Model-driven Software Development in Practice

Lecture TU Darmstadt
Dr. Vladimir Rubin, 15.06.2015

TECHNISCHE UNIVERSITÄT DARMSTADT

.consulting .solutions .partnership

msg systems

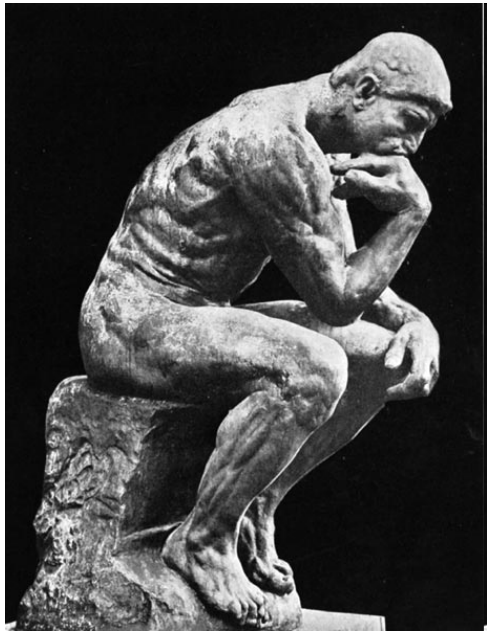
Dr. Vladimir Rubin

- M.Sc. in Computer Science at Moscow State University of Railway Transport
- PhD in Computer Science at the University of Paderborn (International Graduate School), Department Software Engineering
- ~ 3 Years Netcracker Technologies Corp, USA
- ~ 3 Years Capgemini sd&m, Frankfurt/M.
- > 3 Years msg systems ag, Frankfurt/M.
- Today: Independent IT Architekt and Consultant, collaboration with msg systems, Frankfurt, Germany
- Points of interest:
  - Methodical SW-Development and IT-Architecture
  - Process Mining and Data Science
  - Model-driven Software Development (MDD)

Private

- 33 years old, married, one child
- Hobbies: Music, Yoga, Badminton, Volleyball, Traveling, …

Why MDD?

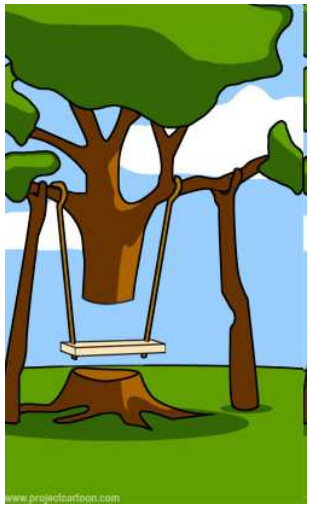


Why do we need
Model-driven Software Development
(MDD)?

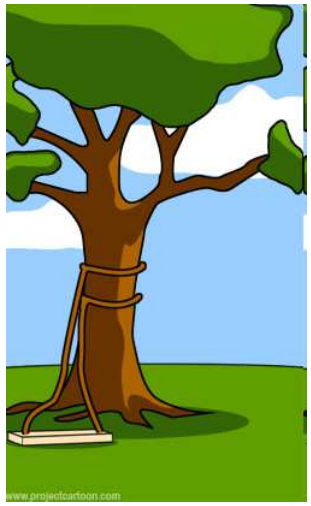
Software Projects – poor Communication

.consulting .solutions .partnership

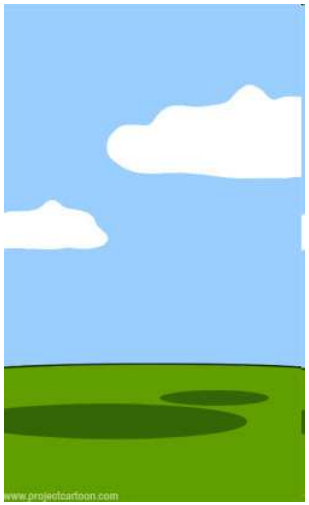
How the **customer** explained it



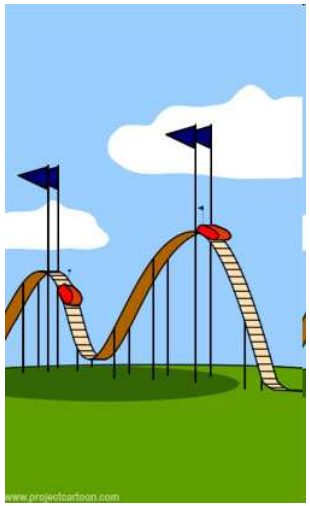
How the **analyst** designed it



How the **programmer** wrote it



How the project was documented



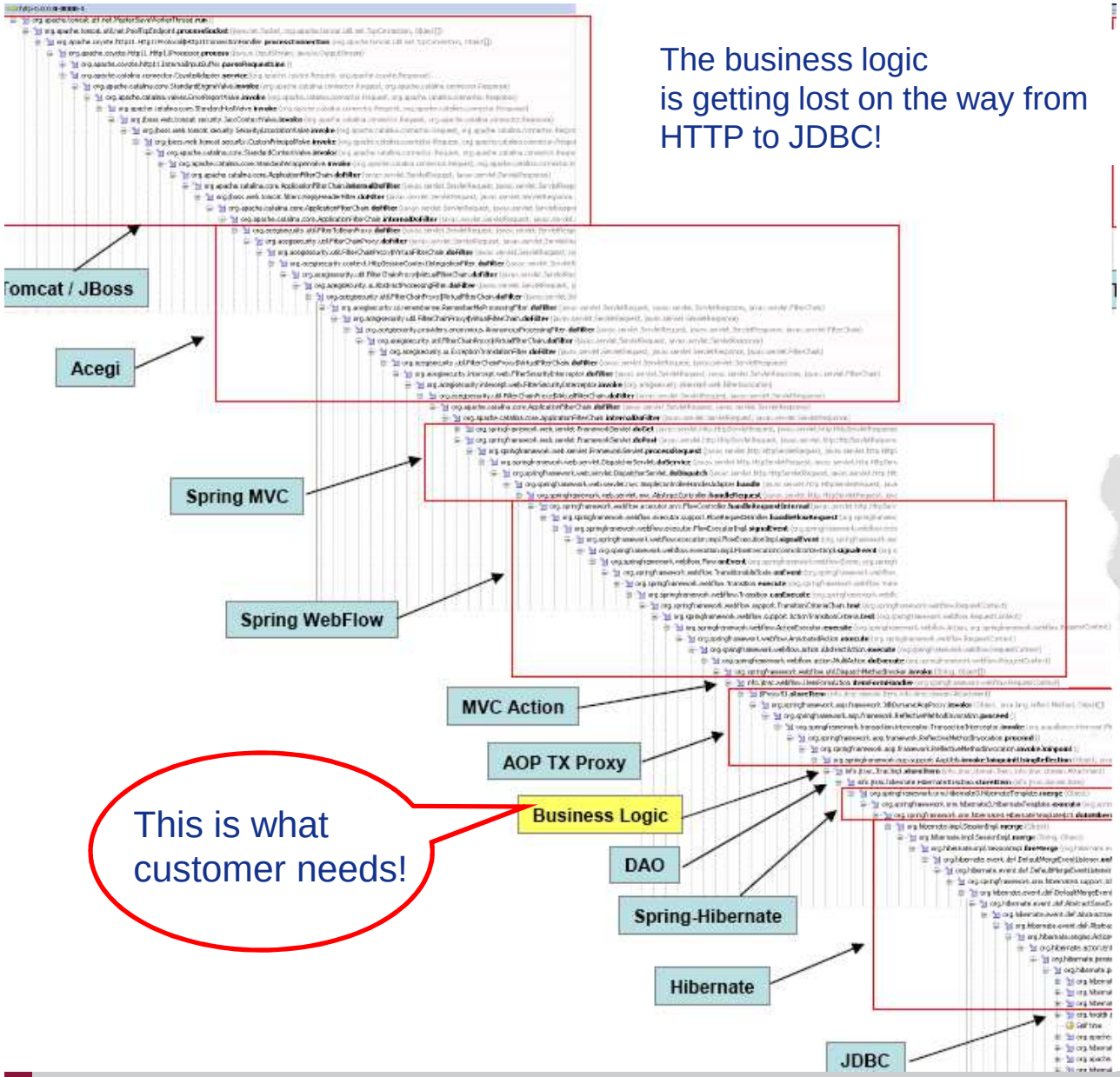
How the customer was billed

What the customer really needed

Good communication between **Business experts** and **IT** is needed – common clear language!

Software Projects – too much Technology

.consulting .solutions .partnership



The business logic
is getting lost on the way from
HTTP to JDBC!

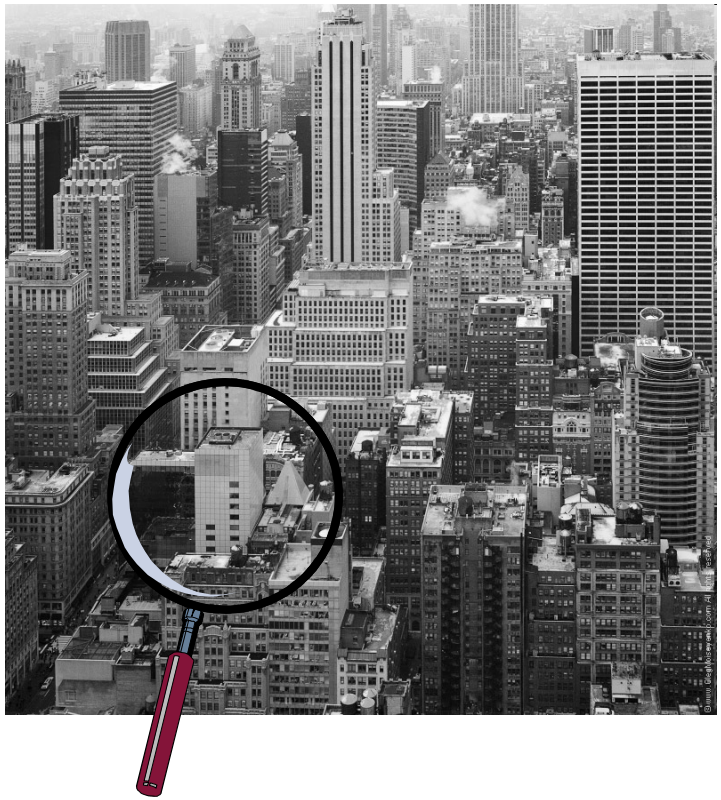
This is what
customer needs!

IT People should
concentrate on the
important things – on the
Business Domain.

* http://ptrthomas.wordpress.com/2006/06/06/java-call-stack-from-http-upto-jdbc-as-a-picture/

Software Projects – enormous Complexity

.consulting .solutions .partnership

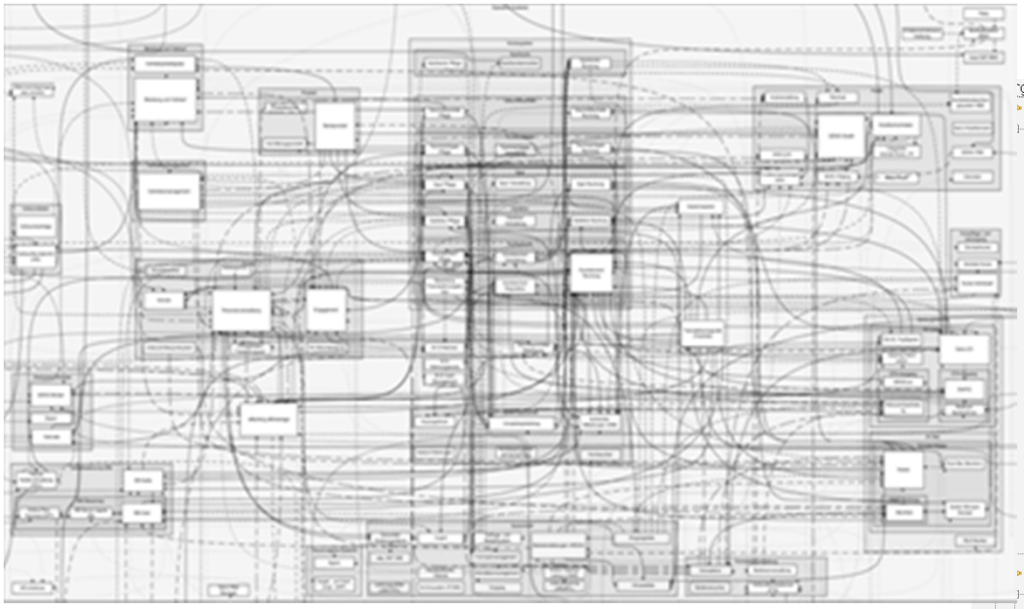


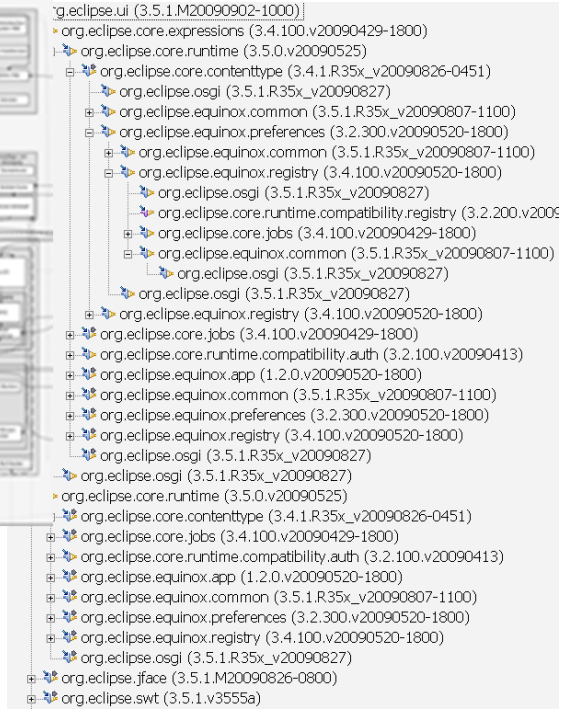
In order to get an overview and not to get lost in details, people use levels of Abstraction.

In order to deal with the complexity (eliminate and understand), people use Composition and Decomposition.

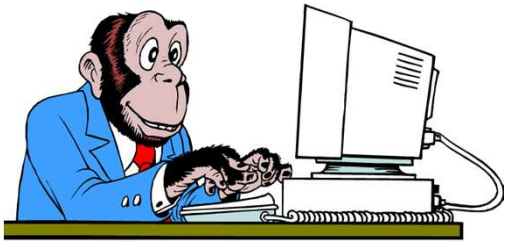
In order to develop complex SW-system, we need
Abstraction,
Composition/Decomposition

Software Projects – Craftsmanship





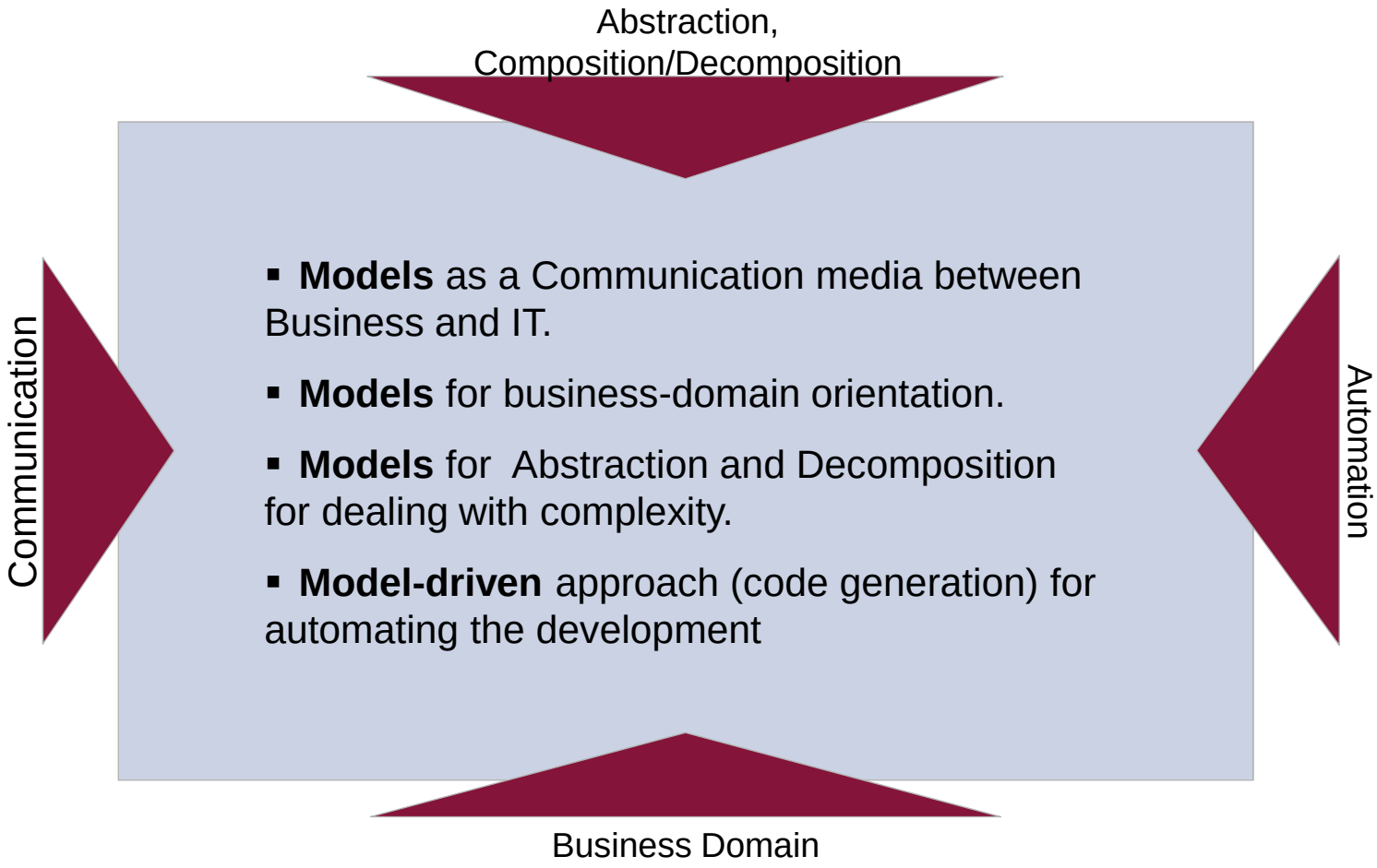
Today, we still assemble the parts manually.



We need **Automation**.

* www.eclipse.org –Eclipse Plugins Dependencies

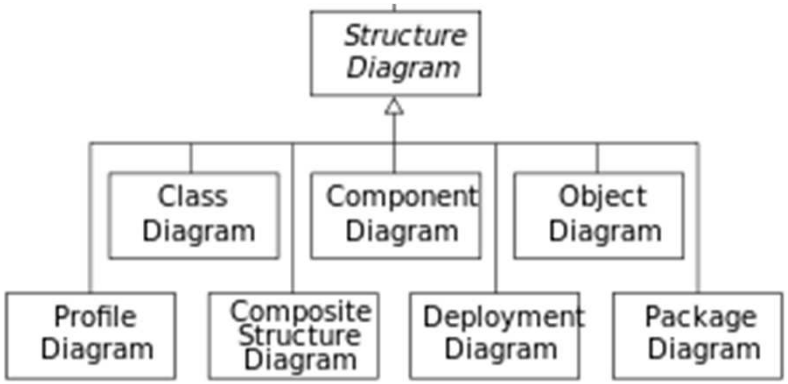
Model Driven Development (MDD)

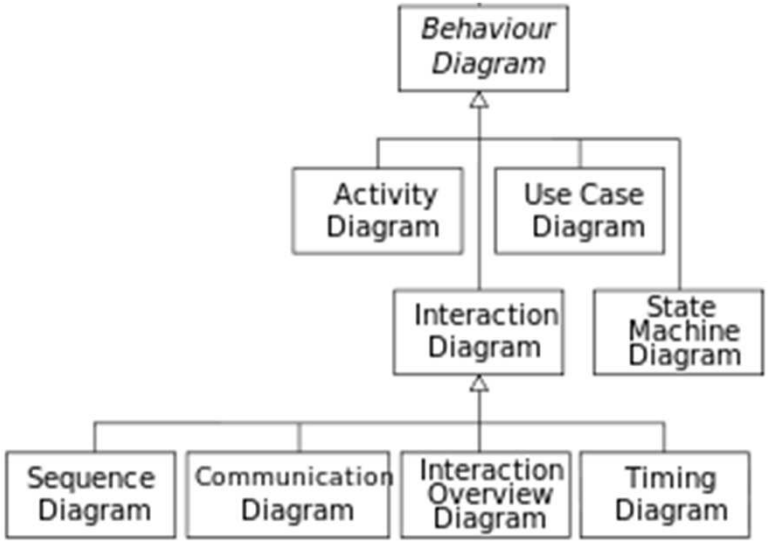


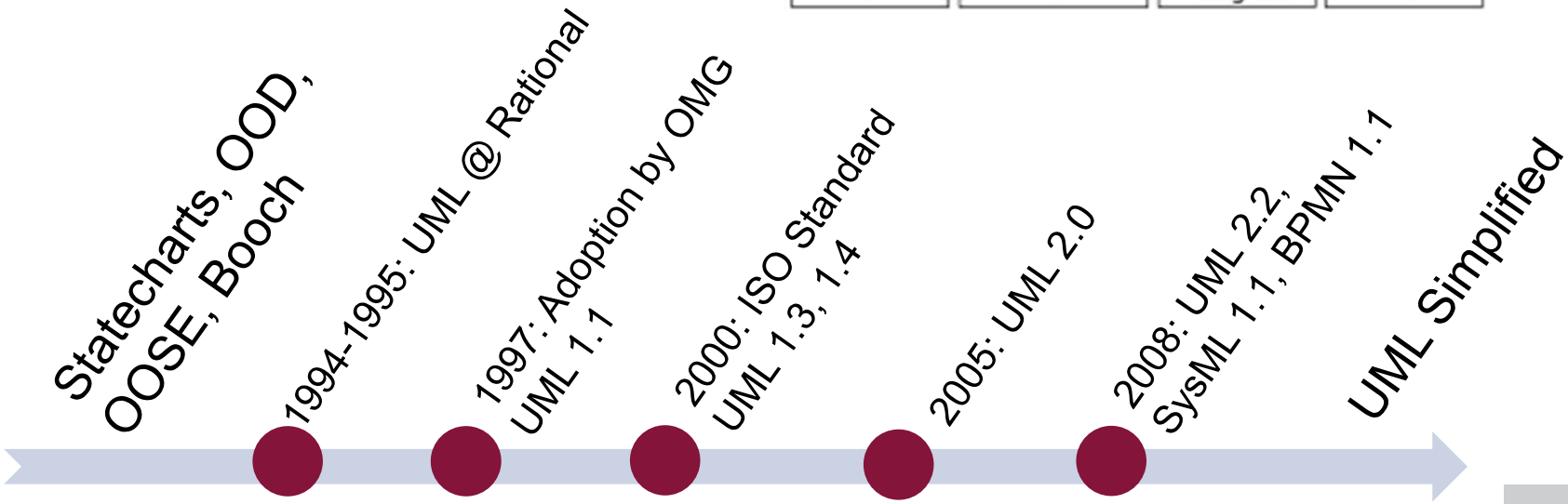
UML – General Purpose modeling language

.consulting .solutions .partnership

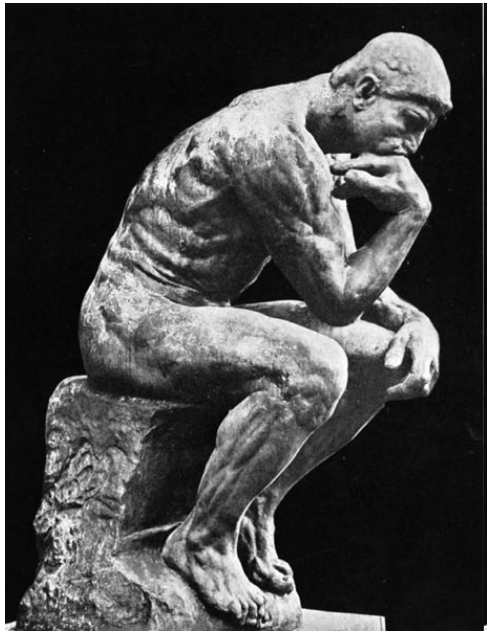
- Family of **graphical** notations, backed by a single **meta-model**
- Combines the best of **object-oriented** modeling methodologies





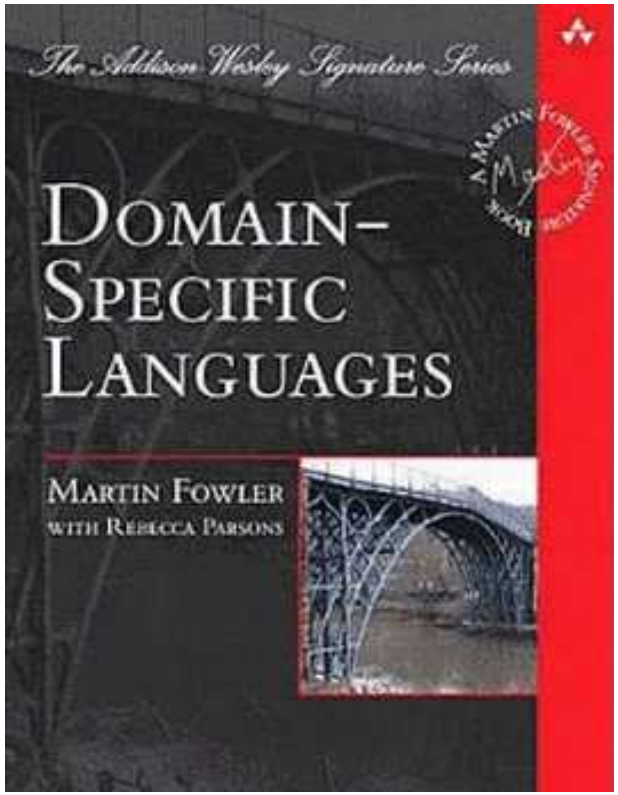


Why DSL?



Why do we need
domain specific languages
(DSL)?

Domain-Specific Languages



Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Martin Fowler

Domain-Specific Language (DSL)

.consulting .solutions .partnership

People use DSL in order to:

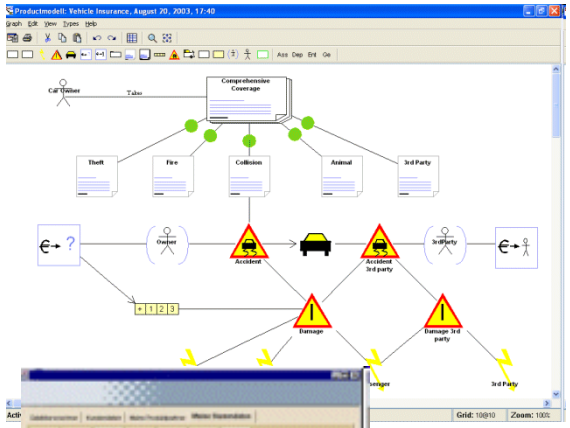
- Carry out more tasks in a better and quicker way
- Reduce „Learning Overhead" and unnecessary complexity
- Stay always in context of the problem

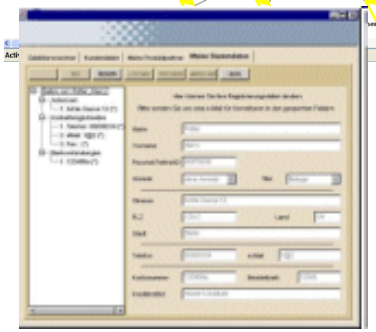
Well known Examples:

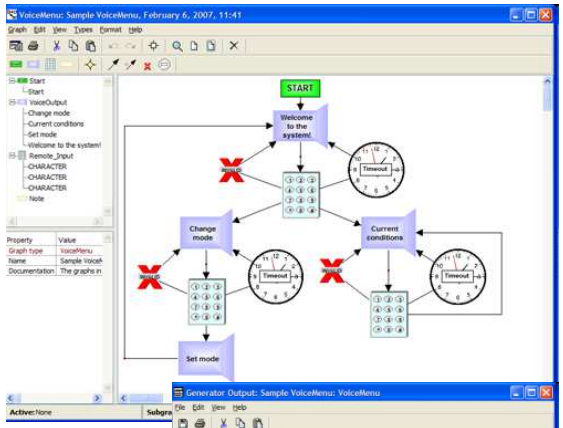
SQL, BPMN, BNF, Excel, Make, Ant, GraphViz, YACC, Unix Shell, HTML, CSS, Visio Stencils, GUI Builders, …

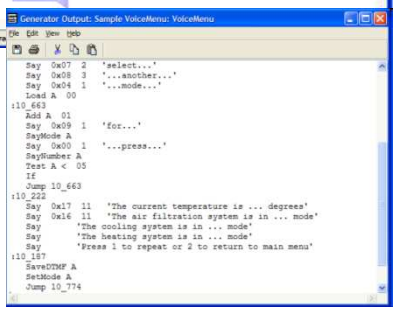
Business Domains & DSLs

.consulting .solutions .partnership





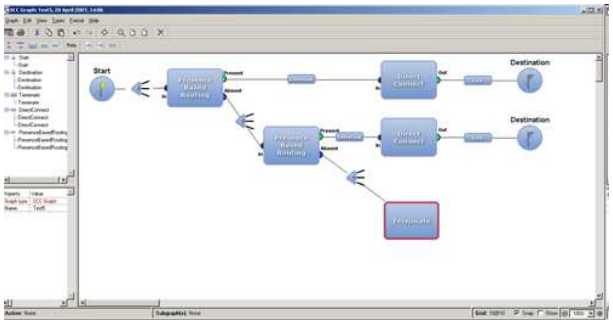




Insurance: Product and Process Definitions J2EE Website Code Generations

Embedded: Menu System for Microcontrollers Assembler Code Generation

DSLs for Business Domains



Telecom: Definitions of IP Telephony Services Generation of Config Scripts for Telephone Servers

* http://www.dsmforum.org/; Domain-Specific Modeling: Enabling Full Code Generation, Kelly & Tolvanen (2008)

AGENDA

1. Why MDD and DSL?
2. **DSL Example**
3. DSL Framework Example
4. MDD for IT-Projects
5. Experience Report

Example: Application *WebAgenda*

Application: Publish Agenda (Meetings, Presentations) online

Idea

MDD + DSLs:

Agenda is modelled as a *Mindmap and Web-Content* is generated and published automatically

Standard solution

- Agenda is created by a speaker on paper or in a presentation tool
- Agenda is manually checked
- Each agenda is recreated in CMS/HTML/Javascript/JSP by a technical person and then published

MDD solution

- Agenda mindmap is created by a speaker
- Agenda is checked automatically (the rules are defined only once)
- Each agenda is automatically transformed to CMS/HTML/JavaScript/JSP (Implementation is done only once)

Example: Application *WebAgenda* (Notation)

Idea

MDD + DSLs: Agenda is modelled as a *Mindmap* and the *Web-Content* is *automatically* generated and published

We use textual DSLs

eclipse modeling PROJECT

Xtext /TMF

openArchitectureWare

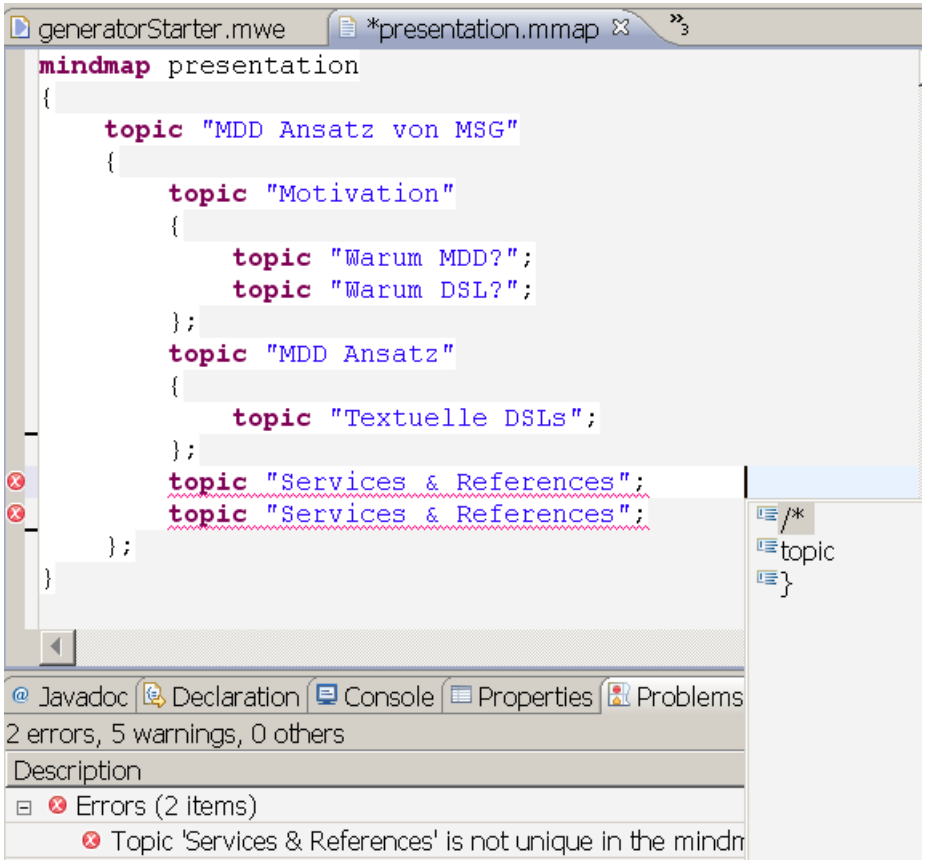
Example: Application *WebAgenda* (Methodology)

Idea

MDD + DSLs: Agenda is modelled as a *Mindmap* and the *Web-Content* is *automatically* generated and published

Methodology:

1. Define DSL (manually)
   a. Grammar:
      Mindmap containsTokens, Tokens contain other Tokens, etc.
   b. Validation Rules: There should be no 2 token with the same text
2. Model Editors and Validators are generated automatically



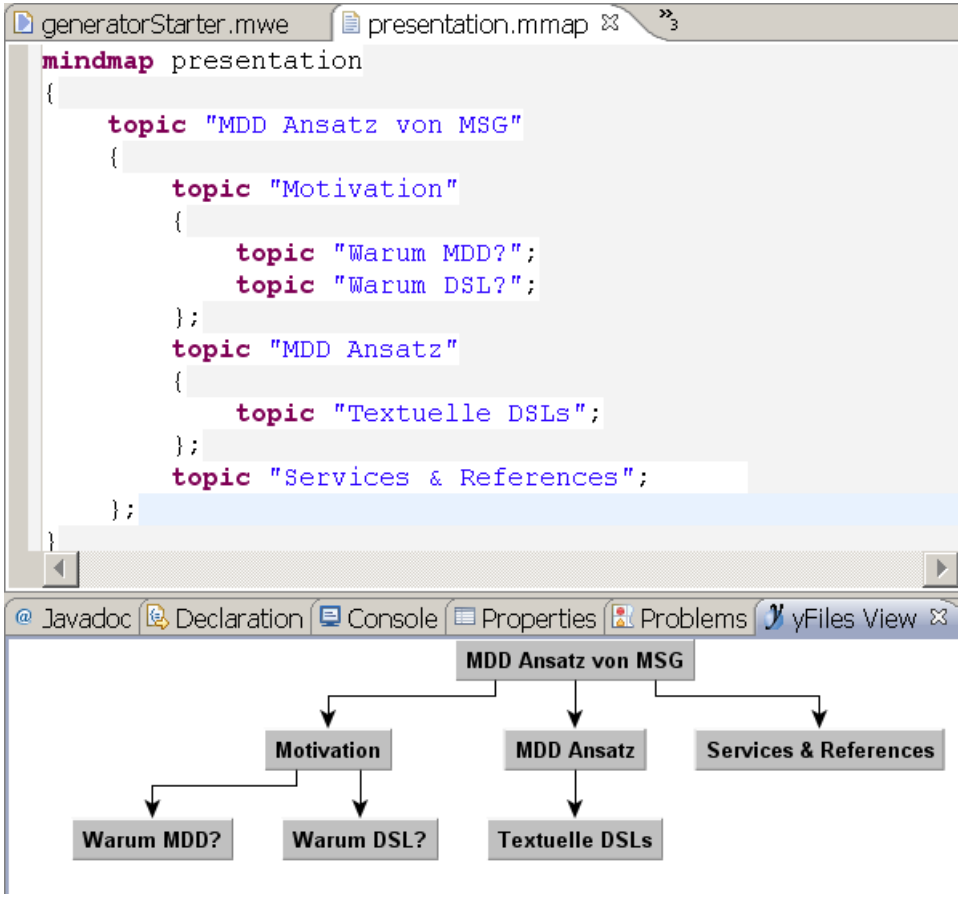
Example: Application *WebAgenda* (Methodology)

Idea

MDD + DSLs: Agenda is modelled as a *Mindmap* and the *Web-Content* is *automatically* generated and published

Methodology:

1. Define DSL (manually)
   a. Grammar:
      Mindmap containsTokens, Tokens contain other Tokens, etc.
   b. Validation Rules: There should be no 2 token with the same text
2. Model Editors and Validators are generated automatically
3. Define the visualization of models



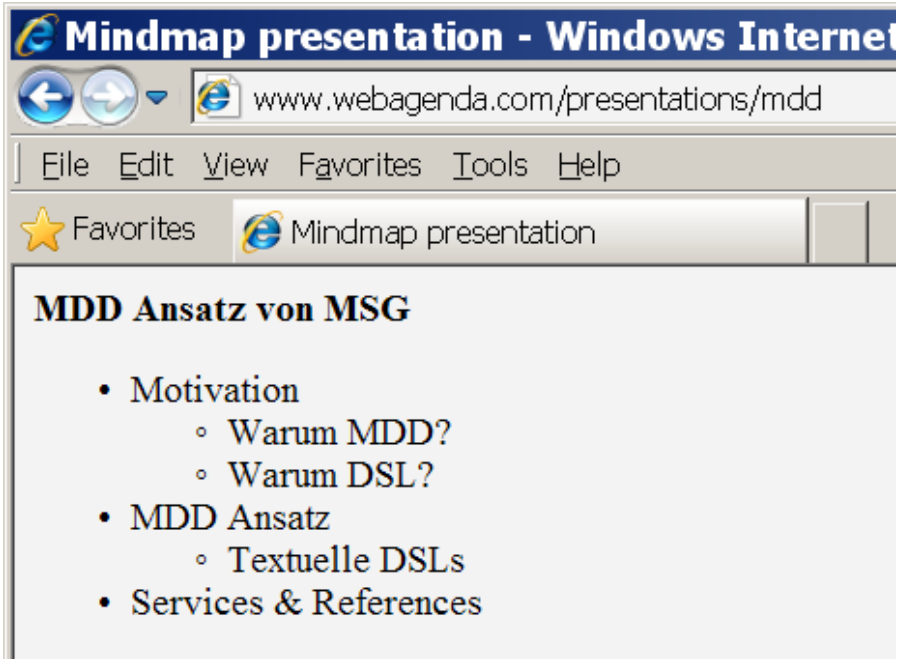
Beispiel: Anwendung *WebAgenda* (Methodologie)

Idea

MDD + DSLs: Agenda is modelled as a *Mindmap* and the *Web-Content* is *automatically* generated and published

Methodology:

1. Define DSL (manually)
   a. Grammar:
      Mindmap containsTokens, Tokens contain other Tokens, etc.
   b. Validation Rules: There should be no 2 token with the same text
2. Model Editors and Validators are generated automatically
3. Define the visualization of models
4. Define the generation to HTML



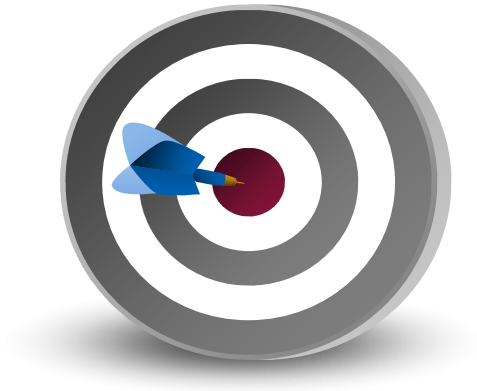
Example: Application *WebAgenda* (Methodology)

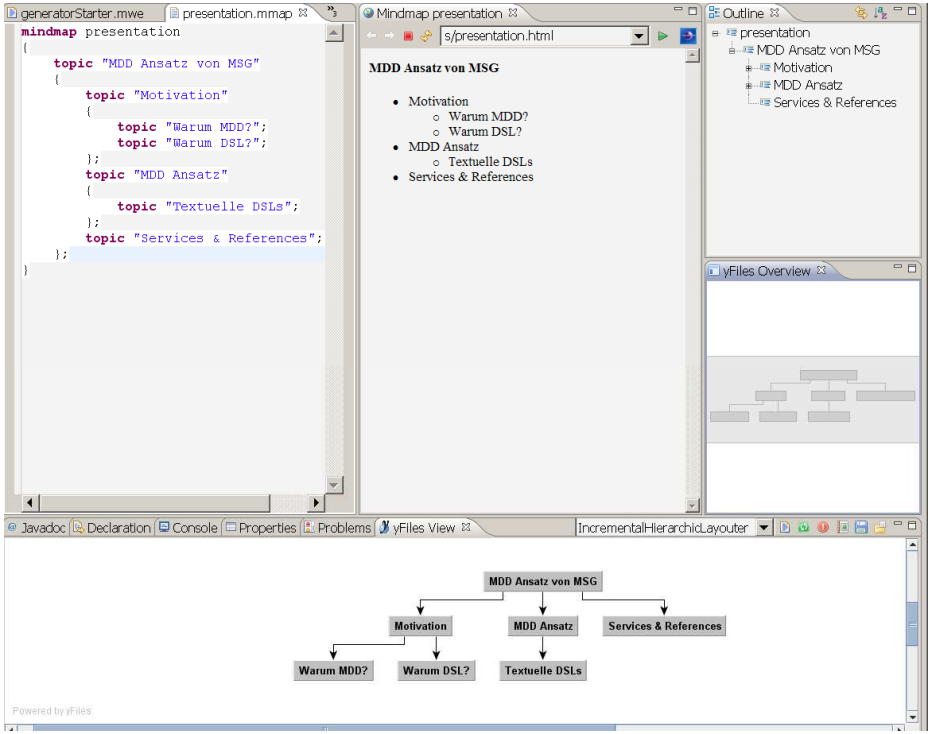
Idea

MDD + DSLs: Agenda is modelled as a *Mindmap* and the *Web-Content* is *automatically* generated and published

Methodology:

1. Define DSL (manually)
   a. Grammar:
      Mindmap containsTokens, Tokens contain other Tokens, etc.
   b. Validation Rules: There should be no 2 token with the same text
2. Model Editors and Validators are generated automatically
3. Define the visualization of models
4. Define the generation to HTML





Conclusion

We have created an application for defining MindMaps, visualizing them and generatin HTML from them.

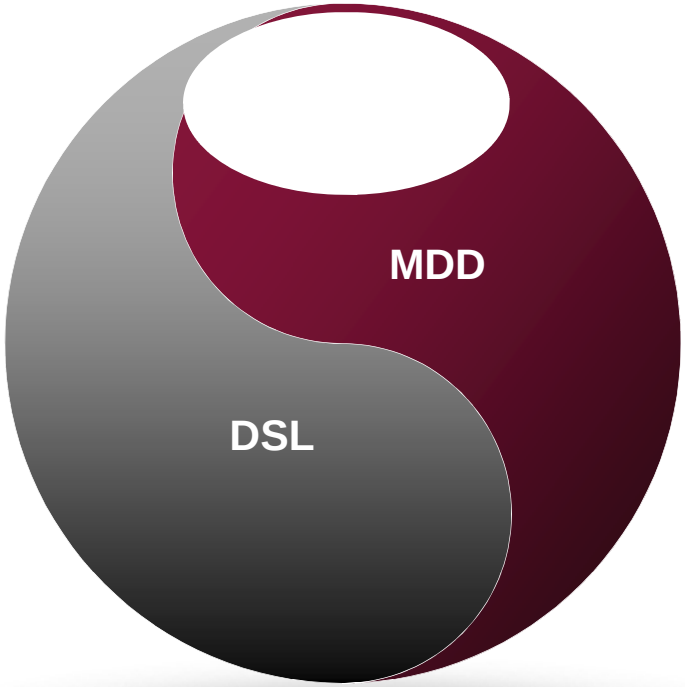
The solution is generic and flexible. Technology (HTML) and business (MindMap) are separated.

The overall cost: **4 Hours. / 1 Person**

AGENDA

1. Why MDD and DSL?
2. DSL Example
3. **DSL Framework Example**
4. MDD for IT-Projects
5. Experience Report

Common project basis = MDD + DSL

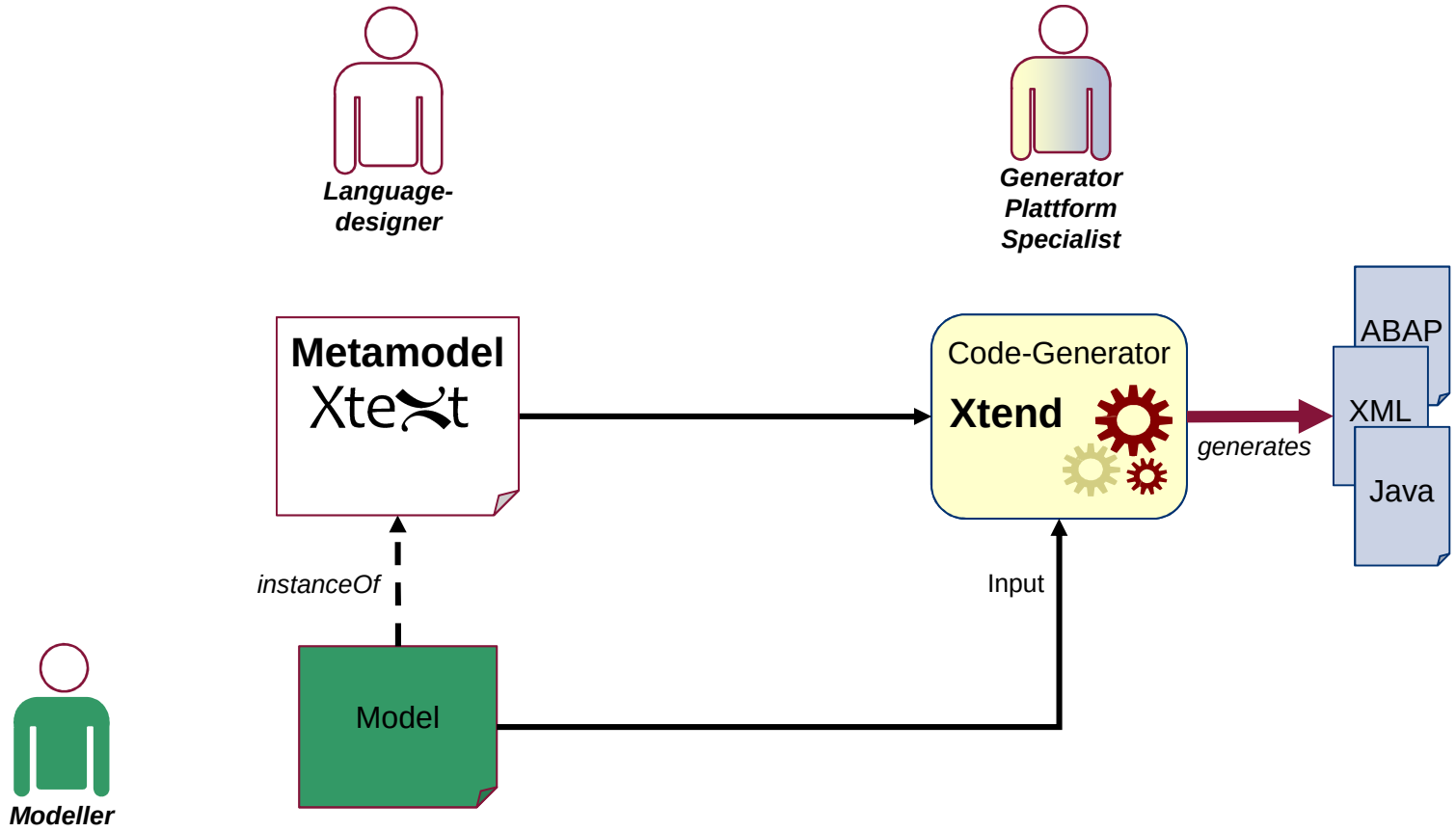


Principles

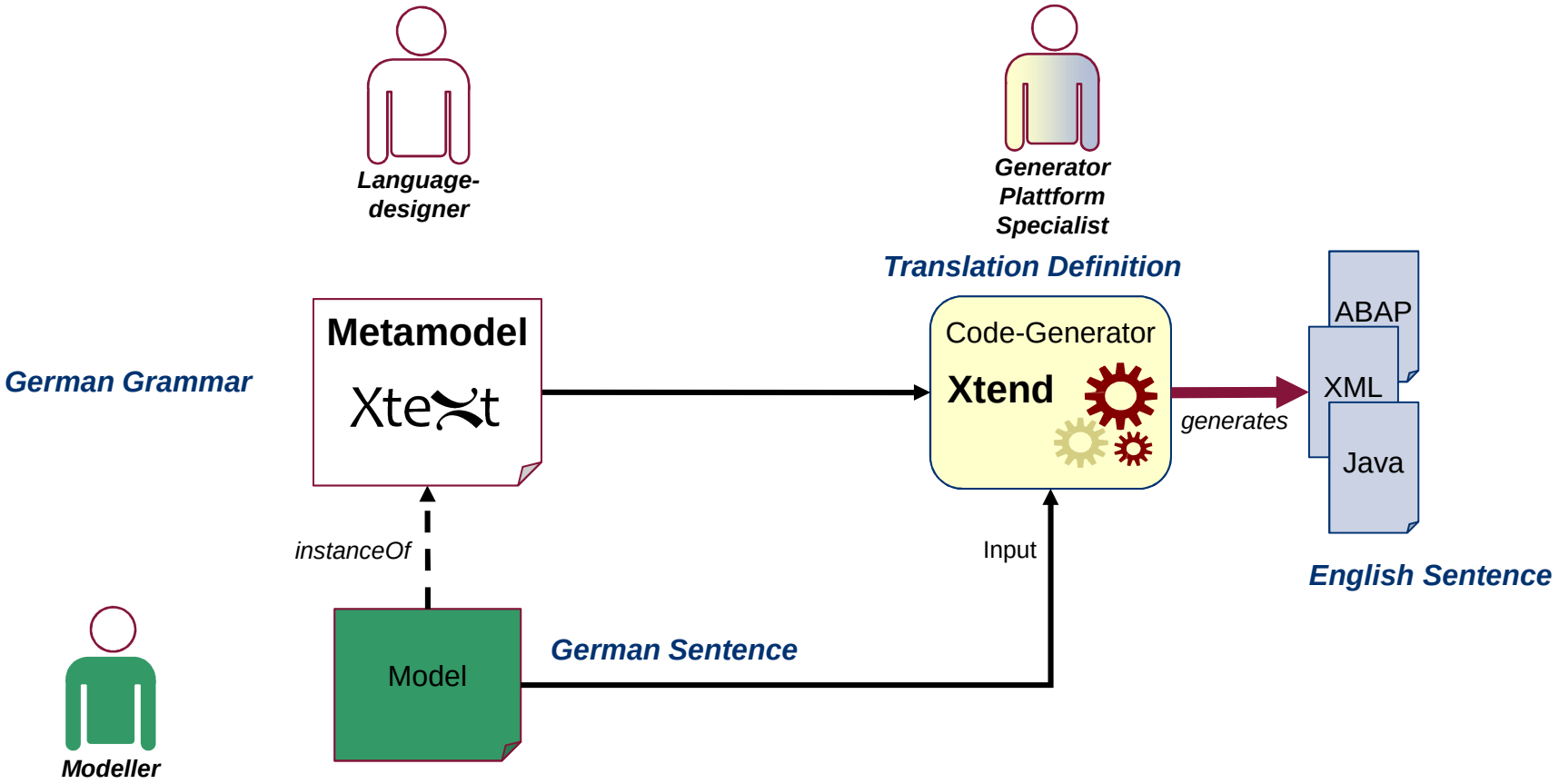
- DSL-Model is the core of the development Process
- Eclipse integrated Tooling (EMF-based)
  - Language Design
  - Generation with Eclipse
  - Variants Management

Important prerequisite for efficiency: Know-how of Project-Keyplayers

Overview: DSL Artefacts, Tools, Roles

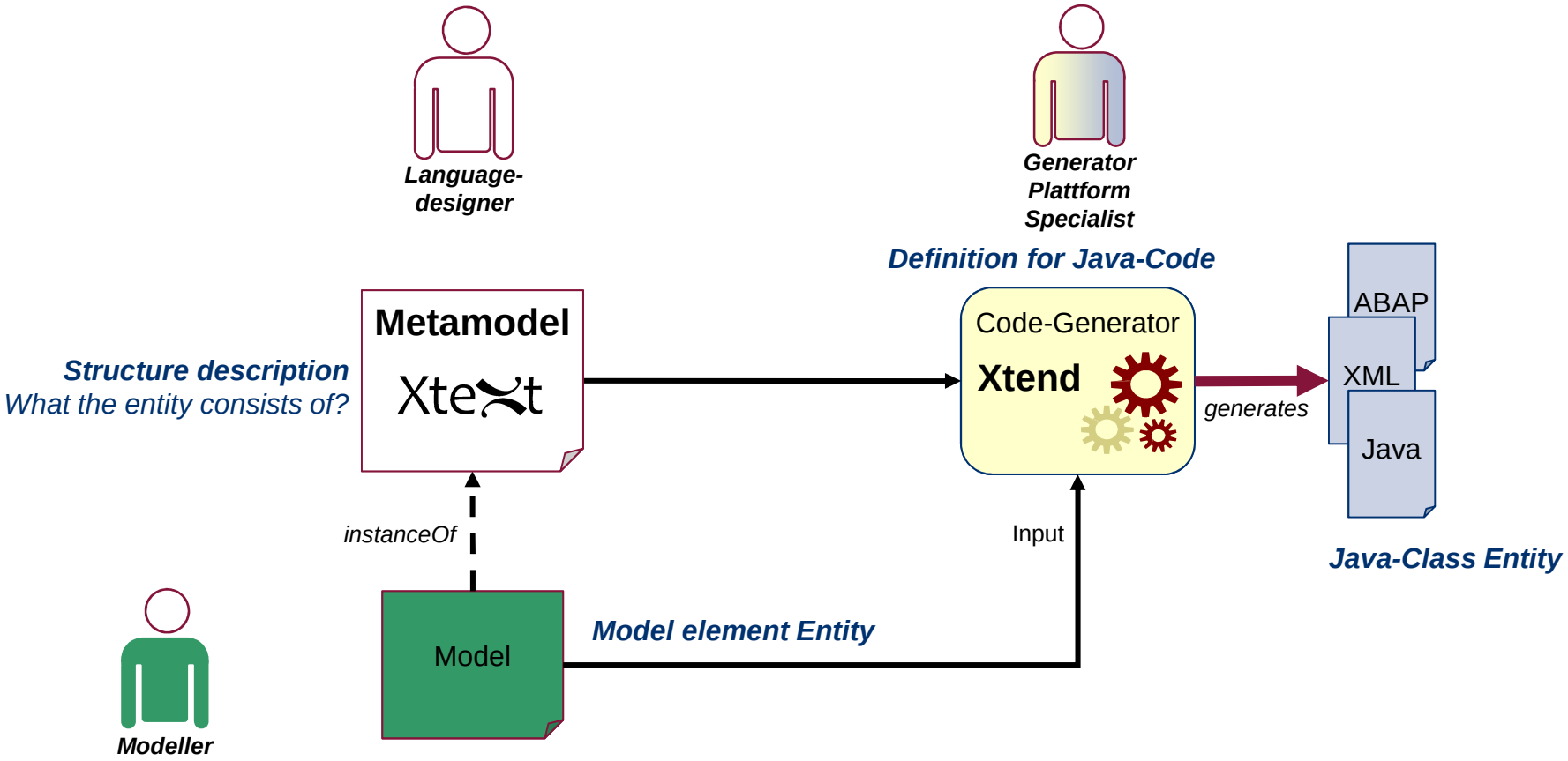


Analogy to german language



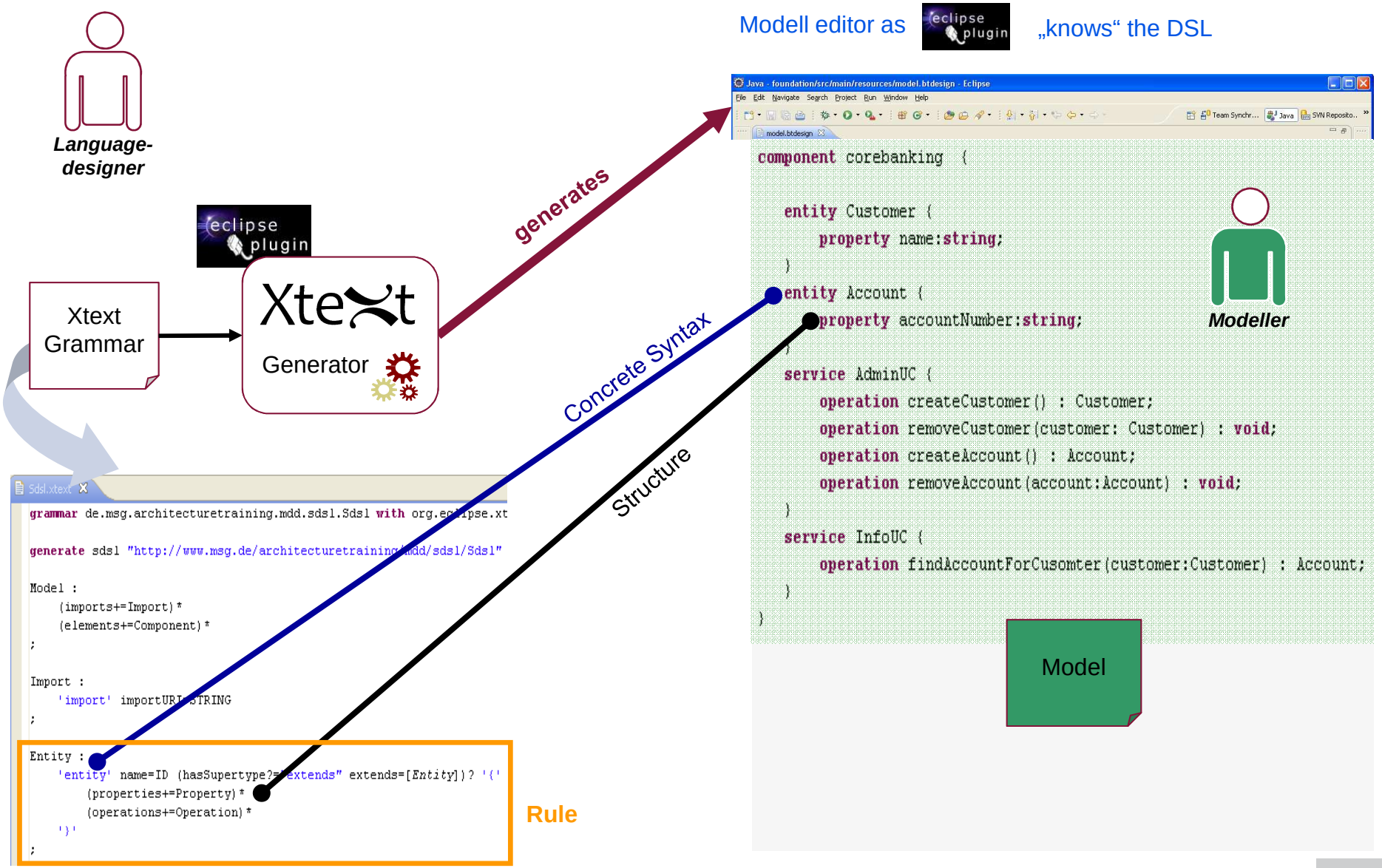
Technical Example: Entity

.consulting .solutions .partnership



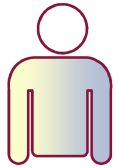
Xtext for Language Design

.consulting .solutions .partnership



Xtext for Code Generation

.consulting .solutions .partnership



Generator Plattform Specialist

Xtend

```
Sdsl.xtext
grammar de.msg.architecturetrain

generate sdsl "http://www.msg.de

Model :
    (imports+=Import)*
    (elements+=Component)*
;

Import :
    'import' importURI=STRING
;

Entity :
    'entity' name=ID (hasSuperty
        (properties+=Property)*
        (operations+=Operation)*
    '}'
;
```

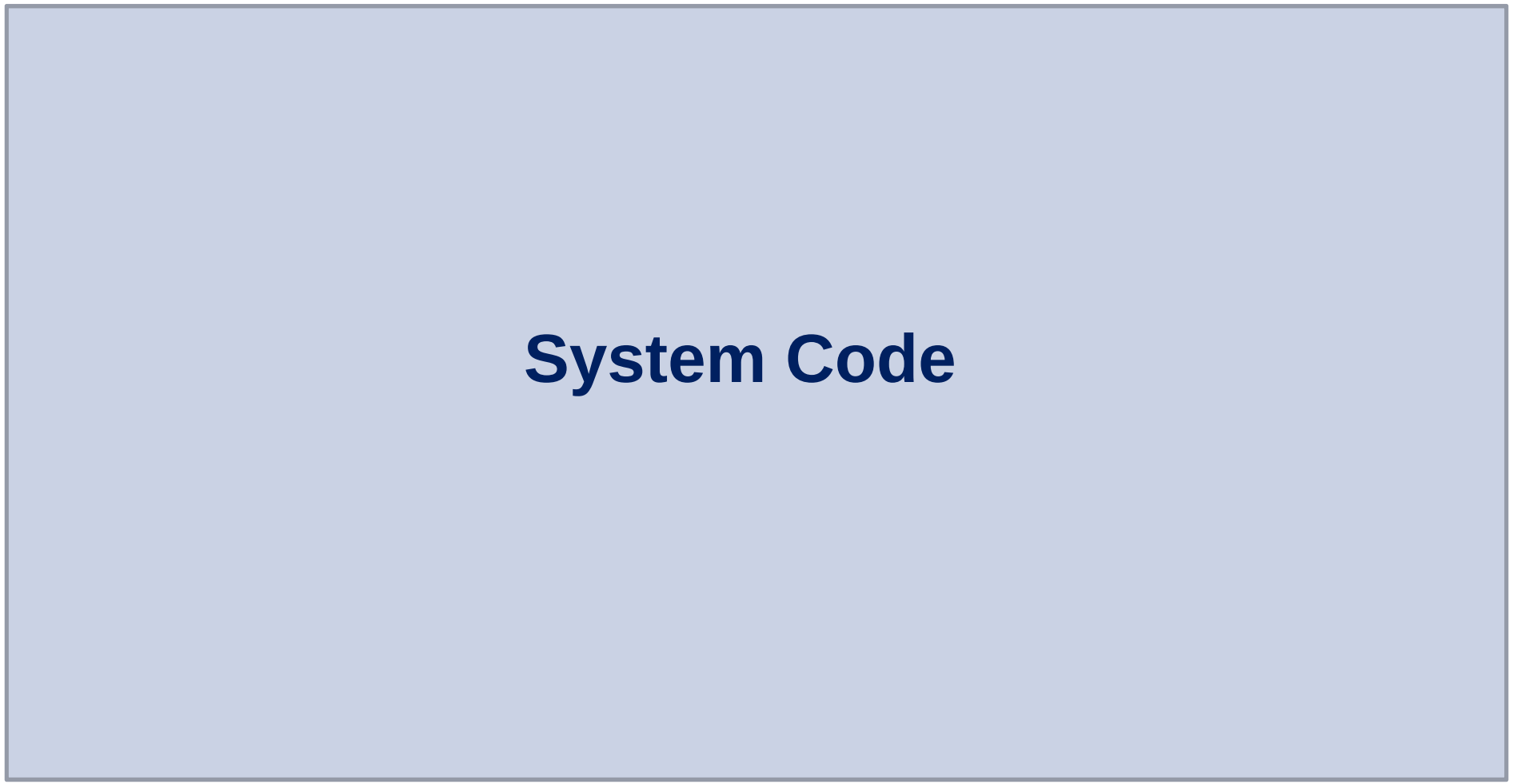
```
class DomainmodelGenerator implements IGenerator {

  @Inject extension IQualifiedNameProvider

  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    for(e: resource.allContents.toIterable.filter(Entity)) {
      fsa.generateFile(
        e.fullyQualifiedName.toString("/") + ".java",
        e.compile)
    }
  }

  def compile(Entity e) '''
    «IF e.eContainer.fullyQualifiedName != null»
      package «e.eContainer.fullyQualifiedName»;
    «ENDIF»

    public class «e.name» «IF e.superType != null
            »extends «e.superType.fullyQualifiedName» «ENDIF»{
      «FOR f:e.features»
        «f.compile»
      «ENDFOR»
    }
  '''
```

AGENDA

1. Why MDD and DSL?
2. DSL Example
3. DSL Framework Example
4. **MDD for IT-Projects**
5. Experience Report

The more formal and explicit the SW-Architecture,

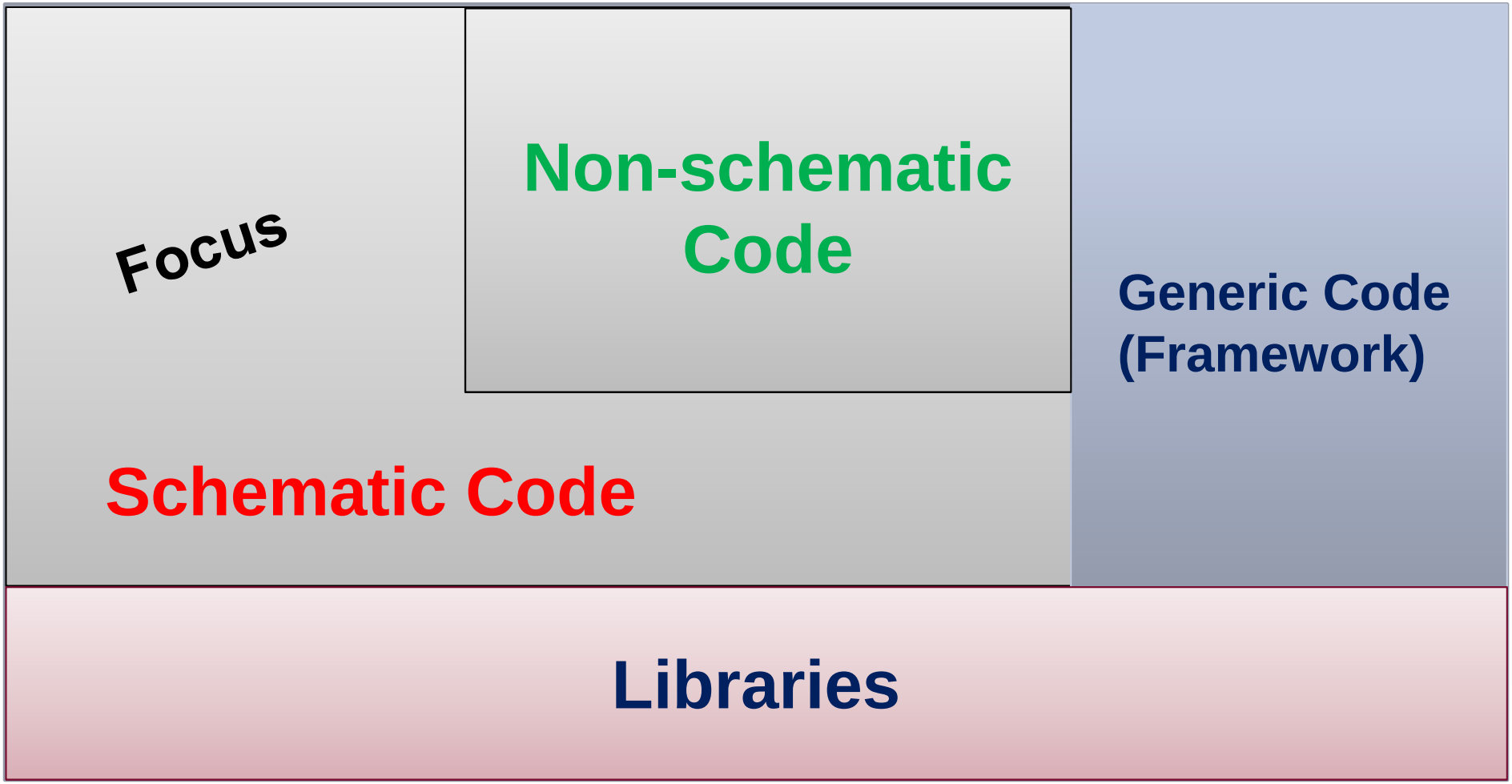
The more schematic & repetitive the SW-code



Automation/ Generation

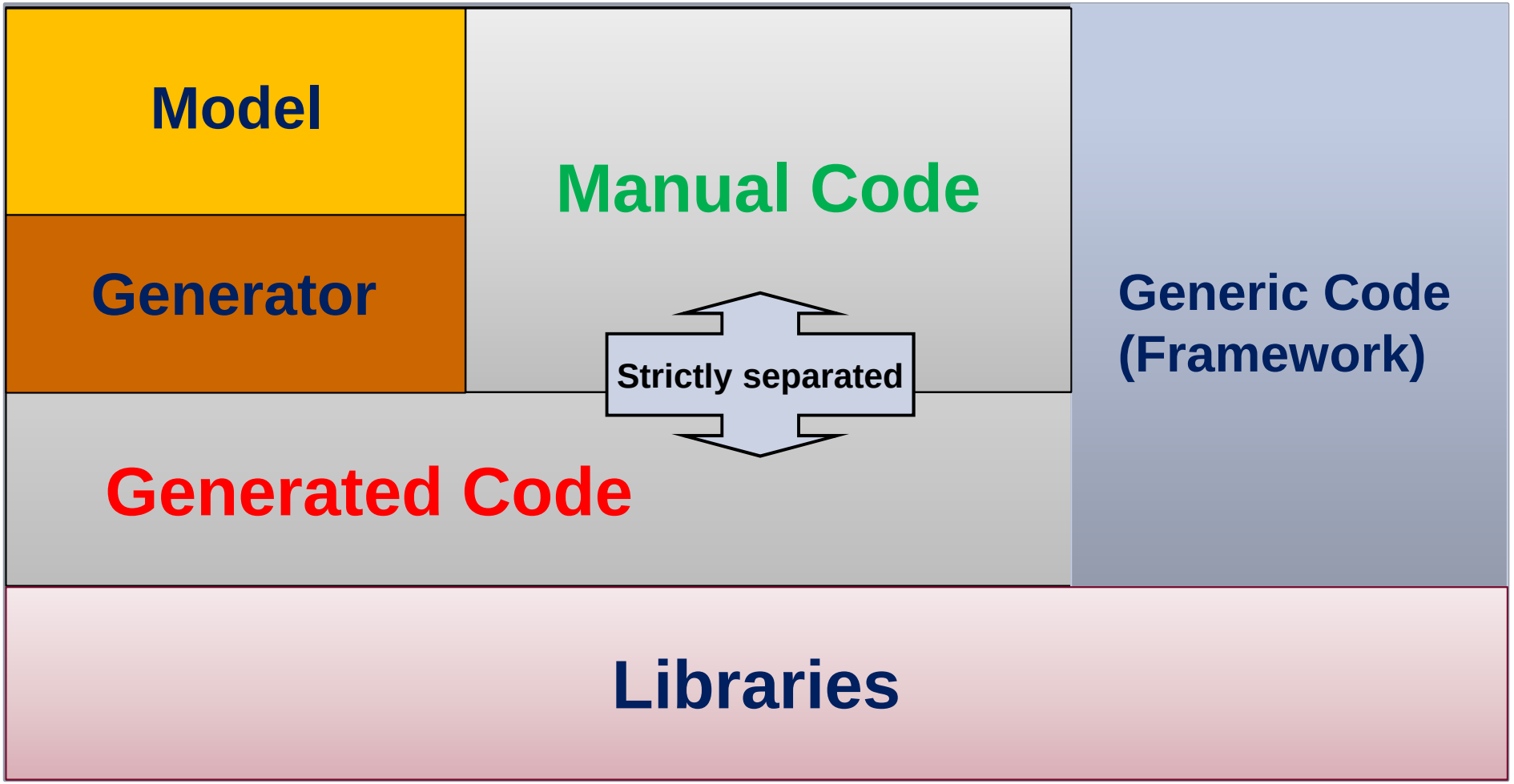
The more formal and explicit the SW-Architecture,

The more schematic & repetitive the SW-code



The more formal and explicit the SW-Architecture,

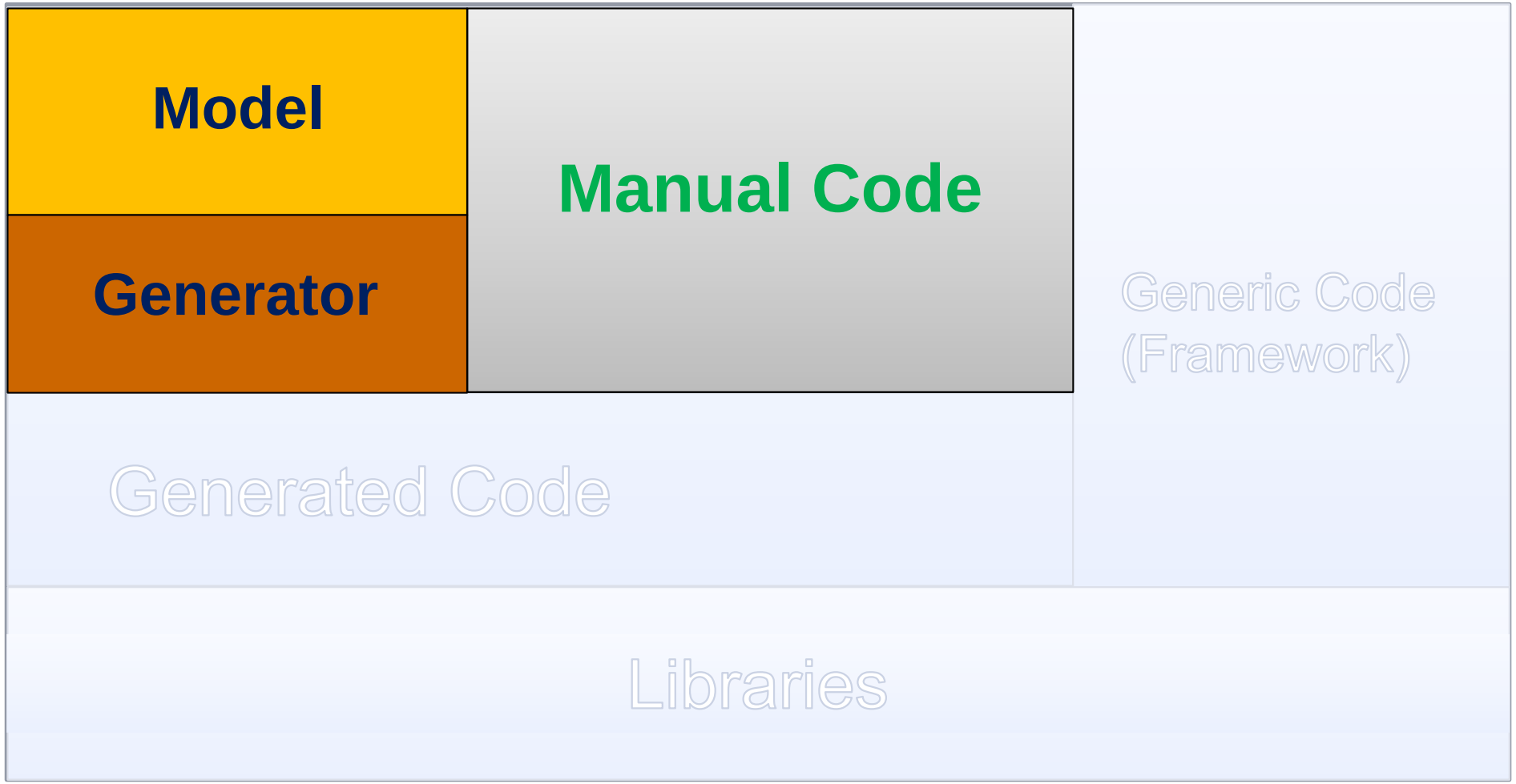
The more schematic & repetitive the SW-code



MDD Advantages for Projects

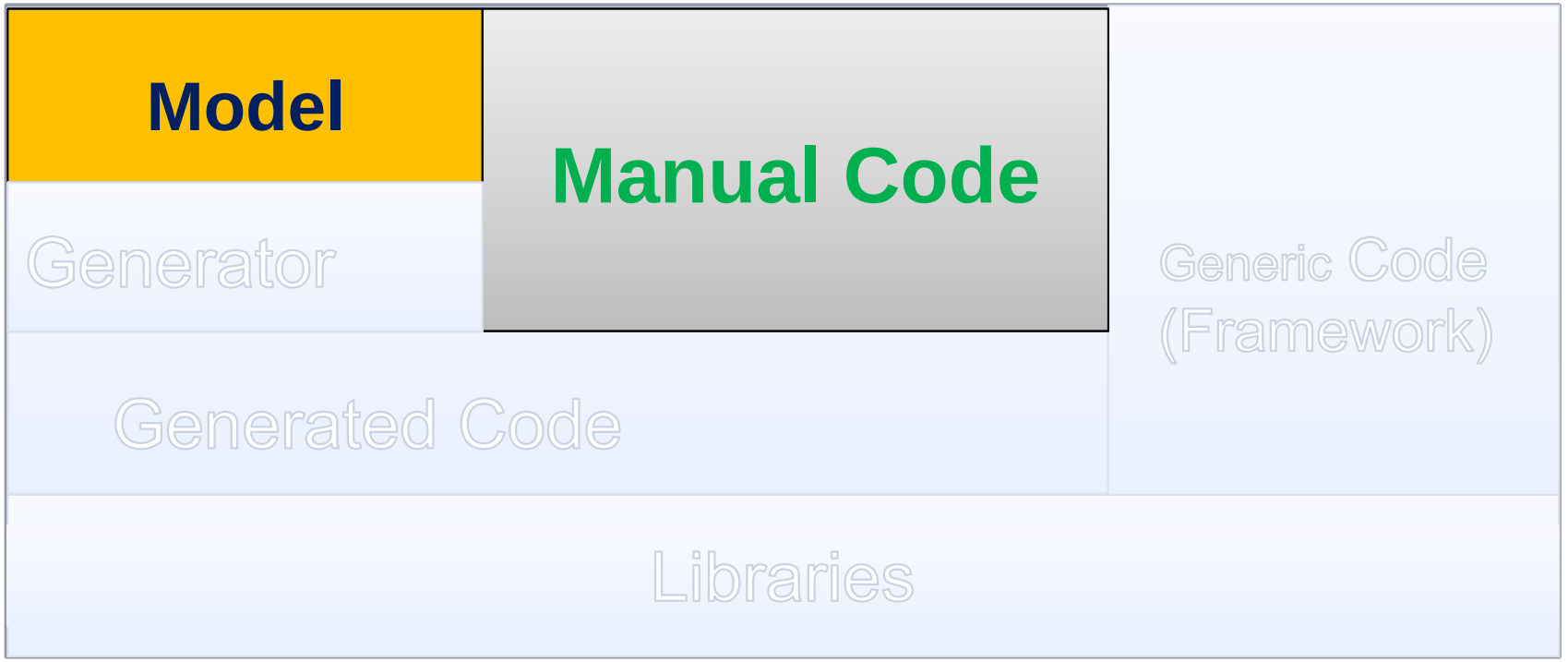
Advantage 1: Increased Productivity

... since developer is concentrating on creative things
and not on the „boilerplate“ code

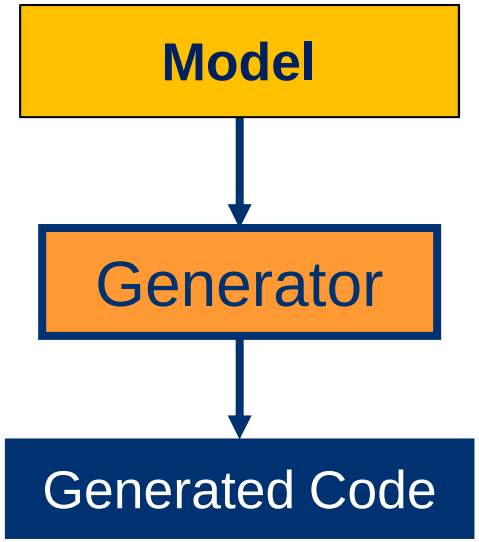


Advantage 2: Resolving the complexity

- Only architects and generator experts have to understand the framework and the generator
- Business Logic (A-Software) and Technology (T-Software) are separated
- Developers focus on the business logic



Advantage 3: Better Quality



Architectural Patterns are implemented only ONCE

- Architectural Changes are controlled and handled centrally in the Generator
- Number of possible design failures decreases
- Principal changes have to be discussed with the architecture team
- Developers tend to produce less code, but better code

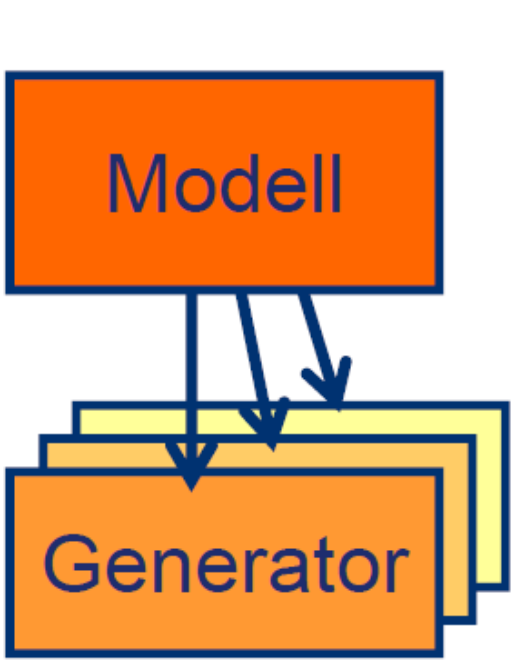
Advantage 4: Flexibility

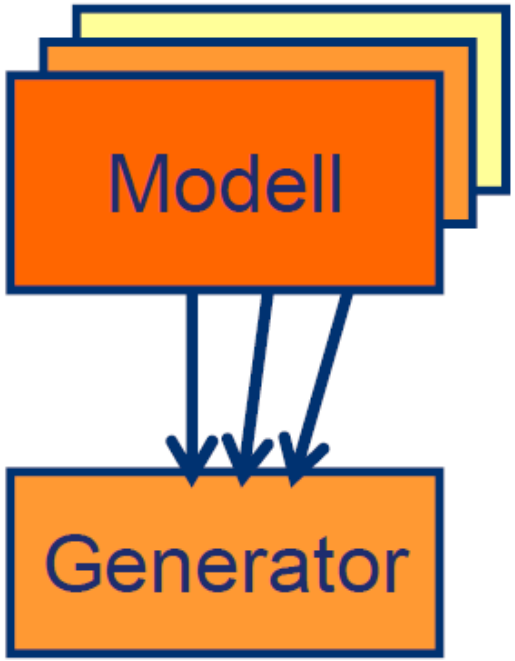
Technology Change

- Model remains
- Generator changes

Product Lines

- Models change
- Generator remains





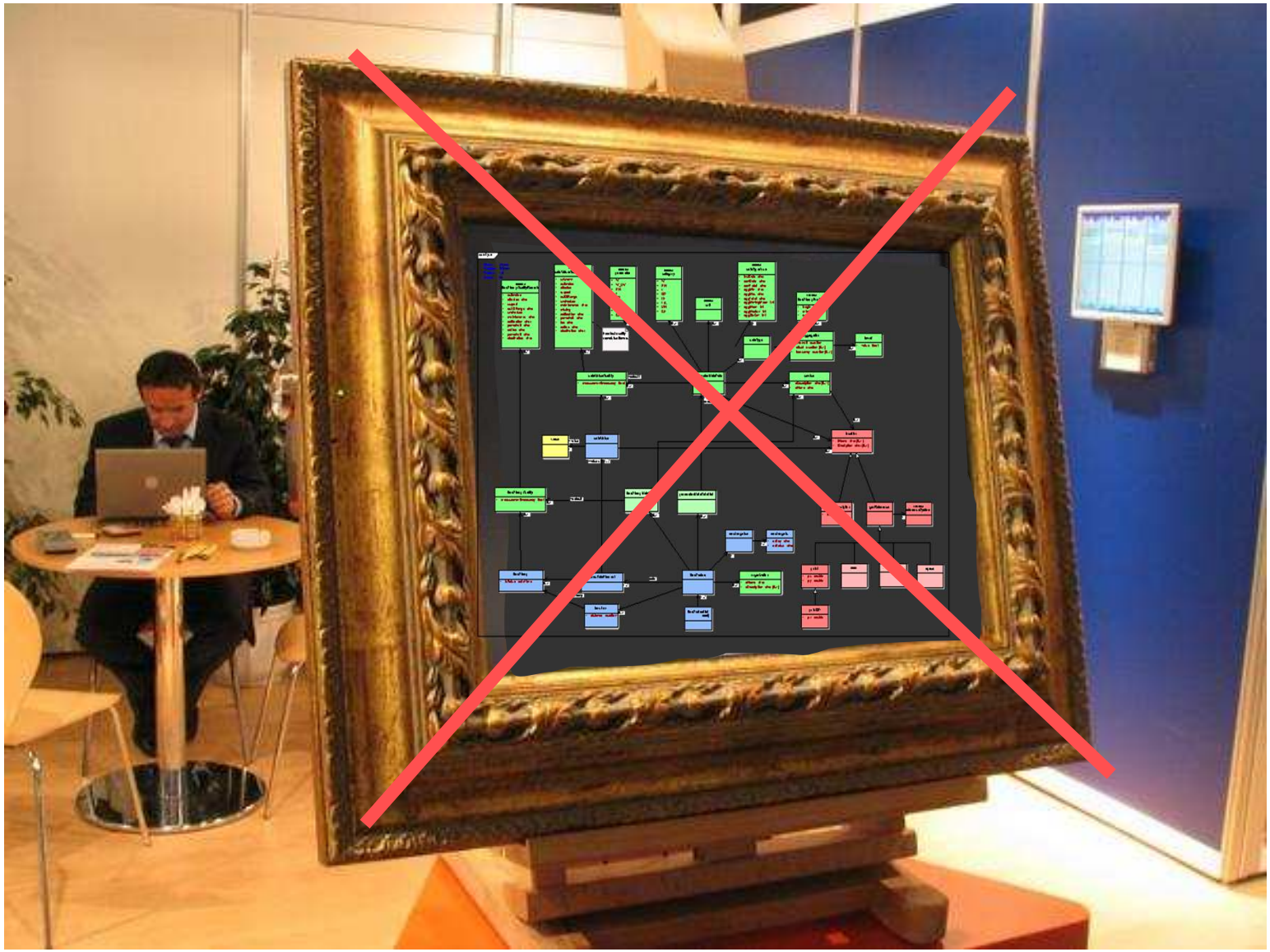
Advantage 5: Documentation

- The model is the core of the development
- The model has to be always up to date (since the code is generated from it)
- The business concepts are documented in the model
- Different documents can be generated from the model

AGENDA

1. Why MDD and DSL?
2. DSL Example
3. DSL Framework Example
4. MDD for IT-Projects
5. **Experience Report**

When is the model useful?!



No Heavy Weight Models

> 1200 Pages Specification

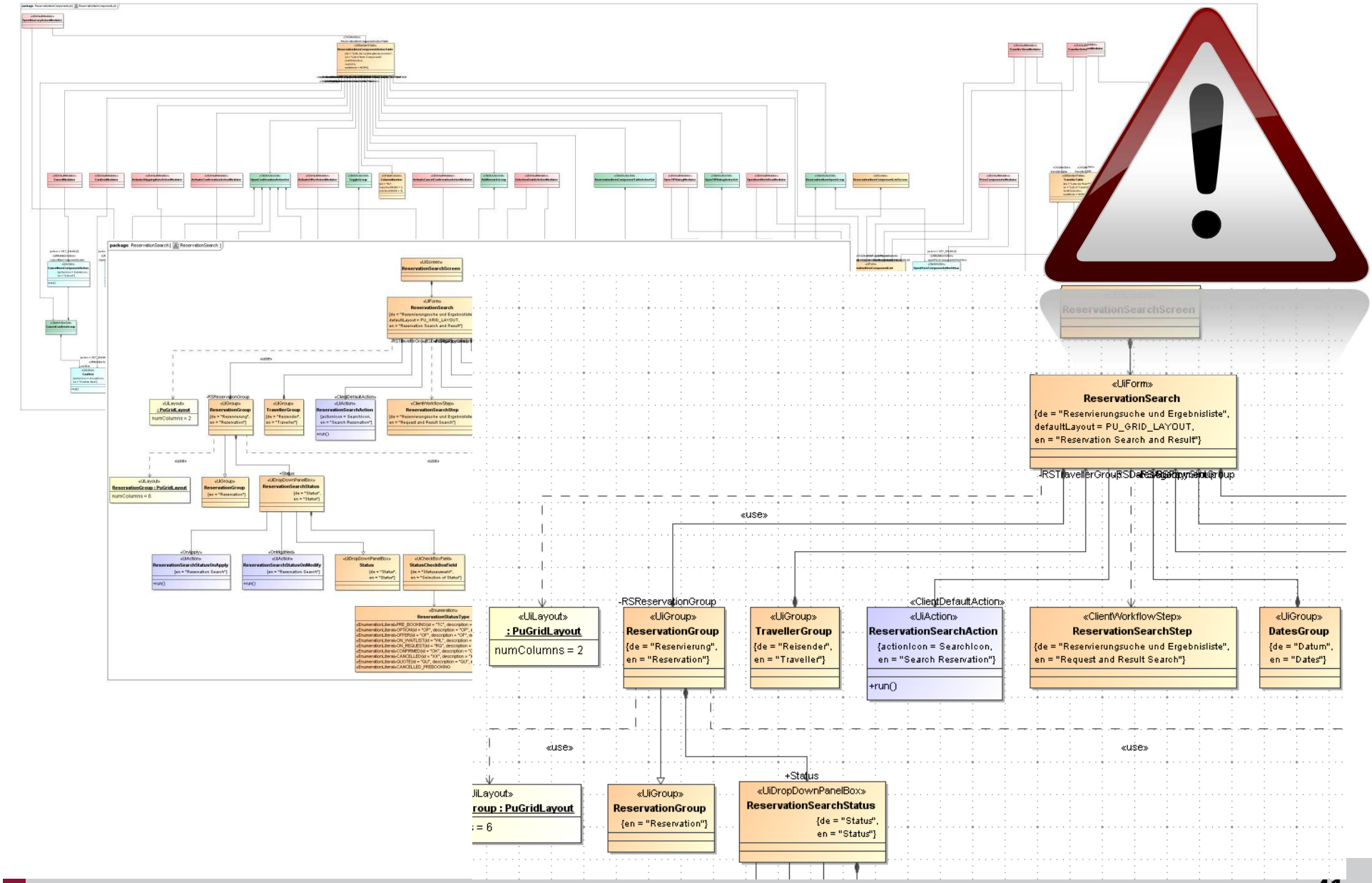
Graphical Representation

**Complex Metamodel –
complex analysis
(Codegenerator, merge, diff)**

**Stereotyped UML-Model
is absolutely individual –
not a standard anymore**

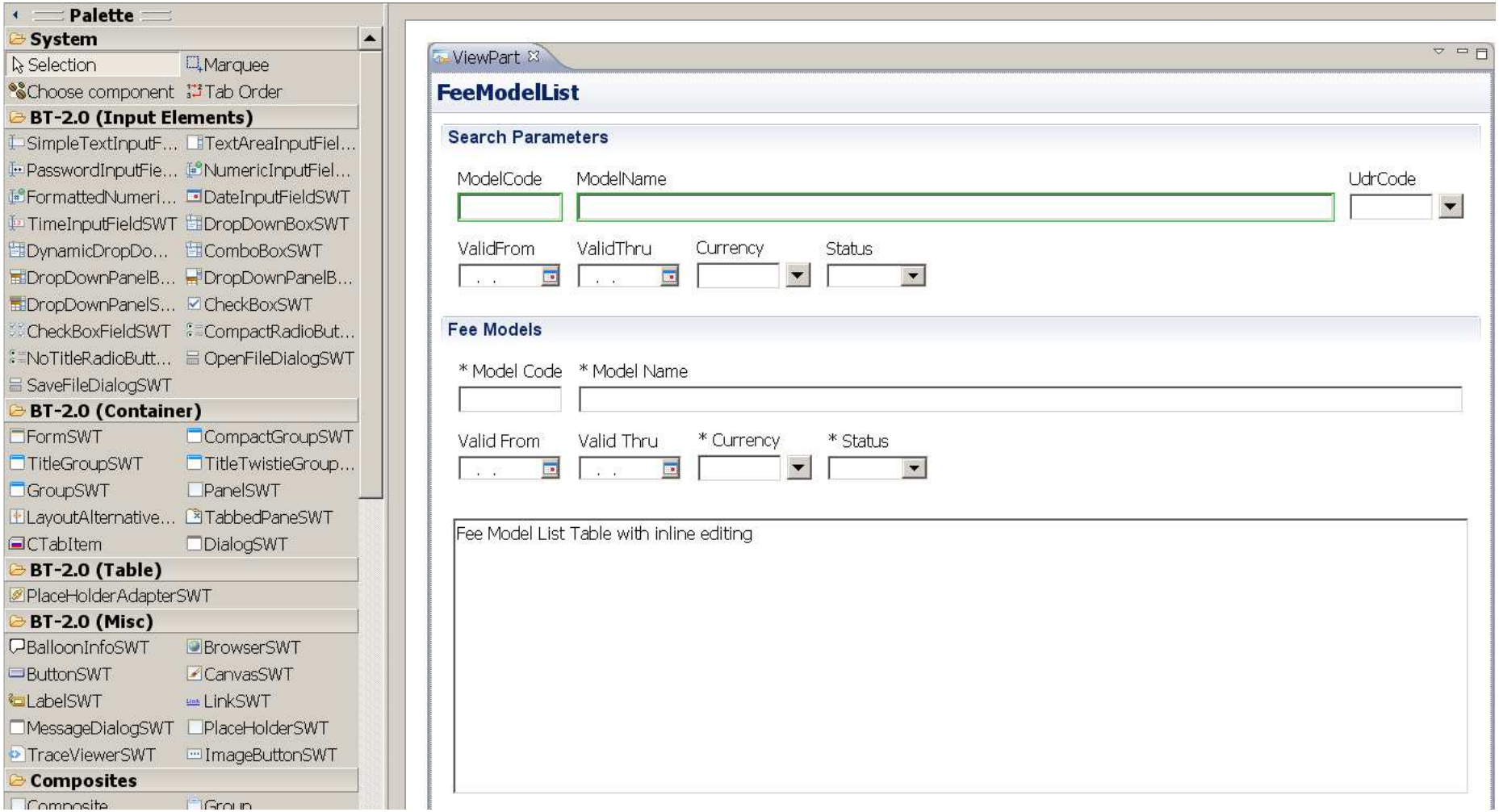
GUI Modeling with UML (Project experience)

.consulting .solutions .partnership

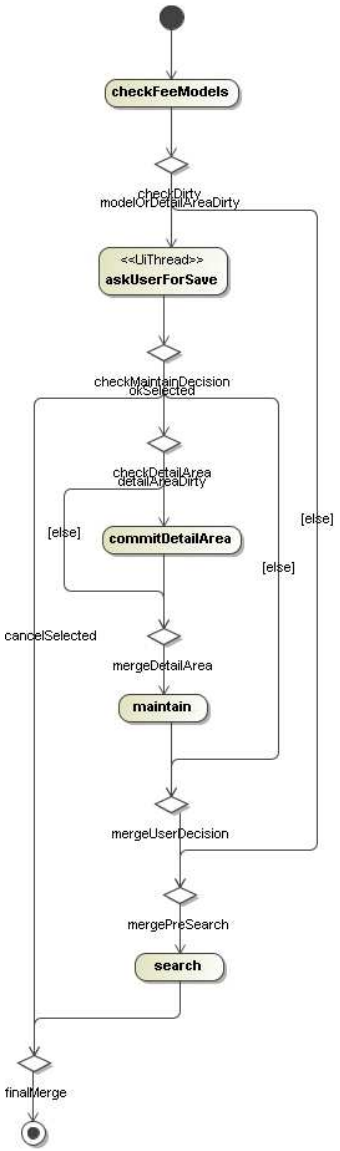


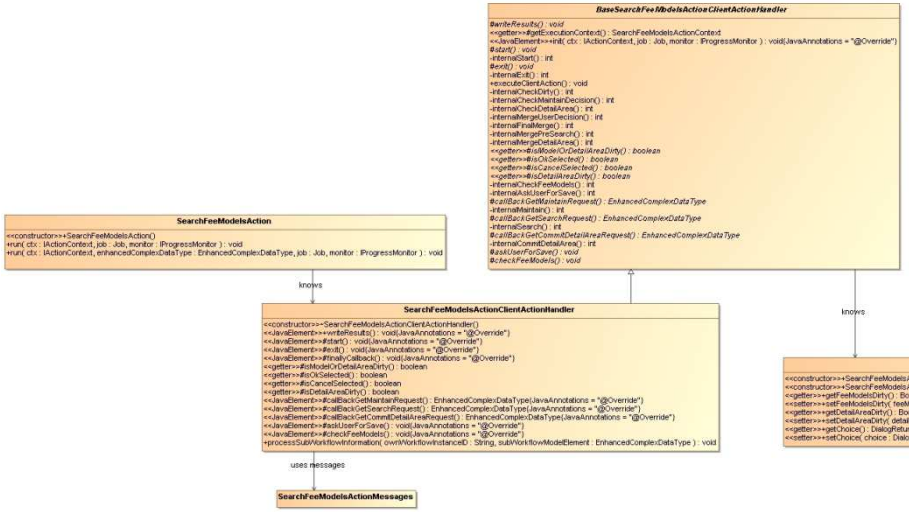
Improvement: GUI Designer (DSL for GUI)

.consulting .solutions .partnership



Behavior: Modeling (Project Experience)





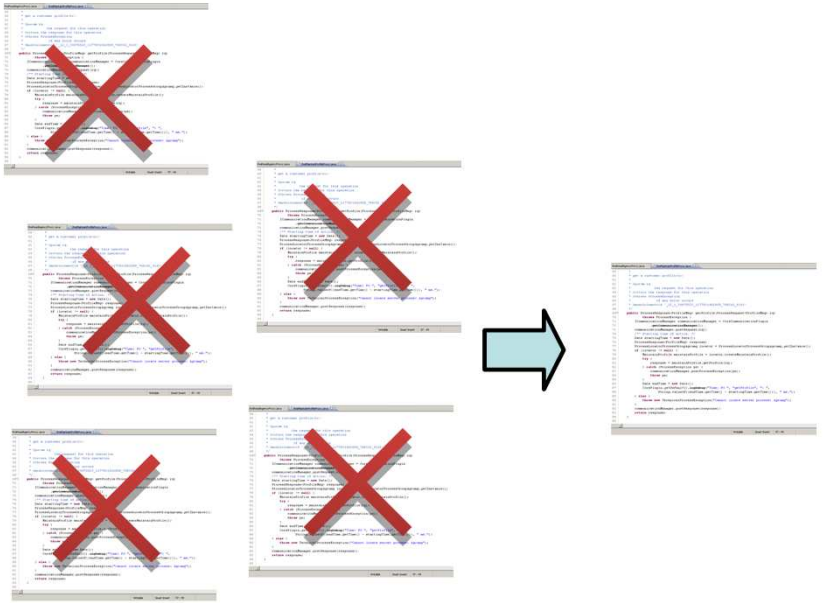
```
public final void executeClientAction() throws ClientActionException {
    try {
        while (clientActionStep != QUIT) {
            if (CorePlugin.getDefault().isDebugEnabled()) {
                CorePlugin.getDefault().logDebug("Executing ClientActionStep: ",
                        clientActionSteps[this.clientActionStep]);
            }
            switch (this.clientActionStep) {
            case CHECK_DIRTY: {
                this.clientActionStep = internalCheckDirty();
                break;
            }
            case CHECK_MAINTAIN_DECISION: {
                this.clientActionStep = internalCheckMaintainDecision();
                break;
            }
            case CHECK_DETAIL_AREA: {
                this.clientActionStep = internalCheckDetailArea();
                break;
            }
            case MERGE_USER_DECISION: {
                this.clientActionStep = internalMergeUserDecision();
                break;
            }
```

Improvement: Generics and manual Solutions

Manual Development of Business Logik

```
public final void executeClientAction() throws ClientActionException {
    checkFeeModels();
    if (isModelOrDetailAreaDirty()) {
        askUserForSave();
        if (isOkSelected()) {
            if (isDetailAreaDirty()) {
                commitDetailArea();
            }
            maintain();
            search();
        } else if (isCancelSelected()) {
        } else {
            search();
        }
    } else {
        search();
    }
}
```

Generic Solutions (DRY Principles)

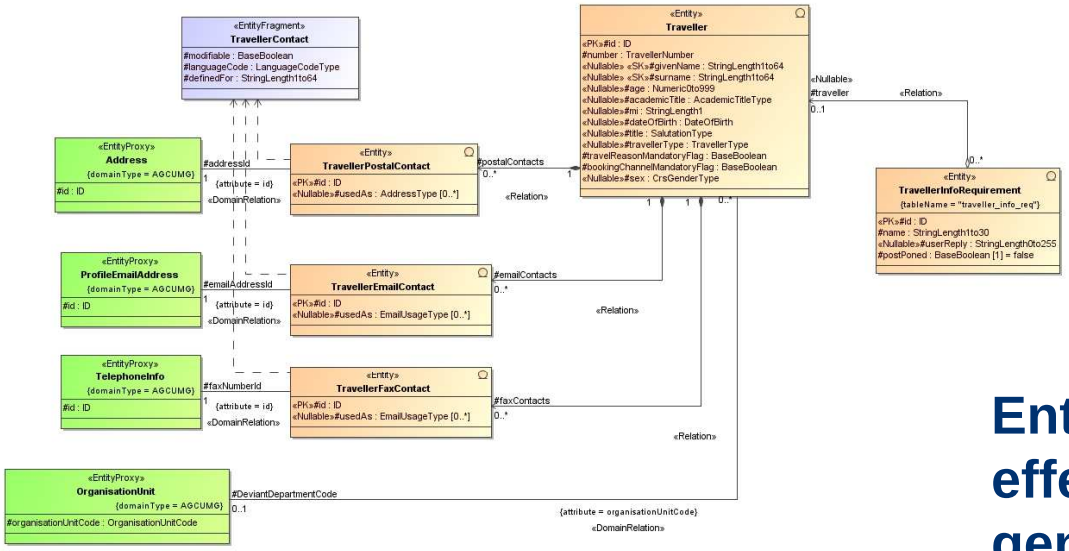


Java-internal DSL

```
bind.ui(form.getTextZIP()).property(path).property(AgencySearchCriterionPropertyType.ZIP);
bind.ui(form.getTextCity()).property(path).property(AgencySearchCriterionPropertyType.CITY)
        .converter(DummyConverter.getInstance());
bind.ui(form.getTextRegion()).property(path).property(AgencySearchCriterionPropertyType.REGION);
```

Entities: Modeling

.consulting .solutions .partnership



Entities and Interfaces can be effectively modeled and generated

Important Practical Hints

Important Hints (Lessons Learned)

1. Development and Maintenance of Generators takes time and costs money.
   Consider the costs for Modeling, Export and Generation in your projects.

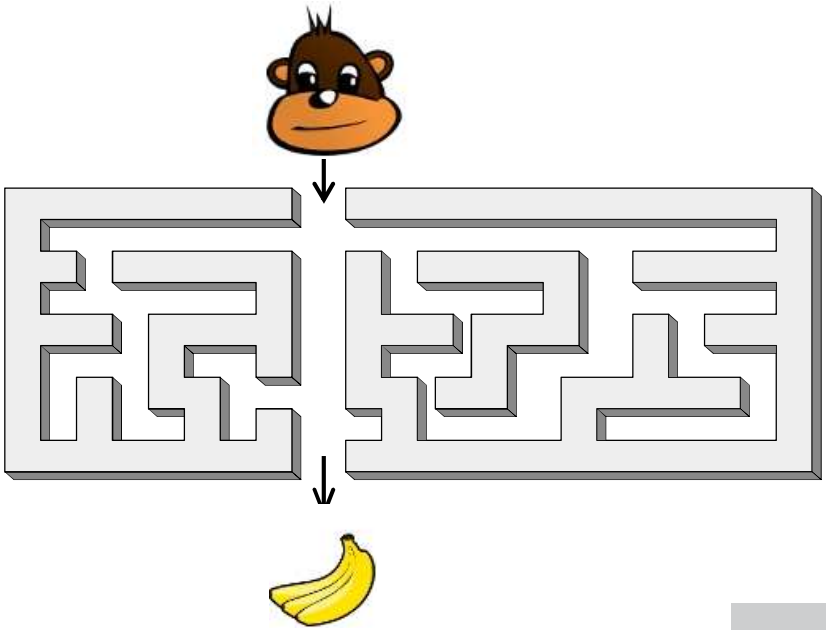
2. Good real *Reference Examples* (Application Prototypes) must be created and tested before the creation of *Generators*.

3. Separation between generated and manual code is necessary.
   „Protected Regions“ (Mix) is an Anti-Pattern.

4. Generated Code should be well structured and *well readable*, since the developers deal with source code.

5. Versioning (Releases, Branches) for Generators, Models and Source Code ist extremely important. A consistent and reasonable Release-Concept is absolutely necessary!

‚but you should know the right way



References

[Fo10] M. Fowler**,** Domain-Specific Languages, Addison-Wesley Professional, 2010

[Gr09] R. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, Addison-Wesley Longman, 2009

[VoSt06] M. Völter, T. Stahl**,** Model-Driven Software Development (Technology, Engineering, Management), Wiley 2006

[VuB04] M. Völter, J. Betting, Patterns for Model-Driven Development, EuroPLoP, 2004; (als PDF auch unter http://www.voelter.de/data/pub/MDDPatterns.pdf)

[oaW] http://www.openarchitectureware.org/

[mar] http://www.martinfowler.com/bliki/FluentInterface.html

[KrRuRo11] J. Krey, V. Rubin, S. Rose, Modellgetriebene Entwicklung einer Rich Client-Anwendung – ein Erfahrungsbericht, Objektspektrum, 2011

Vielen Dank für Ihre Aufmerksamkeit

Dr. Vladimir Rubin
GB Travel & Logistics
Mergenthalerallee 55-59
65760 Eschborn
Telefon: +49 176 24014405
vladimir@rubin-it.com

www.msg-systems.com

.consulting .solutions .partnership