering

Using GeneAuto with UML/SysML models

Current point of view:

- SysML does not provide basic blocks (GeneAuto block library)
- The UML action language may be more expressive than Stateflow ones
- SysML could be used for high level architecture, mode management, protocols
- SysML should be extended to integrate block libraries
- Studies of mathematical expression in SysML for continuous systems
- Studies of property/requirement formal description in SysML and generation toward Frama-C and Spark/ADA
- Simulink/Stateflow has no concurrent semantics whereas SysML/UML is concurrent

ACADIE

Optimisation issues

- Model level optimisation: Block sequencing to minimize memory use, or WCET
- Code level optimisation: Classical issues, partly implemented
- CodeModel/C/Ada too high level for specifying low level memory model
- GA models express concurrency that is lost after the sequencing
- Potential approach: switch to intermediate SSA/data flow format inside compilers (for example LLVM format)
- Main problem: Qualification issues for optimisers (Specification Validation and Implementation Verification)

IRIT ACADIE

Certification issues

- GeneAuto project ended with a very good understanding between academic and industrial partners
- Some experiments were conducted with proof assistants for tool verification
- A process was designed and submitted to CEAT with very good feedbacks
- However, proof assistant qualification issues must be taken into account at a potential high cost (meeting with X. Leroy and P. Letouzey in 07/2009)
- The other approach is tool usage verification
  - Classical specification: Source and target metamodels, black box translation function
  - Automatic synthesis of the relation between model and generated code is very costly
  - Analysis and correction of detected generation errors is very costly

Certification issues

- Another proposal experimented in summer 2009: Gray box specification
  - Add a traceability metamodel which links source and target ones
  - The implementation must produce both the target model and the link model
  - Add OCL assertion on the links between and target to express the correctness of transformation
  - The result of each transformation will be checked with an OCL verifier
  - The implementation can be fully independant in any languages/technologies without constraints
  - Even the end user can correct the generated models and links

Certification issues

- Key issues:
  - development of a qualified OCL verifier (extremely useful for the community),
  - design of a validation process for the traceability metamodel and OCL constraints
- Much more user friendly than proof assistant use (proof assistant should be used for specification validation)
- The specification costs is almost the same of the currently done ones, and the verification is costless

Last, starting exchanges with Giorgia Tech and Nasa regarding verified continuous models to discrete algorithms