

Certification of Transformation Algorithms in Model-Driven Software Development

Miguel Garcia, Ralf Möller
<http://www.sts.tu-harburg.de>
2007-03-30



Miguel Garcia – STS – TUHH

Agenda

- Problem Statement and some Definitions
- Examples of metamodels and model transformations (EJB3QL, model compilers for Java EE)
- The certification mechanisms at our disposal, followed by a more in depth discussion of the particular technology choices made
- Tooling to support the certification process

Miguel Garcia – STS – TUHH



Hurdles for a program to overcome to be considered “useful”

1. Syntactically valid
2. Well-formed (for example, each usage is in scope of its single previous declaration).
Also called “static semantics”
3. Correctly typed, i.e. no runtime execution may result in a value of type T2 assigned to a location declared to hold values of type T1, with T2 not a subtype of T1

Even after all those filters, programs may still:

never terminate,
crash, or
compute wrong results!

Miguel Garcia – STS – TUHH



Making model compilers live up to their promises

Model compilers are programs that

1. take sentences in a (high-level, platform-independent) language as input
2. compile them into sentences in a language at a lower level of abstraction

- **The minimum we expect of a model transformations is**

Given well-formed, correctly typed input sentences
generate well-formed, correctly typed output sentences.

- **Sounds fair, right?**

Miguel Garcia – STS – TUHH



What we mean by model compilers: Examples

- **DASL, Sun Labs,**
a “model to deploy” web application specification language, abstracting away from persistence, transaction, and distribution
- **SecureUML, ETH Zurich's Information Security Group,**
“Security mechanisms are rather easy to state but difficult to realize ... starting from access-control descriptions, code and security configuration are generated”
- **WebML, Dipartimento di Elettronica e Informazione at Politecnico di Milano,**
“a visual notation for specifying the content, composition, and navigation features of hypertext applications”

Miguel Garcia – STS – TUHH



Metamodels are our friends

- **Whenever we want to determine if sentences are “well-formed”, we resort to the invariants specified in the metamodel for the DSL in question**
- **We can check at transformation time, before and after each run, whether well-formedness is maintained (i.e. whether the invariants in Object Constraint Language are satisfied) for some particular input-output pair**
- **But we can do better. (like 3GL compilers do)**

Miguel Garcia – STS – TUHH



Well-formedness example: EJB3QL (OO DB query language in Java EE)

Comparing values from different enumeration types makes no sense, but the grammar does not rule it out.

Solve that by adding an OCL invariant to the metamodel of EJB3QL:

```
context EnumCompExp
  inv comparedValuesBelongToTheSameEnumerationType :
    left.type() = right.type()
```

Table 1: EBNF for comparison_expression

```
comparison_expression ::=
  string_expression comparison_operator
  {string_expression | all_or_any_expression} |
  boolean_expression { = | <> }
  {boolean_expression | all_or_any_expression} |
  enum_expression { = | <> }
  {enum_expression | all_or_any_expression} |
  datetime_expression comparison_operator
  {datetime_expression | all_or_any_expression} |
  entity_expression { = | <> }
  {entity_expression | all_or_any_expression} |
  arithmetic_expression comparison_operator
  {arithmetic_expression | all_or_any_expression}
```

More examples at: Garcia, M. Formalizing the well-formedness rules of EJB3QL in UML + OCL, ATEM 2006 Workshop, co-located with MoDELS / UML 2006

Miguel Garcia – STS – TUHH



Aside: More praise for metamodels (1 of 2)

As if certification of transformations were not enough, metamodels are also leveraged to:

- Enable interoperability in a toolchain by facilitating the exchange of Abstract Syntax Trees (ASTs)

(Have you ever tried to further process the ASTs prepared by a C++ compiler front-end?)

- Once specified declaratively in OCL, invariants can be automatically translated into Java. Additionally, ad-hoc OCL queries can be interpreted at runtime.

(Compare that with writing a visitor each time one wants to find out something about an AST)

Miguel Garcia – STS – TUHH



Aside: More praise for metamodels (2 of 2)

- A metamodel can be augmented with annotations to univocally determine a concrete syntax. From it, a generator can derive:
 - a) grammars for different parser generators, thus encapsulating the parsing functionality and making parsers interchangeable;
 - b) Java classes whose instances represent Concrete Syntax Tree (CST) nodes, thus allowing for OCL to be used to query and constrain a CST;
 - c) an unparser (pretty-printer) from CST to textual notation;
 - d) a text editor supporting usability features such as syntax-directed completion, problem markers for violations of well-formedness, content completion suggestions, language-aware navigation, folding, and structural views, among others.
- Following a similar approach, a concrete *visual* syntax can be defined

Miguel Garcia – STS – TUHH



Back to certifying transformations (at transformation-design time)

Tough way: Hoare logic

- Tough because no end-to-end automation is possible, demanding skills in theorem proving

Not-so-tough way: Model Checking

Famous for spotting unwanted interactions in:

- cryptographic protocols
(e.g. impersonation in Needham-Schroeder)
 - memory-cache coherence protocols
 - wrong instruction reorderings in RISC processors
- In general, ideal for problems that can be formulated with state transition systems

Miguel Garcia – STS – TUHH



**However, model checking of
arbitrary algorithms is also possible**

Why?

Miguel Garcia – STS – TUHH



**However, model checking of
arbitrary algorithms is also possible**

Why?

Short answer: Because Leslie Lamport says so.

Miguel Garcia – STS – TUHH



However, model checking of arbitrary algorithms is also possible

Why?

Short answer: Because Leslie Lamport says so.

In detail ...

- TLA (the Temporal Logic of Actions) is a logic for specifying and reasoning about time-evolving, concurrent and reactive systems.
- It is the basis for TLA+, a complete system specification language.
- Which in turn is the basis for +CAL, a language to specify algorithms (be they deterministic or not, sequential or concurrent)

Miguel Garcia – STS – TUHH



If you used to know Pascal, you also know +CAL

For illustration, let's take a look at Euclid's algorithm (find the greatest common divisor of u and v)

```
while (u != 0)
{ if (u < v) { u := v || v := u } ; \* swap u and v
  u := u - v };
```

+CAL allows not just running an algorithm, but also checking whether assertions hold, for all (but finite) possible runs fulfilling start conditions

In the example, assuming u and v are integers in a range, +CAL can check whether v is actually the mathematical gcd:

$$\begin{aligned} \text{gcd}(x, y) \triangleq & \text{CHOOSE } i \in 1..x : \\ & \wedge x \% i = 0 \\ & \wedge y \% i = 0 \\ & \wedge \forall j \in 1..x : \wedge x \% j = 0 \\ & \wedge y \% j = 0 \\ & \Rightarrow i \geq j \end{aligned}$$

Miguel Garcia – STS – TUHH



The big picture on certifying transformation algorithms

We will use +CAL to check whether an algorithm generates well-formed, correctly typed sentences (given well-formed, correctly typed input sentences)

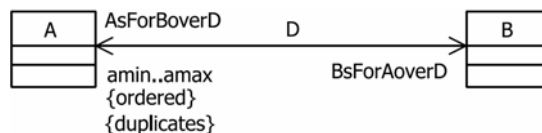
To avoid writing everything by hand we need:

- A way to translate the metamodel of the input language (i.e. an object-oriented class model with OCL invariants) into TLA+
 - Same thing for the output language
 - A way to translate the transformation specification from (visitor, QVT-style transformation, etc.) into +CAL
- By the way, pretty much the same infrastructure applies in case one wants to model check, let's say, BPEL processes. Investing in this know-how pays off!

Miguel Garcia – STS – TUHH



What the metamodel-to-TLA+ translation looks like



for each instance b of B , those instances of A reachable over D from it must in turn have b among their instances reachable over D

$$\begin{aligned} \text{InvariantBidirectionality_AtoB} &\triangleq \\ &\vee \text{BsForAoverD} = \{\} \\ &\vee \forall b \in \text{DOMAIN } \text{BsForAoverD} : \\ &\quad \text{LET } \text{AsReachable} \triangleq \text{AsForAoverD}[b] \text{ IN} \\ &\quad \forall i \in \text{DOMAIN } \text{AsReachable} : \\ &\quad \quad \text{LET } \text{anA} \triangleq \text{AsReachable}[i] \text{ IN} \\ &\quad \quad b \in \text{BsForAoverD}[\text{anA}] \end{aligned}$$

counterpart of the above, this time for each a

$$\begin{aligned} \text{InvariantBidirectionality_BtoA} &\triangleq \\ &\vee \text{BsForAoverD} = \{\} \\ &\vee \forall a \in \text{DOMAIN } \text{BsForAoverD} : \\ &\quad \text{LET } \text{BsReachable} \triangleq \text{BsForAoverD}[a] \text{ IN} \\ &\quad \forall aB \in \text{BsReachable} : \\ &\quad \quad \text{ElemIsInSeq}(a, \text{AsForAoverD}[aB]) \end{aligned}$$

$$\text{InvariantsDirectionality} \triangleq \wedge \text{InvariantBidirectionality_AtoB} \wedge \text{InvariantBidirectionality_BtoA}$$

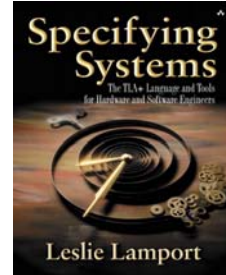
How to process OCL ASTs: A tutorial can be found in this Eclipse Technical Article (still under review)
https://bugs.eclipse.org/bugs/show_bug.cgi?id=167543

Miguel Garcia – STS – TUHH



A word on tooling

- We're working on an Eclipse plugin to translate a metamodel (including OCL) into TLA+, to be released mid 2007
- We aim at translating transformations specified in Eclipse ATL into +CAL
- An open-source Eclipse-based text editor for TLA+ has been authored by another team from our university (Telematics department)
- Leslie Lamport has made the +CAL and TLA+ tools (and documentation) open source:
<http://research.microsoft.com/users/lamport/tla/pluscal.html>



Miguel Garcia – STS – TUHH



Lessons learnt from a sample transformation: Schorr-Waite

This algorithm modifies a graph in-place, by performing pointer reversal (twice) while visiting reachable nodes

Time-dependent state is a tough nut to crack for any verification technique. Even so, +CAL can handle Schorr-Waite without simplifications

Certification of model transformations in the Eclipse ecosystem is already feasible (and will become only easier)

*Using a certified model compiler is like traveling by train.
Using a non-certified one is like hitchhiking.*

Miguel Garcia – STS – TUHH

