Challenges in the Evolution of Metamodels: Smells and Anti-Patterns of a Historically-Grown Metamodel

ME@MODELS’16, 02.10.2016
Misha Strittmatter, Georg Hinkel, Michael Langhammer, Reiner Jung, Robert Heinrich

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS
Introduction

- Palladio Component Model (PCM)
- Language for component-based software architectures
- Analysis of quality attributes (initial focus performance)
Introduction

- Palladio Component Model (PCM)
  - Language for component-based software architectures
  - Analysis of quality attributes (initial focus performance)
Introduction

- Palladio Component Model (PCM)
  - Language for component-based software architectures
  - Analysis of quality attributes (initial focus performance)
Introduction: PCM

- First revision in the repository 2006
- Size more than doubled until 2012
- Added features: reliability, event communication, middleware infrastructure, resource interfaces, …
- Currently: 151 classes, 20 packages
Foundations

Metamodel

→ instance

Model
Foundations

- EMOF
  - Language for metamodels
  - PCM implemented in Ecore
Foundations

- **EMOF**
  - Language for metamodels
  - PCM implemented in Ecore
Foundations

- EMOF
  - Language for metamodels
  - PCM implemented in Ecore

- Smells
  - Indicators for potential issues
Foundations

- EMOF
  - Language for metamodels
  - PCM implemented in Ecore

- Smells
  - Indicators for potential issues
  - Code smells
  - Design smells
  - Metamodel smells
Foundations

- EMOF
  - Language for metamodels
  - PCM implemented in Ecore

- Smells
  - Indicators for potential issues
  - Code smells
  - Design smells
  - Metamodel smells
Foundations

- **EMOF**
  - Language for metamodels
  - PCM implemented in Ecore

- **Smells**
  - Indicators for potential issues
  - Code smells
  - Design smells
  - **Metamodel smells**
    - No semantic errors
    - No metamodel/validation errors
# Metamodel Smells

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
</tr>
<tr>
<td>Dead Class</td>
</tr>
</tbody>
</table>

**Containment (Metamodel Specific)**
# Metamodel Smells

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
</tr>
<tr>
<td>Dead Class</td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
</tr>
<tr>
<td>Package Dependency Cycles</td>
</tr>
</tbody>
</table>

- **Containment (Metamodel Specific)**
- **Modularization**
## Metamodel Smells

<table>
<thead>
<tr>
<th>Name</th>
<th>Containment (Metamodel Specific)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
<td></td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
<td></td>
</tr>
<tr>
<td>Dead Class</td>
<td>Modularization</td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
<td></td>
</tr>
<tr>
<td>Package Dependency Cycles</td>
<td></td>
</tr>
<tr>
<td>Dependency Inversion Principle Violated</td>
<td></td>
</tr>
</tbody>
</table>

- **Containment (Metamodel Specific)**
  - Redundant Container Relation
  - Obligatory Container Relation
  - Dead Class
  - Concern Scattered in Package Hierarchy
  - Multiple Concerns in Package
  - Package Dependency Cycles
  - Dependency Inversion Principle Violated

- **Modularization**

- **Dependency Direction**
## Metamodel Smells

<table>
<thead>
<tr>
<th>Name</th>
<th>Containment (Metamodel Specific)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
<td></td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
<td></td>
</tr>
<tr>
<td>Dead Class</td>
<td></td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
<td></td>
</tr>
<tr>
<td>Package Dependency Cycles</td>
<td>Modularization</td>
</tr>
<tr>
<td>Dependency Inversion Principle Violated</td>
<td>Dependency Direction</td>
</tr>
<tr>
<td>Concrete Abstract Class</td>
<td>Inheritance</td>
</tr>
<tr>
<td>Duplicate Features in Sibling Classes</td>
<td></td>
</tr>
<tr>
<td>Classification by Enum</td>
<td></td>
</tr>
</tbody>
</table>
Metamodel Smells

<table>
<thead>
<tr>
<th>Name</th>
<th>Containment (Metamodel Specific)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
<td></td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
<td></td>
</tr>
<tr>
<td>Dead Class</td>
<td></td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
<td>Modularization</td>
</tr>
<tr>
<td>Package Dependency Cycles</td>
<td></td>
</tr>
<tr>
<td>Dependency Inversion Principle Violated</td>
<td>Dependency Direction</td>
</tr>
<tr>
<td>Concrete Abstract Class</td>
<td></td>
</tr>
<tr>
<td>Duplicate Features in Sibling Classes</td>
<td>Inheritance</td>
</tr>
<tr>
<td>Classification by Enum</td>
<td></td>
</tr>
</tbody>
</table>
Why is this of interest?

- PCM good specimen of big, grown over time metamodel
Why is this of interest?

- PCM good specimen of big, grown over time metamodel
- Smells arising from metamodel evolution
- Smells impacting metamodel evolution
Why is this of interest?

- PCM good specimen of big, grown over time metamodel
- Smells arising from metamodel evolution
- Smells impacting metamodel evolution
- No exclusive focus on automatically detectable
Redundant Container Relation

Component Behavior \(\rightarrow\) Action \(\rightarrow\) Variable Usage

Misha Strittmatter – Smells and Anti-Patterns of a Historically-Grown Metamodel
Redundant Container Relation

Component Behavior \( \rightarrow \) Action \( \leftarrow \) Variable Usage

\( 0..1 \)
Redundant Container Relation

Component Behavior $\rightarrow^* \text{Action} \rightarrow^{0..1} \text{Variable Usage}$
Redundant Container Relation

- Consequence:
  - Redundancy
Redundant Container Relation

- Consequence:
  - Redundancy
  - Complexity (esp. with multiple containers)
Redundant Container Relation

Consequence:

- Redundancy
- Complexity (esp. with multiple containers)
- Unwanted dependencies between packages
Redundant Container Relation

- Consequence:
  - Redundancy
  - ↑ Complexity (esp. with multiple containers)
  - Unwanted dependencies between packages

- Reason:
  - Graphical editors
  - UML habit
Redundant Container Relation

- **Consequence:**
  - Redundancy
  - ⬆ Complexity (esp. with multiple containers)
  - Unwanted dependencies between packages

- **Reason:**
  - Graphical editors
  - UML habit
  - (Type safety)
Redundant Container Relation

Consequence:
- Redundancy
- ↑ Complexity (esp. with multiple containers)
- Unwanted dependencies between packages

Reason:
- Graphical editors
- UML habit
- (Type safety)

Correction:
- remove
- use eContainer()
Redundant Container Relation

Consequence:
- Redundancy
- Complexity (esp. with multiple containers)
- Unwanted dependencies between packages

Reason:
- Graphical editors
- UML habit
- (Type safety)

Correction:
- remove
- use eContainer()
Obligatory Container Relation

Component Behavior \[\xrightarrow{1} \] Action
Obligatory Container Relation

Additional consequence:
- No reuse!

![Diagram showing Component Behavior with a mandatory relationship to Action with no reuse allowed.](image)
Concern Scattered in Package Hierarchy

- repository
  - GenModel
  - PassiveResource -> Entity
  - BasicComponent -> ImplementationComponentType
  - ImplementationComponentType -> RepositoryComponent
  - RepositoryComponent -> InterfaceProvidingRequiringEntity
  - ProvidedRole -> Role
  - Parameter
  - DataType
  - Repository -> Entity
  - Interface -> Entity
  - RequiredCharacterisation
  - EventGroup -> Interface
  - EventType -> Signature
  - Signature -> Entity
  - ExceptionType
  - InfrastructureSignature -> Signature
  - InfrastructureInterface -> Interface
  - InfrastructureRequiredRole -> RequiredRole
  - RequiredRole -> Role
  - OperationSignature -> Signature
  - OperationInterface -> Interface
  - OperationRequiredRole -> RequiredRole
  - SourceRole -> RequiredRole
  - SinkRole -> ProvidedRole
  - OperationProvidedRole -> ProvidedRole
  - InfrastructureProvidedRole -> ProvidedRole
- ParameterModifier
  - CompleteComponentType -> RepositoryComponent
  - ProvidesComponentType -> RepositoryComponent
- ComponentType
  - CompositeComponent -> ComposedProvidingRequiringEntity
  - PrimitiveDataType -> DataType
- PrimitiveTypeEnum
  - CollectionDataType -> Entity, Entity
  - CompositeDataType -> Entity, Entity
  - InnerDeclaration -> NamedElement
  - Role -> Entity
Concern Scattered in Package Hierarchy

Repository
(Components & Interfaces)

- repository
  - GenModel
  - PassiveResource -> Entity
  - BasicComponent -> ImplementationComponentType
  - ImplementationComponentType -> RepositoryComponent
  - RepositoryComponent -> InterfaceProvidingRequiringEntity
  - ProvidedRole -> Role
  - Parameter
  - DataType
  - Repository -> Entity
  - Interface -> Entity
  - RequiredCharacterisation
  - EventGroup -> Interface
  - EventType -> Signature
  - Signature -> Entity
  - ExceptionType
  - InfrastructureSignature -> Signature
  - InfrastructureInterface -> Interface
  - InfrastructureRequiredRole -> RequiredRole
  - RequiredRole -> Role
  - OperationSignature -> Signature
  - OperationInterface -> Interface
  - OperationRequiredRole -> RequiredRole
  - SourceRole -> RequiredRole
  - SinkRole -> ProvidedRole
  - OperationProvidedRole -> ProvidedRole
  - InfrastructureProvidedRole -> ProvidedRole
  - ParameterModifier
  - CompleteComponentType -> RepositoryComponent
  - ProvidesComponentType -> RepositoryComponent
  - ComponentType
  - CompositeComponent -> ComposedProvidingRequiringEntity
  - PrimitiveDataType -> DataType
  - PrimitiveTypeEnum
  - CollectionDataType -> Entity, DataType
  - CompositeDataType -> Entity, DataType
  - InnerDeclaration -> NamedElement
  - Role -> Entity

Event Communication

Placeholder Components

DataTypes

Infrastructure Components
Concern Scattered in Package Hierarchy

Consequence:

- ↓ Understandability
- ↓ Package cohesion
- ↑ Package coupling
Concern Scattered in Package Hierarchy

- Consequence:
  - 🔄 Understandability
  - 🔄 Package cohesion
  - ↑ Package coupling

- Reason:
  - Intrusive extension
Concern Scattered in Package Hierarchy

Correction
Concern Scattered in Package Hierarchy

 Correction

![Diagram showing concern scattered in package hierarchy before and after correction.](image-url)
Concern Scattered in Package Hierarchy

Correction
Classification by Enum

```
Component
+CompType
```

<table>
<thead>
<tr>
<th>CompType</th>
</tr>
</thead>
<tbody>
<tr>
<td>«enum»</td>
</tr>
<tr>
<td>BusinessComp</td>
</tr>
<tr>
<td>InfrastructureComp</td>
</tr>
</tbody>
</table>

Classification by Enum

Component
+CompType

Atomic Component
Composite Component

CompType «enum»
BusinessComp
InfrastructureComp
Classification by Enum

Component
+CompType

Atomic Component
Composite Component

Consequence:
- ↓ External extensibility
Classification by Enum

```
Component
+CompType

Atomic Component
Composite Component
Mixed Component
```

CompType
«enum»
BusinessComp
InfrastructureComp

Consequence:

↓ External extensibility
Classification by Enum

Consequence:

- External extensibility
- Situationaly unused features
Classification by Enum

Correction 1: strategy pattern

Component

Constitution

Atomic

Composite

Domain

Business

Infrastructure
Classification by Enum

- Correction 2: external extension
Conclusion

Summary

- 4 of 10 metamodel smells presented
- Refactorings to correct smells
- Smells impair: maintainability, understandability, …
Conclusion

Summary

- 4 of 10 metamodel smells presented
- Refactorings to correct smells
- Smells impair: maintainability, understandability, …

Benefit

- Awareness of these smells is important, especially in metamodel evolution
- Fix smells as soon as possible (else: technical debt)
Conclusion

Summary
- 4 of 10 metamodel smells presented
- Refactorings to correct smells
- Smells impair: maintainability, understandability, …

Benefit
- Awareness of these smells is important, especially in metamodel evolution
- Fix smells as soon as possible (else: technical debt)

Future work
- Bundle scripts for detection
  - Update and extend EMF Refactor?
- Review/analyze further metamodels
- Document further smell types
## Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Containment (Metamodel Specific)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Container Relation</td>
<td></td>
</tr>
<tr>
<td>Obligatory Container Relation</td>
<td></td>
</tr>
<tr>
<td>Dead Class</td>
<td></td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
<td>Modularization</td>
</tr>
<tr>
<td>Package Dependency Cycles</td>
<td></td>
</tr>
<tr>
<td>Dependency Inversion Principle Violated</td>
<td>Dependency Direction</td>
</tr>
<tr>
<td>Concrete Abstract Class</td>
<td></td>
</tr>
<tr>
<td>Duplicate Features in Sibling Classes</td>
<td>Inheritance</td>
</tr>
<tr>
<td>Classification by Enum</td>
<td></td>
</tr>
</tbody>
</table>

- **Redundant Container Relation**: Indicates unnecessary repetition of container relations.
- **Obligatory Container Relation**: Shows required container relations.
- **Dead Class**: Represents classes that are no longer used.
- **Concern Scattered in Package Hierarchy**: Alerts to concerns spread across packages.
- **Multiple Concerns in Package**: Highlights numerous concerns within a single package.
- **Package Dependency Cycles**: Identifies circular dependencies between packages.
- **Dependency Inversion Principle Violated**: Points out violations of dependency inversion principle.
- **Concrete Abstract Class**: Indicates use of concrete classes in an abstract context.
- **Duplicate Features in Sibling Classes**: Shows duplication of features between sibling classes.
- **Classification by Enum**: Uses enumeration for classification.
BACKUP
Component Developers

Software Architect

System Deployer

Domain Expert

Ralf Reussner – The Palladio Approach Symposium on Software Performance, Munich, 2015
Component Developers

Software Architect

System Deployer

Domain Expert
Component Developers

Software Architect

System Deployer

Domain Expert

Ralf Reussner – The Palladio Approach Symposium on Software Performance, Munich, 2015
## Automatic Detection

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Redundant Container Relation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Obligatory Container Relation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Dead Class</strong></td>
<td></td>
</tr>
<tr>
<td>Concern Scattered in Package Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Multiple Concerns in Package</td>
<td></td>
</tr>
<tr>
<td><strong>Package Dependency Cycles</strong></td>
<td></td>
</tr>
<tr>
<td>Dependency Inversion Principle Violated</td>
<td></td>
</tr>
<tr>
<td>(Concrete Abstract Class)</td>
<td></td>
</tr>
<tr>
<td>(Duplicate Features in Sibling Classes)</td>
<td></td>
</tr>
<tr>
<td>(Classification by Enum)</td>
<td></td>
</tr>
</tbody>
</table>

**Bold Green** = reliable automatic detection  
(Green) = automatic indicator collection
Redundant Container Relation: Unwanted Package Dependencies
Redundant Container Relation: Unwanted Package Dependencies

```
<table>
<thead>
<tr>
<th>Action</th>
<th>*</th>
<th>Variable Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Dead Class

- **Conditions:**
  - No incoming containment
  - Superclasses not contained

- **Consequence:**
  - ↓ Understandability

- **Reason:** remains

- **Correction:** delete
Multiple Concerns in Package

Consequence:

- **↑** Complexity
- **↓** Cohesion
Multiple Concerns in Package

- Consequence:
  - ↑ Complexity
  - ↓ Cohesion
- Reason: convenience
Multiple Concerns in Package

- **Consequence:**
  - ↑ Complexity
  - ↓ Cohesion

- **Reason:** convenience

- **Correction:**
  - sub features
  - no sub features
Package Dependency Cycles

(only inheritance)
Package Dependency Cycles

Consequence:

- ↓ Understandability
- ↑ Coupling
- ↑ Change impact
- Difficult to modularize

(only inheritance)
Package Dependency Cycles

- Consequence:
  - ↓ Understandability
  - ↑ Coupling
  - ↑ Change impact
  - Difficult to modularize

- Reason: unawareness
Package Dependency Cycles

- **Consequence:**
  - ↓ Understandability
  - ↑ Coupling
  - ↑ Change impact
  - Difficult to modularize

- **Reason:** unawareness

- **Correction:**
  - Correct dependencies: reversion, move classes
  - Explicit dependencies between packages
  - Package dependency graphs
Dependency Inversion Principle Violated

Consequence:

- ↑ Coupling
- ↓ Understandability

Reason:

- Type safety
- Cardinality
- O(1) retrieval
Dependency Inversion Principle Violated

Correction:

```
A

K

data
```
Dependency Inversion Principle Violated

Correction:

![Diagram showing the correction of the Dependency Inversion Principle]
Dependency Inversion Principle Violated

Correction:

![Diagram showing correction examples]

or
Concrete Abstract Class

- Consequence:
  - ↓ Correctness
  - ↓ Understandability
- Reason: carelessness
- Correction: trivial
Duplicate Features in Sibling Classes

Consequence:

- Redundancy
Duplicate Features in Sibling Classes

- Consequence:
  - ⬆ Redundancy
- Reason: iterative extension

![Diagram showing two sibling classes with a parent class and two features labeled 'a']
Duplicate Features in Sibling Classes

- Consequence:
  - Redundancy

- Reason: iterative extension

- Correction:
  - pull up refactoring
  - pull up into abstract class