Model Driven Software Engineering creates tomorrow's legacy

Mark van den Brand

Department of Mathematics and Computer Science
Overview of presentation

• Introduction
• Observations
• Model driven software engineering
• Legacy and model driven software engineering
• Legacy challenges
• Conclusions

Model Driven Software Engineering creates tomorrow's legacy
Introduction

Joint work with

- Önder Babur
- Josh Mengerink
- Ramon Schiffelers
- Alexander Serebrenik
Introduction

1992 - 1997: Assistant professor at UvA (NL)
1997 - 2005: Senior researcher at CWI (NL)
2006 - now: Full professor TU/e (NL):

• Chair of Software Engineering and Technology (SET)
• SET focuses on Model Driven Software Engineering:
  • Domain specific language design
  • Analysis of models, meta-models, and model transformations
  • Modeling of Functional Safety in Automotive Domain
• Industry motivated research

Model Driven Software Engineering creates tomorrow's legacy
Introduction

The research area of the SET group is software engineering, and model-based software engineering in particular.

• Given the high-tech software-intensive industry in the Eindhoven region, we consider time- and cost-efficient development of high-quality software as crucial.
Introduction

SET focuses research-wise on two subareas:

• meta-modeling/domain specific languages including semantics of domain specific languages, language workbenches, and verification of model transformations;

• Data Science applied to software engineering focusing on software evolution of (multi-lingual) systems, including advanced metrics, repository mining, and social aspects of software development to extract “relevant” information of existing software
Introduction

Software Engineering and Technology (SET) group has strong cooperation with High Tech industrial partners via research projects:

• Océ (document handling)
• VanDerLande Industries (luggage handling, warehouses)
• Philips Healthcare (medical equipment: MRI, CT, X-ray, invasive surgery)
• ASML (lithography systems)
• DAF ((long distance heavy) trucks)
Observations

• Software quality research is vital for modern society
• Software is omnipresent, hardly any modern device or equipment is without software
• Software connects people and devices/equipment with ever increasing complexity
• Our society depends on software and improves the quality of living, among others in the following domains:
  • medical
  • automotive
  • domestic
  • social media
Observations

Software has become leading in high-tech equipment:
- without software no production

Increase in the amount of software has raised:
- correctness of the software
- need for efficient software development
- awareness with respect to legacy
Observations

Software effort doubles every new generation

We need to stop this trend, it is not sustainable
Observations

Software evolves, continuous growth in:

- size of software (amount of LOC)
- complexity of software
- features in (software) systems
- costs to build software
- number of languages in software systems
Observations

High tech industry in the Eindhoven region has embraced model driven software development

- To tackle the ever increasing amount of software
  - In-house DSLs are developed, using EMF and MPS, and applied
  - UML and SysML are used for modeling behavior
- To ensure correctness and robustness via (model) checking:
  - ASD (Verum) is intensively used to define interfaces and protocols
- To facilitate opportunities for virtualization (aka digital twin)
- To be afraid to miss the boat
Model driven software engineering

Models are commonly used when designing mechatronic systems

- hardware design
- electronic design
- physical models
- Matlab/Simulink models
- software models

Software has proven to be crucial but at the same time a challenge

- model driven engineering has become very popular
Model driven software engineering

Model driven engineering

• considers models as first class citizens
• increases level of abstraction because of the use of models
• offers the choice between general purpose modeling languages or domain specific languages
  • the first may lead to a vendor lock-in
  • the second may involve a huge investment in language design, implementation, and tooling
Model driven software engineering

Requirements

Product

design

implementation
Model driven software engineering

Requirements

- Specification
  - In terms of problem domain
  - Expressive for concise specification of large multi-disciplinary systems
  - ‘Look-and-feel’ primarily determined by domain experts
  - Crucial for adoption

Product

16 Model Driven Software Engineering creates tomorrow's legacy
Model driven software engineering
Model driven software engineering

Requirements

analysis

design

implementation

Deadlocks
Safety
Performance
Throughput

Product

Model Driven Software Engineering creates tomorrow's legacy
Model driven software engineering

Requirements

- Deadlocks
- Safety
- Performance
- Throughput

Analysis

Design

Synthesis

- Property preserving
- Automated

Implementation

Product

Model Driven Software Engineering creates tomorrow's legacy
Model driven software engineering

(Domain) languages

Models

20  Model Driven Software Engineering creates tomorrow's legacy
Model driven software engineering

Domain models (specification)

Aspect models (analysis)

Transformations

Model Driven Software Engineering creates tomorrow's legacy
Model driven software engineering

A few (research/engineering) challenges:

• Identification of common semantic concepts in a certain domain:
  • High-Tech Industry
    • real-time, state machines, supervisory control, material flow (paper, wafers), etc.
  • Capturing these concepts in domain specific languages
• Quality and correctness of model transformations wrt
  • property preservation
  • underlying semantics
• Modularity of meta-models and composition semantic building blocks
• Evolution of meta-models and co-evolution of models
• Mixing multiple domain specific languages
Model driven software engineering

What is a Domain Specific Language (DSL)?

- A DSL is a formal, procesable language targeting at a specific aspect of a system
- Its semantics, flexibility and notation is designed in order to support working with that aspect as efficiently as possible
- “A language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain”
Model driven software engineering

DSL meta-model

code generation
model transformation

Execution platform
(C, Java code)
Model driven software engineering

Model transformations and code generation
Model driven software engineering

Domain specific languages offer

• Reduction of development time via increase of abstraction
• Increase of robustness via verification of models and model transformations

However,

• efficient design of domain specific languages and corresponding tooling has to be implemented and maintained as well
Model driven software engineering

Modeling in practice

• Used across many disciplines
• Abstraction mechanism, used for
  • communication, guidance
  • sketch, blueprint, prototype
  • analysis, verification, optimization
  • automated production
Model driven software engineering

Models
• UML
Model driven software engineering

Models
- BPM
Model driven software engineering

Models
• Simulink
Legacy and model driven software engineering

- Number of modeling languages, including Domain Specific Languages, is growing
- Number of models created via these modeling languages is also growing
- What are the consequences? But first some figures!
Legacy and model driven software engineering

Expanding the universe of modeling

• Widespread adoption of modeling, industrial cases
  ➔ Too many modeling artifacts (models, transformations, …)
  ➔ Possibly heterogeneous as well (software + hardware, software + business, …)
  ➔ Impossible to manage them ad-hoc
Legacy and model driven software engineering

Expanding the universe of modeling

• Widespread adoption of modeling, industrial cases
  → Too many modeling artifacts (models, transformations, ...)

• Examples (software domain)
  • Repositories
    • ATL Zoo: ~300 meta-models
    • SPLOT: >900 feature models, growing
    • GitHub Ecore crawl: ~7k meta-models
    • Lindholmen UML dataset: ~90k models!
Legacy and model driven software engineering

Expanding the universe of modeling

- Ecore meta-models in GitHub

---

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Domain clustering analysis results on the ATL Meta-model Zoo

Number = index of meta-model
Height = distance between meta-models
Legacy and model driven software engineering

Domain clustering analysis results on the ATL Meta-model Zoo

Petri net

State machine

Conference management

Bibliography

Word

Excel
Legacy and model driven software engineering

Expanding the universe of modeling

• Widespread adoption of modeling, industrial cases
  → Too many modeling artifacts (models, transformations,...)

• Examples (software domain)
  • Industrial case 1: just one of the DSL eco-systems of an OEM-er:
    • >20 DSLs
    • >40 meta-models
    • >90 QVTo files, >90k LOC, >600 rules/assistants
    • >5000 models

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Expanding the universe of modeling

• Widespread adoption of modeling, industrial cases
  → Too many modeling artifacts (models, transformations, ...)

• Examples (software domain)
  • Industrial case 2: 6 projects (~eco-systems) of a supplier, some figures:
    • 19 DSLs, 43 meta-models
    • 1670 models
    • QVTo: 107 files, 34k LOC, 80 transformations, 979 mappings, 1872 helper/queries
    • Xpand/Xtend: 368 files, 67k LOC, 6 transformations, 501 templates, 2004 queries
    • Acceleo: 251 files, 41k LOC, 1298 templates, 1377 queries
Legacy and model driven software engineering

Expanding the universe of modeling

• Widespread adoption of modeling, industrial cases
  → Too many modeling artifacts (models, transformations, ...)

• Examples (software domain)
  • What about evolution?
    • Industrial case 1: > 50K artifacts
    • Industrial case 2: 10K artifacts
  • What kind of evolution?
Legacy and model driven software engineering

**Statistical Analysis of MOdelS (SAMOS)**

- SAMOS is developed by Önder Babur
- SAMOS is capable of:
  - feature extraction (fragmentation)
    - n-grams, subtrees, metrics, ...
  - feature comparison
  - natural language processing
  - elaborate weight and comparison schemes
  - Vector Space Model computation
  - distance measures, statistical analyses in R
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Cross-DSL analysis: ~100 meta-models for 4 DSL eco systems
Legacy and model driven software engineering

Cross-DSL analysis: ~100 meta-models for 4 DSL eco systems

Conceptual overview?
Legacy and model driven software engineering

Cross-DSL analysis: ~100 meta-models for 4 DSL eco systems

Copy-pasted at some point + evolved in time?
Legacy and model driven software engineering

An OEM-er in the Eindhoven region has

- 22 DSLs built using EMF + OCL
- 95 model transformations
- 5500 models created using the meta-models
Legacy and model driven software engineering
Legacy and model driven software engineering

What are legacy systems?

• Systems developed for a specific client that have been in service for a long-time
• Many of these systems were developed years ago using obsolete technologies
• They are likely to be business critical systems required for normal operation of a business
Legacy and model driven software engineering

DSLs evolve

±5000 of models and growing

#models per DSL
(logarithmic scale)
Legacy and model driven software engineering

Manually maintaining models in response to DSL evolution is NOT feasible!

#models per DSL
(logarithmic scale)
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Obtain Evolution Specification
Evolve
Derive Co-evolution Specification
Co-evolve

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

1. **Obtain**
2. **Evolve**
3. **Derive**
4. **Co-evolve**

**Transaction Diagram**

- **Legacy and Model Driven Software Engineering (MDSSE)** creates tomorrow's legacy.
Legacy and model driven software engineering

Co-evolution problem

Model version 1  evolution  DSL version 2

Model version 1  co-evolution  Model version 2

DSL version 1  DSL version 2

Model version 1

change  conformance
Legacy and model driven software engineering

Models evolve but meta-models evolve as well:
• Abstract syntax
• Constraints (static semantics)
• Semantics
• Semantic domain
Legacy and model driven software engineering

Evolution on all levels

Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

Syntax only

Semantics only

Multi level

Which of these evolution patterns occur in practice?
Legacy and model driven software engineering

Which of these evolution patterns occur in practice?

• Industrial case study
  • 22 DSLs
  • 95 model-to-model transformations
  • >5500 models

• Look at subsequent versions in repository
### Legacy and model driven software engineering

<table>
<thead>
<tr>
<th>Code</th>
<th>Syntax evolves</th>
<th>Semantics evolves</th>
<th>Semantic domain evolves</th>
</tr>
</thead>
<tbody>
<tr>
<td>E000</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>E001</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>E010</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>E100</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>E011</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>E101</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>E110</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>E111</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Diagram:**

- **DSL Ecosystem Version 1**
  - Metamodel X version 1 ($M_1$)
  - XtoY version 1 ($S_1$)
  - SD evolution

- **DSL Ecosystem Version 2**
  - Metamodel X version 1 ($M_2$)
  - XtoY version 2 ($S_2$)
  - SD evolution

---

62  Model Driven Software Engineering creates tomorrow's legacy
Legacy and model driven software engineering

E011, example:

- Take snapshots at time $t=1$ and $t=2$
- For every possible triple $(X : MM, Y : M2M-trans, Z : MM)$
- Does it hold that:
  - $Y : X \rightarrow Z$
  - $X$ does not evolve from $t=1$ to $t=2$
  - $Y$ evolves from $t=1$ to $t=2$
  - $Z$ evolves from $t=1$ to $t=2$
# Legacy and model driven software engineering

<table>
<thead>
<tr>
<th>Code</th>
<th>Syntax evolves</th>
<th>Semantics evolves</th>
<th>Semantic domain evolves</th>
<th># occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>E001</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>344</td>
</tr>
<tr>
<td>E010</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>865</td>
</tr>
<tr>
<td>E100</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>84</td>
</tr>
<tr>
<td>E011</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>368</td>
</tr>
<tr>
<td>E101</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>E110</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>86</td>
</tr>
<tr>
<td>E111</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>292</td>
</tr>
</tbody>
</table>
## Legacy and model driven software engineering

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th># occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>E010</td>
<td>Only semantic change</td>
<td>865</td>
</tr>
<tr>
<td>E011</td>
<td>Only syntax change</td>
<td>368</td>
</tr>
<tr>
<td>E001</td>
<td>Only semantic domain change</td>
<td>344</td>
</tr>
<tr>
<td>E111</td>
<td>Full evolution</td>
<td>292</td>
</tr>
<tr>
<td>E110</td>
<td>Different syntax with different semantics expressed in the same semantic domain</td>
<td>86</td>
</tr>
<tr>
<td>E100</td>
<td>Same syntax gets new semantics in a different semantic domain</td>
<td>84</td>
</tr>
<tr>
<td>E101</td>
<td>Syntax and semantics domain are changed, but no new semantics is provided</td>
<td>0</td>
</tr>
</tbody>
</table>
## Legacy and model driven software engineering

### Code | Description | # occurrences
--- | --- | ---
E010 | Only semantic change | 865
E011 | Only syntax change | 368
E001 | Only semantic domain change | 344
E111 | Full evolution | 292
E110 | Different syntax with different semantics expressed in the same semantic domain | 86
E100 | Same syntax gets new semantics in a different semantic domain | 84
E101 | Syntax and semantics domain are changed, but no new semantics is provided | 0

### 77%
Legacy challenges

The following challenges can be observed:

• The tooling to create and use DSLs is far from mature.
• The creation of a DSL involves understanding of the domain for which the languages are created.
• The increased level of abstraction and introduction of domain concepts makes the models harder to understand and maintain.
• The interactions between software models and models from other (system) engineering domains, e.g. describing physical behavior, are becoming more and more important.
Legacy challenges

The tooling to create and use DSLs is far from mature

• unstable
• badly documented
• deprecates rapidly

Industry uses

• standard tools (QVTo) with large user communities
• tools developed and maintained by companies, e.g. MPS, Sirius, Acceleo
Legacy challenges

The creation of a DSL involves understanding of the domain for which the languages are created

- having the capability of translating this knowledge to concepts at the right level of abstraction.

Industry applies different DSL development strategies:

- outsourcing to research institutes.
- team of software language engineers combined with domain experts
- in house prototyping and transfer to supplier for maturing and maintenance

Model Driven Software Engineering creates tomorrow's legacy
Legacy challenges

The increased level of abstraction and introduction of domain concepts makes the models harder to understand and maintain.

- The use of DSLs involves also a risk, if the developer(s) of a DSL leaves the company then the maintenance of the DSL may be jeopardized.
- The number of developers that are able to understand and maintain DSLs is low; the number of developers understanding general purpose languages, e.g. C, will always be higher.

Industry asks for courses on modeling and meta-modeling
Legacy challenges

The interactions between software models and models from other (system) engineering domains has become important.

• For describing physical behavior
• For enabling virtualization
• For facilitating system engineering, moving from mono-disciplinary to multi-disciplinary modeling
Conclusions

• Research on model driven software engineering should
  • move away from just focusing on tooling
  • start focusing on proper methodologies to extract domain concepts in order to create usable DSLs.
  • deal with evolutionary aspects of DSLs and created models
  • work on stabilizing the tooling needed to create languages and corresponding models, ensuring consistency
    • between languages and between models, and
    • between languages and models.

If we are able to make this happen then we might have a silver bullet after all and the promised increase in quality and productivity will be realized.