On the Formalization of Model-Driven Software Engineering

Miguel Garcia

2009-02-27

http://www.sts.tu-harburg.de/people/mi.garcia
Agenda

– Thesis Statement

– The processing of Abstract Syntax Trees makes the world go round
– Custom languages require custom IDE support

– Certification of language transformations
– Runtime services for new language abstractions

– Insights and outlook
Active X controls are required to use this product. Please enable Active X controls in your Internet Explorer security settings and restart the application.
Imagine a world where …

… running programs could tell us which high-level integrity constraints have just been broken.

… Compilers could output high-level framework extensions, certified to run without glitches on an application container (e.g., the Eclipse IDE).

And where

… The overhead of defining and tooling a new language were minimal, thus fostering innovation.
Thesis Statement

The application of formal methods in
– the specification of programming languages,
– the design of translation algorithms, and
– the design of supporting runtime systems,
improves the productivity and quality of model-driven software engineering.
Language metamodels are here to stay

All major recent languages adopt OO concepts in their definition (BPEL, Scala, WebML, etc.)

Query and Transformation of object graphs is however more difficult to

- optimize for speed
- verify automatically
- scale up

Formal methods to the rescue:

- avoid redundant computation of intermediate results
- handle destructive updates, reduce AST shapes to representatives
- rephrase nested loops
JPQL, cutting-edge state of the practice

Its (informal) spec is a shaky foundation for:
  a) Type checkers for JPQL queries
  b) Checkers of static semantics of JPQL
  c) Authoring tools for (Java) embeddings of JPQL
  d) Robust transformation algorithms into SQL’92

A formal metamodel, presented in

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

already solved a) and b) above, while

Garcia M.
Automating the embedding of Domain Specific Languages in Eclipse JDT
Eclipse Technical Article

solves c). Future work may build upon to address d)
ASTs under the hood

context UpdateItem
inv WFR_4_10_A :

if left.oclIsKindOf(ejb3q1mm::pathExp::StateField)
then
  -- the LHS is typed with SupportedJavaType, the RHS must be compatible with that
  if not rightNewValue.oclIsKindOf(ejb3q1mm::stmts::RHSUpdateItemSupportedJavaType)
  then false
  else let
    t1 : ejb3q1mm::schema::SupportedJavaType = left.oclAsType(ejb3q1mm::pathExp::StateField).type(),
    t2 : ejb3q1mm::schema::SupportedJavaType = rightNewValue.oclAsType(ejb3q1mm::stmts::RHSUpdateItemSupportedJavaType).type()
  in
    ejb3q1mm::schema::SupportedJavaType::areTypeCompatible(t1, t2)
endif

Best practices for OO AST processing

Garcia, M.
How to process OCL Abstract Syntax Trees
Eclipse Technical Article

Detecting implicit constraints in metamodels

Garcia M., Kaplunova A., Möller, R.
Model Generation in Description Logics: What can we learn from Software Engineering?
Technical report, Institute for Software Systems (STS)
“Who will custody the custodians?”

Or, How to type the metamodelling language that is used for typing domain-specific languages?

Type systems show an unbeatable ratio:
unsafe situations detected for a given amount of specification effort

An EMOF implementation (widely used in industry) added parametric polymorphism, but lacked declarative formal rules for type checking

Garcia M.
Rules for Type-checking of Parametric Polymorphism in EMF Generics
First Workshop MDSD Today, co-located with Software Engineering 2007

And the Java implementation was generated “for free” with

Garcia M., Shidqie, A. J.
OCL Compiler for EMF
Eclipse Modeling Symposium, co-located with Eclipse Summit Europe 2007
EMOF + OCL models are so versatile because …

They are shorthands for specifications with a semantic anchoring in logic (first-order logic, relational logic)

It’s not EMOF + OCL as such that is powerful, but the theory and algorithms that can be leveraged:

– model-checking to certify model transformations (+CAL, Alloy)
– logic engines to uncover implicit constraints (Description Logics)
– calculus-based manipulation to optimize expressions (Monoid calculus)
DSL support in IDEs: IDE generation

An IDE-based text editor for a DSL is a prime example of a framework extension.

Does all the theory about model compilers work in practice? It does.

Garcia M., Sentosa P.
Generation of Eclipse-based IDEs for Custom DSLs.
Technical report, Institute for Software Systems (STS)
More DSL support in IDEs: Multiview editing

Solving the multiview synchronization problem at the level of the DSL definition gives a clear semantics to roundtripping behavior.

Garcia M. Bidirectional Synchronization of Multiple Views of Software Models Domain-specific Modeling (DSML'08) Workshop, co-located with Modellierung 2008, Berlin, Germany.
Certification of transformations: Imperative style

Model transformations are algorithms, that
1. take sentences in a high-level, platform-independent language
2. compile them into *framework extensions + configuration scripts*

The minimum we expect of a model transformations is,
   given well-formed, correctly typed input sentences
   Generate
   well-formed, correctly typed output sentences.

The state of the practice does not always guarantee
that minimum, and thus the need for the techniques in

Garcia M., Möller R.
Certification of Transformation Algorithms in Model-Driven Software Development
Software Engineering 2007, Hamburg, Germany
Certification of transformations: Pattern-matching

Input: Transformations expressed in the QVT-Relations language
Output: A spec for the Alloy model-checker to find a counterexample

relation AttributeToColumn
{
  checkonly domain uml c:Class {}
  enforce domain rdbms t:Table {}
  primitive domain prefix: String;
  where {
    PrimitiveAttributeToColumn(c, t, prefix);
    ComplexAttributeToColumn(c, t, prefix);
    SuperAttributeToColumn(c, t, prefix);
  }
}

relation SuperAttributeToColumn
{
  checkonly domain uml c:Class
  {general= sc:Class {}};
  enforce domain rdbms t:Table {}
  primitive domain prefix: String;
  where {
    AttributeToColumn(sc, t, prefix);
  }
}

pred AttributeToColumn[c:umlDomain/Class, t:rdbmsDomain/Table] {
  /* given that the string prefix being received as argument is constantly empty,
  it has been optimized away by applying the constant propagation optimization */
  PrimitiveAttributeToColumn[c, t]
  -- assume for now no ComplexAttributeToColumn[c,t]
  SuperAttributeToColumn[c, t]
  -- no other columns other than supported by the above conditions
  t.tableColumns.originatingClass = { cWithAttr:c.(^general) | some cWithAttr.attributes }
  -- all col:t.tableColumns | col.name in col.originatingClass.attributes.name
}

pred SuperAttributeToColumn[c:umlDomain/Class, t:rdbmsDomain/Table] {
  all superC:c.(^general) | PrimitiveAttributeToColumn[superC, t]
}

Garcia M.
Formalization of QVT-Relations: OCL-based Static Semantics and Alloy-based Validation
MDSD Today 2008, Elmshorn, Germany
Counterexample for output well-formedness of the UML2RDBMS transformation
Runtime services for DSL abstractions

Repeated evaluation of OCL invariants (i.e., boolean queries over an object graph) causes a noticeable performance slowdown.

This problem has two variants:
- The main-memory case
- The secondary-storage case

Garcia M., Möller R.
Incremental evaluation of OCL invariants in the Essential MOF object model
Modellierung 2008, Berlin, Germany.

Garcia M.
Efficient Integrity Checking for Essential MOF + OCL in Software Repositories
Thesis Statement confirmed in practice

The application of formal methods in
– the specification of programming languages,
– the design of translation algorithms, and
– the design of supporting runtime systems,

improves the productivity and quality of model-driven software engineering.
Insights and outlook

Traditional approaches to balancing quality and cost in the software industry have too often resulted in *low quality* and *cost overruns*.

Candidate solutions:
- Ontologies, aka Reference Modeling, to jumpstart development and increase interoperability
- Advanced Program Verification, to more precisely predict runtime behavior
- Software authoring environments, automating specialist know-how, multiplying its benefits

*All of them rely on formal methods, which are instrumental to the success of Model-Driven Software Engineering.*
Insights and outlook

Traditional approaches to balancing quality and cost in the software industry have too often resulted in *low quality* and *cost overruns*.

Candidate solutions:
- Ontologies, aka Reference Modeling, to jumpstart development and increase interoperability
- Advanced Program Verification, to more precisely predict runtime behavior
- Software authoring environments, automating specialist know-how, multiplying its benefits

All of them rely on formal methods, which are instrumental to the success of Model-Driven Software Engineering

(that’s good news for researchers)
In closing: yet more post-PhD papers

Already accepted

Garcia M., Prithiviraj R.
Rethinking the Architecture of O/R Mapping for EMF in terms of LINQ
Eclipse Modeling Symposium at Eclipse Summit Europe 2008

In the pipeline

Garcia, M.
Compiler plugins can handle nested languages: AST-level expansion of LINQ queries for Java. Under submission.

Scaling up the execution of QVT-Relations transformations on large model repositories
In preparation.

A few last words of advice:
There are pastry and cakes already waiting for us
Backup slides
DSL support in IDEs: DSL embedding in Java

Authoring DSL statements with a Progressive API

- Underlying theory: constraint-based configuration
- Compile-time OCL well-formedness rules on ASTs
- Useful in connection with compile-time transformations, for example from LINQ into Java (compiler plugins)
The big picture on certifying transformation algorithms with +CAL

In order to avoid writing everything by hand, we need:

a) A way to translate the metamodel of the input language (i.e. an object-oriented class model with OCL invariants) into TLA+
b) Same thing for the output language
c) 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!
Model checking

- Alloy (http://alloy.mit.edu/) allows declaring “possible worlds” consisting of mathematical relations connecting atomic symbols

- Three kinds of automatic analyses are possible:
  - *Model Finding*, whose output are (visual) depictions of concrete worlds that satisfy the specified constraints
  
  - Assertions can be given, which are claimed to follow from the rest of the spec. *Counterexample Finding* reveals concrete worlds that are conformant save for the broken assertion
  
  - For unsatisfiable predicates, *Unsat Core* can be used to highlight the relevant portions of the Alloy spec that contributed to unsatisfiability
Optimistic Memoization: go ahead and use cached results, correct later if stale

Life and times of a misprediction in a sorted binary tree:

```java
boolean isSorted(Tree t) {
    if (t == null) return true;
    if (t.left != null && t.left.value >= t.value) return false;
    if (t.right != null && t.right.value <= t.value) return false;
    return isSorted(t.left) && isSorted(t.right);
}
```
Related Work: Software Transactional Memory (STM) in EMOF managed runtimes

The idea, intuitively:

```c
void deposit(...) { atomic { ... } }
void withdraw(...) { atomic { ... } }
int balance(...) { atomic { ... } }
void transfer(Acct from, int amt) {
    atomic {
        //correct and parallelism-preserving!
        if(from.balance() >= amt && amt < maxXfer) {
            from.withdraw(amt);
            this.deposit(amt);
        }
    }
}
```

With memoization, we are already tracking updates:

Outlook

- **Where to apply formal machinery next?**
  - OO databases
    - Verification of query optimization
      (a proof of concept project is underway)
  - Programming languages
    - Concurrency analyses
    - OO program verification
    - Software Transactional Memory for Scala
  - Bridging both of them
    - Seamlessly partitionable programs across DBs and main-memory