Model-supported Software Creation: Towards Holistic Model-driven Software Engineering

The 2023 IARIA Annual Congress on Frontiers in Science, Technology, Services, and Applications

15 Nov 2023, Valencia, Spain, Hans-Werner Sehring
Agenda

01 Model-driven Software Engineering (MDSE)
03 Model-Supported Software Creation (MSSC)
05 An MSSC Approach with the M³L
02 MDSE in Practice
04 The Minimalistic Meta Modeling Language (M³L)
06 Summary and Outlook
15.11.2023
1. Model-driven Software Engineering

Various approaches to model-driven software engineering exist, for example,

- **Model-driven Architecture (MDA)**
  - Early MDSE approach
  - Models are created on (originally) three levels of abstraction
  - A *Computation-Independent Model (CIM)* from the perspective of the subject domain.
  - A *Platform-Independent Model (PIM)* as a first formal model.
  - Transformed into a *Platform-Specific Model (PSM)* used to generate a working implementation.

- **Software Generation**
  - Model contained in code
  - Different approaches, e.g., metaprogramming, templates, generative AI

- **Domain-specific Languages (DSLs)**
  - Languages \(\triangleq\) Metamodels
  - Defined for a specific domain

- **Generic Software**
  - A domain model was used during the development of the software
  - If the model is parameterized, then software is configurable (low code / no code development)
2. MDSE in Practice

Approaches based on **formal** models and model transformations

Software engineering **reality** (at least in some domains):

Depending on the kind of software and the kind of project, we find

- Heterogeneous modeling artifacts: varying **degrees of formalism, ambiguity, detail**, etc.
- Artifacts often part of a **methodology or a tool**: notation and representations matter
- Several project stages, **not only software** (engineering) related; from inception to operations stages
## Sample Development Artifacts and Formalizability

<table>
<thead>
<tr>
<th>Phase</th>
<th>Order</th>
<th>Discipline</th>
<th>Artifact</th>
<th>Formal(izable) model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inception / Research</strong></td>
<td></td>
<td>Management</td>
<td>Goals</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Management</td>
<td>Inception</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Requirements (inside-out)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Research (outside-in)</td>
<td>–</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
<td>←</td>
<td>Concept</td>
<td>Personas</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Customer journeys</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Existing tools</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Information demand / data flows</td>
<td>X</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td>←</td>
<td>Concept</td>
<td>Information architectures (stationary web, mobile web, mobile app)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Graphics</td>
<td>Wireframes (stationary web, mobile web, mobile app)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Solution architecture</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Graphics</td>
<td>UI design / style guide</td>
<td>(X)</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>←</td>
<td>Technology</td>
<td>SW arch (if not agile)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>System arch (if not agile)</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Code design</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Code</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Test cases</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Test scripts</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Documentation</td>
<td>(X)</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>←</td>
<td>Technology</td>
<td>Infrastructure</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Technology</td>
<td>Build and deploy scripts</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>Concept</td>
<td>Training</td>
<td>–</td>
</tr>
</tbody>
</table>
Support for Informal Processes and Artifacts

Given that the various process steps and artifacts that are
• not formal
• ambiguous
• not producible by model transformations
• etc.
we cannot have MDSE.

Still, we want ...
• Support in managing (modeling) artifacts
• Checks on models
• Deriving software from specifications
• Traceability
• Etc.

We want the benefits of MDSE.
3. Holistic Model-Supported Software Creation

For those software projects with imprecise, creative development steps, we need ...

**Holistic** MDSE that covers all project stages

For example: project success is measured based on business goals, not requirements

Model-supported SE acknowledges the fact that we cannot purely rely on formal models and model transformations

In the absence of formal models, these cannot be the overarching communication base

Model-supported Software Creation acknowledges the creative work that is part of the process

There is creative work on artifacts that cannot adequately be formalized by model transformations

Need to **model activities and artifacts outside SW production**

Models can describe the (final) informal artifacts

But: **model transformations to describe development steps**
## Modeling Stages and Artifacts

<table>
<thead>
<tr>
<th>Creation stage</th>
<th>Sample model entities on the stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Business) Goals</td>
<td>KPIs</td>
</tr>
<tr>
<td></td>
<td>OKRs</td>
</tr>
<tr>
<td>Subject domain model</td>
<td>Information architecture</td>
</tr>
<tr>
<td></td>
<td>Interaction design</td>
</tr>
<tr>
<td></td>
<td>Wireframes</td>
</tr>
<tr>
<td></td>
<td>Processes, data flows</td>
</tr>
<tr>
<td>Requirements, Conceptualization</td>
<td>Solution hypothesis</td>
</tr>
<tr>
<td></td>
<td>Functional ~</td>
</tr>
<tr>
<td></td>
<td>Non-functional ~</td>
</tr>
<tr>
<td></td>
<td>Customer journeys</td>
</tr>
<tr>
<td></td>
<td>Touch points</td>
</tr>
<tr>
<td>Solution architecture</td>
<td>Interfaces</td>
</tr>
<tr>
<td></td>
<td>High-level architecture</td>
</tr>
<tr>
<td></td>
<td>Functional mapping</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Creation stage</th>
<th>Sample model entities on the stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software architecture(s)</td>
<td>Components</td>
</tr>
<tr>
<td></td>
<td>Communication between those components</td>
</tr>
<tr>
<td></td>
<td>Interfaces to the environment</td>
</tr>
<tr>
<td></td>
<td>Constraints of the resulting software system</td>
</tr>
<tr>
<td></td>
<td>Requirements met by the architecture</td>
</tr>
<tr>
<td></td>
<td>Rationale behind architecture decisions</td>
</tr>
<tr>
<td>Code</td>
<td>Metaprograms</td>
</tr>
<tr>
<td></td>
<td>Input for software generators</td>
</tr>
<tr>
<td></td>
<td>Domain-specific language expressions</td>
</tr>
<tr>
<td>Systems architecture</td>
<td>Infrastructure definition (IaC)</td>
</tr>
<tr>
<td></td>
<td>Automated deployments (CI/CD)</td>
</tr>
<tr>
<td>Operations</td>
<td>Service level agreement</td>
</tr>
<tr>
<td></td>
<td>Monitoring</td>
</tr>
</tbody>
</table>

15.11.2023
Examples of Description Models for Informal Artifacts

CIM

(if we use MDA terminology here)

Personas

Persona A

Persona B

Customer Journey 1

Touchpoint 1.1

Touchpoint 1.2

Example Diagram:

- Personas
  - Persona A
  - Persona B
- Customer Journey 1
  - Touchpoint 1.1
  - Touchpoint 1.2

Diagram Elements:

- User Profiles
- Touchpoints
- Customer Journey

(model-based approach)

(if we use MDA terminology here)
The general theme of model transformations we consider:

- **Models on one layer are refined** until the result of the corresponding phase.
- **Models on a subsequent layer are created** from models of previous stages.
Model Refinement and Transformations

Typical phases of a software creation process and model transformations connecting them.
## 4. A Brief Introduction to the M³L

**Basic language constructs. More complete descriptions can be found in the literature.**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The declaration of or reference to a <strong>concept</strong> named A</td>
</tr>
<tr>
<td>A is a B</td>
<td>The <strong>refinement</strong> of a concept B to a concept A;</td>
</tr>
<tr>
<td>A is the B</td>
<td>A is a specialization of B, B is a generalization of A (<strong>the</strong>: A is the only specialization of B)</td>
</tr>
</tbody>
</table>
| A is a B { C } | Containment of concepts;  
C belongs to the **content** of A, A is the **context** of C |
| A |= D | The **semantic rule** of a concept of a concept A;  
whenever A is referenced, D is bound;  
if D does not exist, it is created in the same context as A |
| A | - E F G. | The **syntactic rule** of a concept A;  
A is printed out as or recognized from the concatenation of the syntactic forms of concepts E, F, and G;  
if not defined, a concept evaluates to / is recognized from its name |
The $M^3L$ has an operational semantics for expression evaluation. It is based on (any combinations of) 

- Refinement
- Semantic rules
- **Visibility** rules
  - All concepts in the content of a concept are also visible in the content of refinements: $A \{ B \}, C$ is an $A \Rightarrow C \{ B \}$
  - All concepts in the content of a concept are also visible in the contents of concepts in the context of that concept:
    $$D \{ E \{ F \}} \Rightarrow E \{ F \{ D \}}$$
- **Narrowing**
  If a concept $A$ has a subconcept $B$, and if all concepts defined in the context of $B$ are equally defined in the context of $A$, then each occurrence of $A$ is narrowed down to $B$. 

Person { 
  Name is a String } 
PersonMary is a Person { 
  Mary is the Name } 
PersonPeter is a Person { 
  Peter is the Name 
  42 is the Age } 

$\Rightarrow$ PersonMary 

Person { 
  Peter is the Name 
  42 is the Age } 
$\Rightarrow$ PersonPeter 

Person { 
  Mary is the Name 
  42 is the Age } 
$\Rightarrow$ Person { 
  Mary is the Name 
  42 is the Age }
Definition of a conditional statement

Boolean
True is a Boolean
False is a Boolean

Statement
PrintStatement { Text is a String }

IfThenElse is a Statement {
    Condition is a Boolean
    IfStatement is a Statement
    ElseStatement is a Statement
}

IfTrue is an IfThenElseStatement {
    True is the Condition
} |= TrueStatement

IfFalse is an IfThenElseStatement {
    False is the Condition
} |= ElseStatement

Application in a program

SomeCondition is a ComputeSomeBoolean { ... }

Conditional1 is an IfThenElse {
    SomeCondition is the Condition
    PrintStatement is the IfStatement {
        "It's true" is the Text
    }
    PrintStatement is the ElseStatement {
        "It's false" is the Text
    }
}
M³L concepts represent **different modeling components**

- Topmost concepts represent modeling stages and models
- They contain concepts that represent domain entities
- They relate models and model items to each other

These contained concepts

- May be **stand-alone concepts as model items** for domain entities or
- May represent **artifacts that represent such domain entities**

**Model transformations** trace the evolution of artifacts created during the course of software creation

Model transformations as considered here can be expressing by the M³L
Dimensions of Model Relationships: Combining Models

In the M³L, concepts are defined in context. Base definitions can be “imported” from foreign contexts. This way, models on one layer can be defined by selecting model components of a previous layer as a basis.

For example, in an e-commerce application DomainModel there may be a definition based on commerce base models

ProductDescriptions is a DomainModel {
  ProductData
  PaymentMethods from Commerce
  PackagingInformation from Logistics
}

As an example from another layer, there may be an abstract model of an information system defined as

OurInformationSystem is a PlatformIndependentModel {
  AppServer from SWComponents
  DBMS from SWComponents
  DataSchema from DBModeling
  WebServer from SWComponents
  WebPage from WebDesign
}
Dimensions of Model Relationships: Refining Models

On one layer, models are refined.

In the M³L, model refinement happens along the different axes of M³L models

• by introducing a refined concept of an existing model concept:
  \textit{OurInformationSystem} \rightarrow \textit{OurInformationSystemConcept}

• by refining base concepts of a concept: \textit{WebServer} is a \textit{ServletEngine}

• by refining the content of a model concept: \textit{ProductDataSchema} is the \textit{DataSchema}

Example:

\textit{OurInformationSystemConcept} is an \textit{OurInformationSystem} {
  \begin{itemize}
    \item \textit{RDBMS} from \textit{SWComponents} is the \textit{DBMS}
    \item \textit{ProductDataSchema} is an \textit{RDBSchema} from \textit{DBModeling}, the \textit{DataSchema}
    \item \textit{WebServer} is a \textit{ServletEngine} from \textit{Java}
  \end{itemize}
}
Semantic rules can be used to

- Evaluate concepts
- Assign (operational) semantics to concepts
- Create models in a subsequent stage

Example for the creation of a model on a subsequent stage:
From the software design model of the information system, OurInformationSystemConcept, we have a more concrete model of the data layer, OurInformationSystemDataLayer, derived by

\[
\text{OurInformationSystemConcept} \mid= \text{OurInformationSystemDataLayer} \{
\begin{array}{l}
\text{RDBMS} \\
\text{ProductDataSchema} \\
\quad \text{ProductsTable is a Table from DBModeling}
\end{array}
\}
\]

Dimensions of Model Relationships: Creating Models
On one modeling stage, concept refinements are used to elaborate models

ModelOnStage1 {
    DomainConcept { Attribute }
    MoreElaboratedDomainConcept is the DomainConcept {
        Attribute is an AttributeClass }
    EvenMoreElaboratedDomainConcept is the MoreElaboratedDomainConcept {
        SpecificValue is the Attribute
        AnotherAttribute is an AnotherAttributeClass }
}

By M³L's contextual definitions and refinements

• All **intermediate modeling** steps are accessible (DomainConcept, MoreElaboratedDomainConcept)
• The **cumulated model** is available in the most specific context (EvenMoreElaboratedDomainConcept)
Model Transformation with the $M^3L$ 2

By equipping the more refined concepts of one stage with semantic rules, models in a subsequent stage are initially created

\[
\text{ModelOnStage1} \{
    \text{EvenMoreElaboratedDomainConcept} \, \models \, \text{ModelOnStage2} \{
        \text{Stage2Concept} \{ \ldots \}
    
    \}
}\, \models \, \text{ModelOnStage2}
\]

Additionally, concepts can refer to concepts of a preceding modeling stage

\[
\text{ModelOnStage2} \{
    \text{DomainConceptSpecification} \, \text{is the DomainConcept from ModelOnStage1} \{ \ldots \}
\}
\]
Software Generation from Models 1

In the case of source code generation, software is generated by the M³L using its syntactic rules.

Example (assume that ProductsTable has content Columns):

```plaintext
OurInformationSystemDBImplementationSQLOutput is an SQL {
  OurInformationSystemDBImplementation is an OurInformationSystemDataLayer {
    ProductDataSchema {
      ProductsTable |- "PRODUCTS(" Columns ")" .
    } |- "CREATE TABLE " ProductsTable .
  }
}
```
With contextual definition of syntactic rules, different output formats can be defined on one model.

Example: generate an external format matching the database schema

```
OurInformationSystemDBImplementationJSONOutput is a JSONSchema {
    OurInformationSystemDBImplementation is an OurInformationSystemDataLayer {
        ProductDataSchema {
            ProductsTable | - " \"title\": \"Product\","
            " \"description\": \"product description\","
            " \"type\": \"object\","
            " \"properties\": {" Columns "} .
            } ) | - "{" 
            " \"$schema\": "https://json-schema.org/draft/2020-12/schema\","
            " \"$id\": "https://example.com/product.schema.json\"
            ProductsTable
            "}" .
            }
        }
    }
```
6. Summary and Outlook

Summary

- Software projects consist of more activities than the software production itself – we need **holistic** processes
- There is a class of software projects that includes activities that lead to the creation of unstructured/informal artifacts; these activities are more creative than they are engineering tasks
- For such projects, a model-driven approach that is based on formal models is not possible
- To benefit from the advantages of model-driven development, models shall **support** the process, though

Outlook

- The **references to artifacts** need to be elaborated; we can build on previous work at this point
- Investigate the utilization of generated models as **checklists** that describe the required artifacts
- Above the topic of this paper, the general modeling with the **M³L in MDSE** will be investigated further. For example, can it additionally be used as a reasoner or combined with one?