Visual Artifacts in Software Engineering Processes

The Sixteenth International Conference on Creative Content Technologies (CONTENT 2024)

17 April 2024, Venice, Italy,

Hans-Werner Sehring, NORDAKADEMIE, Germany
sehring@nordakademie.de

IARIA

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Agenda

01 MDSE and Creative SW Dev. Artifacts

02 A Brief Introduction into the M³L

03 Descriptive Models

04 Artifact Creation from Models

05 Prototype Creation from Models

06 Summary and Outlook

Section 1

MDSE and Creative Software Development Artifacts

Model-driven Software Engineering (MDSE)

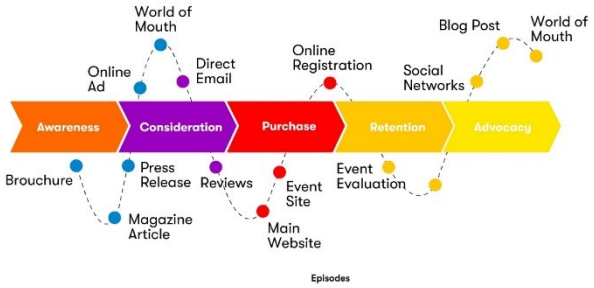
Various approaches to model-driven software engineering exist, for example,

- **Model-driven Architecture (MDA)**
- **Software Generation**
- **Domain-specific Languages (DSLs)**
- **Generic Software**

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

Approaches

First research of two approaches for the combination of MDSE processes and creative software development artifacts:

1) Model elements that **describe visual artifacts**
2) Visual artifacts that are **created from specifications** given in model elements

For experiments with both approaches, we use the Minimalistic Meta Modeling Language (M³L) as a testbed for experimentation.

Section 2

A Brief Introduction into the M³L

M³L at a Glance

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

`A`	The declaration of or reference to a **concept** named `A`
`A is a B` `A is the B`	The **refinement** of a concept `B` to a concept `A`; `A` is a specialization of `B`, `B` is a generalization of `A` (`the`: `A` is the only specialization of `B`)
`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

M³L Concept Narrowing and Implicit Subconcepts

```
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 }

Person {
  Peter is the Name
  42 is the Age }
⇒ PersonPeter

Person {
  Mary is the Name
  42 is the Age }
⇒ Person {
    Mary is the Name
    42 is the Age }
  is a PersonMary
```

Concepts are analyzed after creation to detect certain constallations

- **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**.

- **Implicit Subconcepts**

  If a concept **A** has all base concepts that a concept **B** has, and if for every content of **A** there is a matching content of **B**, then **A** is a base concept of **B**.

M³L Concept Evaluation

```
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 }

Person {
  Peter is the Name
  42 is the Age }
⇒ PersonPeter

Person {
  Mary is the Name
  42 is the Age }
⇒ Person {
    Mary is the Name
    42 is the Age }
```

The M³L has an operational semantics for **concept evaluation**

It is based on (any combinations of)

- Refinement, including implicit refinements
- 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 ⇒ 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 } ⇒ E { F { D } }`

Section 3

Descriptive Models

Examples of Description Models for Informal Artifacts

CIM

(if we use MDA terminology here)

Personas

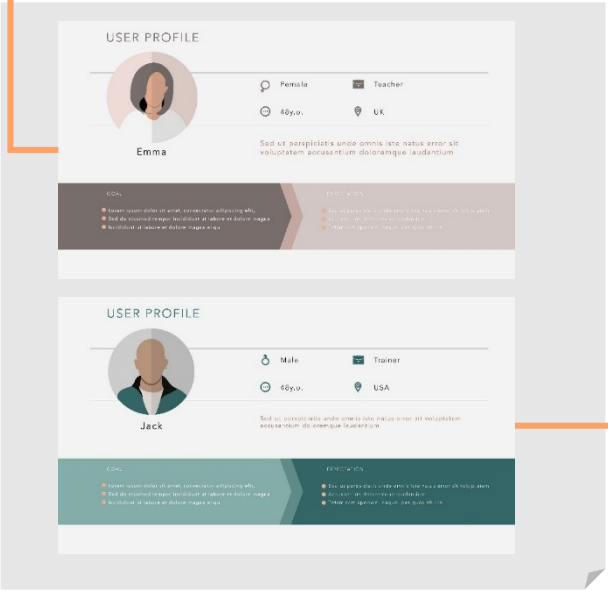
Persona A

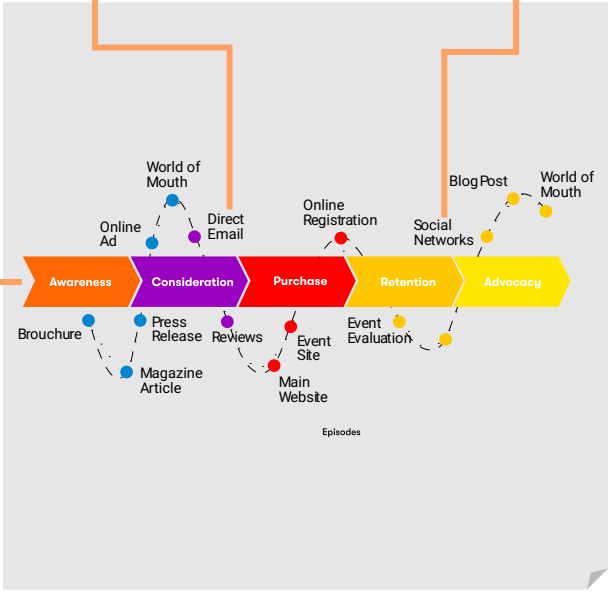
Persona B

Customer Journey 1

Touchpoint 1.1

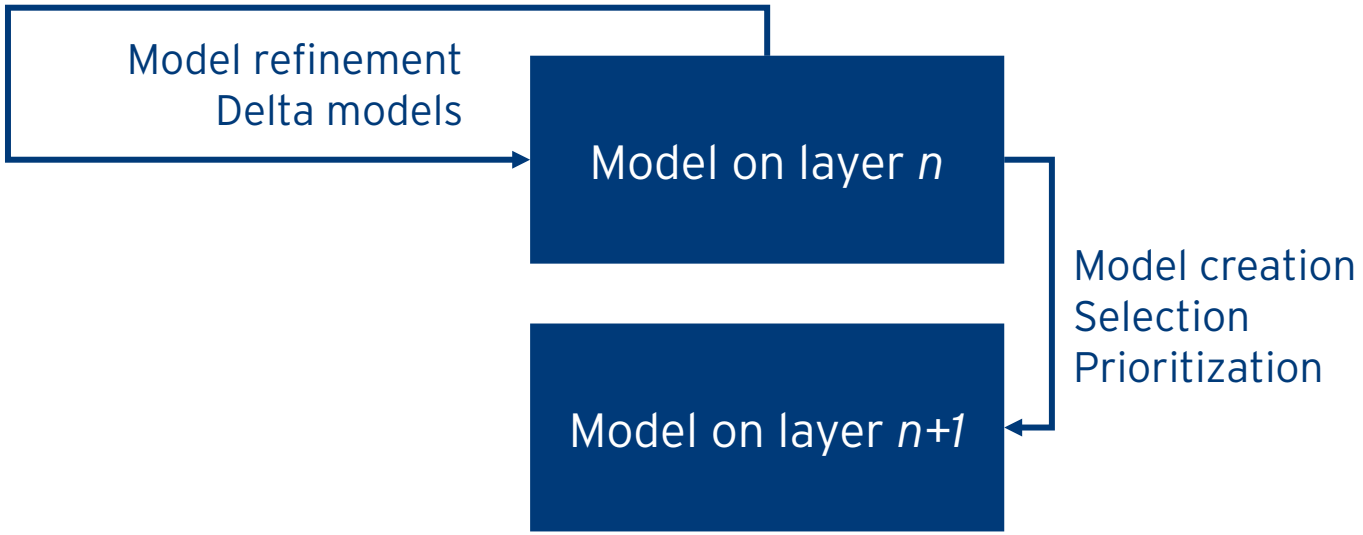
Touchpoint 1.2





Model Refinement and Transformations

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 Transformation with the M³L (1)

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 M³L (2)

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

```
ModelOnStage1 {
  EvenMoreElaboratedDomainConcept |= ModelOnStage2 {
    Stage2Concept { … }
  }
} |= ModelOnStage2
```

Definition on newly created model stage 2

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

```
ModelOnStage2 {
  DomainConceptSpecification is the DomainConcept from ModelOnStage1 { … }
}
```

Section 4

Artifact Creation from Models

Content Management for SW Engineering Artifacts

The content of software engineering documents can be managed by a CMS

For an example of structured content, with a content model like:

```
UserStory is a Content {
  Title       is a String
  Text        is a FormattedString
  StoryPoints is a FibonacciNumber }
```

according content can be created:

```
UserStory123 is a UserStory {
  "As a sales person I …" is the Title
  "When a sales person …" is the Text
  8                       is the StoryPoints }
```

A Model of Software Engineering Artifacts

Based on such content, **models of documents** / artifacts can be defined.
For a start, in a media agnostic manner.

The structure/navigation of documents can also be define in an abstract way.

Refinements for specific media types can capture the specifics of certain output channels.

For example, when creating text documents, there might be definitions like:

```
PublishedDocPart { Content }

UserStoryDetailPage is a PublishedDocPart {
  Content is a UserStory }

WebPage is a PublishedDocPart {
  Title is a String }

PrintDocumentPage is a PublishedDocPart {
  PageNumber is a Number }
```

Media- and Application-Specific Definitions

Refinements of such document models define how actual documents are created.

Example: the HTML format of a webpage

```
HTML {
  <html> </html> <head> </head>
  <title> </title> <body> </body> …
  WebPage
  |- <html>
       <head>
         <title> Title </title>
       </head>
       <body> Content </body>
     </html> . }
```

Based on such media-specific implementations of document models, application-specific documents can be defined.

In the example of user story descriptions:

```
UserStoryHTML is an HTML {
  UserStoryDetailPage {
    Title is the Title from Content
    Content {
      Title       |- <h1> Title </h1> .
      Text        |- <p> Text </p> .
      StoryPoints |- "SP: " StoryPoints .
    } |- Title StoryPoints Text . } }
```

Document/Artifact Generation

With such

- media-specific implementations and
- application-specific layouts,

content can be associated with document models for publication.

Example:

```
MyUserStoryPage is a UserStoryDetailPage {
  UserStory123 is the Content }
```

Section 5

Prototype Creation from Models

Interactive Software Engineering Artifacts (1)

In addition to consumable documents, like text documents, **“live” documents** are used in creative software development; for example; click dummies of Uis.

There may be given definitions of typical UI elements:

```
UIElement { Id }

VisibleUIElement is a UIElement {
  Dimension is a …
  … }

Container is a UIElement {
  Components is a UIElement }

TextField is a VisibleUIElement {
  Text is a String }
```

Interactive Software Engineering Artifacts (2)

Adding dynamic behavior requires some technical definitions:

```
ActivatableUIElement is a UIElement { ActionHandler }

Button is a VisibleUIElement, an ActivatableUIElement { Label is a String
                                                        Icon  is an Image }

ActionHandler { Condition is a Boolean
                Effect }

ExecutedActionHandler { True is the Condition } |= Effect

UIEvent { Target is an ActivatableUIElement }
|= ActionHandler from Target { True is the Condition }
|- "{" "\"target\"" ":" "\""Id from Target"\"" "}"

ClickEvent is a UIEvent
```

Interactive Software Engineering Artifacts (3)

We such base definitions, a concrete UI can be defined

```
SearchDialog is a UIModel {
 SearchView is a Panel {
  1 is the Id
  QueryField is a TextField
  SearchButton is a Button {
   2 is the Id
   Search is the Label }
  …
} } |- SearchView
```

View states and state transitions are provided for a “living style guide”

```
SearchDialogInitial is a SearchDialog {
 SearchView {
  SearchButton {
   Action1 is an ActionHandler {
    "manyres" is the Text from QueryField
    SearchDialogManyResults is the Effect }
   Action2 is an ActionHandler {
    "fewres" is the Text from QueryField
    SearchDialogFewResults is the Effect }
} } }

SearchDialogManyResults is a SearchDialog { ... }

SearchDialogFewResults is a SearchDialog { ... }
```

Section 6

Summary and Outlook

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; there 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
- Concrete formats of **typical creative tools** need to be reviewed to derive required syntactic rule formats
- Investigate the utilization of generated models as **checklists** that describe the required artifacts

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

NORDAKADEMIE gAG Hochschule der Wirtschaft
Köllner Chaussee 11 · 25337 Elmshorn · Tel.: +49 (0) 4121 4090-0 · E-Mail: info@nordakademie.de · Web: www.nordakademie.de