

A decorative graphic consisting of a series of overlapping circles in shades of gray, arranged in a horizontal line from left to right, with the leftmost circle being the darkest and the rightmost being the lightest.

Model-Driven Software Engineering using Specialization Patterns

Jukka Viljamaa

Jan 19, 2007

Department of Computer Science
University of Helsinki

Contents

- Part I: Introduction to Model-Driven Engineering (MDE)
- Part II: Specialization Patterns
- Part III: Examples
- Part IV: Discussion and Conclusion

History of Abstraction and Modeling in SE

- Languages *solution space
(implementation technology)*
 - machine code → assembly → C → OO
- Platforms
 - OS APIs → middleware and frameworks

- Modeling *problem space
(application domain)*
 - structured modeling → UML
 - domain-specific modeling languages
 - cross-cutting concerns
- Modeling tools
 - CASE → round-trip engineering → MDE

Current Challenges *[Schmidt, 2006]*

- **Modeling**
 - Model interchange format standardization
 - Model evolution
 - Support for arbitrary application domains
- **Engineering and implementation**
 - Multiple target platforms & their complexity and evolution
 - Deployment and configuration using XML
 - semantic gap between design intent and language
 - Round-trip engineering
 - most code still written manually
 - model and produced code should be kept in sync

What Kind of Modeling is MDE?

- Modeling is essential in SE, but...
- Great variety in what the models represent and how they are used
 - *“If I create a visualization of some part of a system, am I practicing MDE?”*

Domain-Specific Modeling

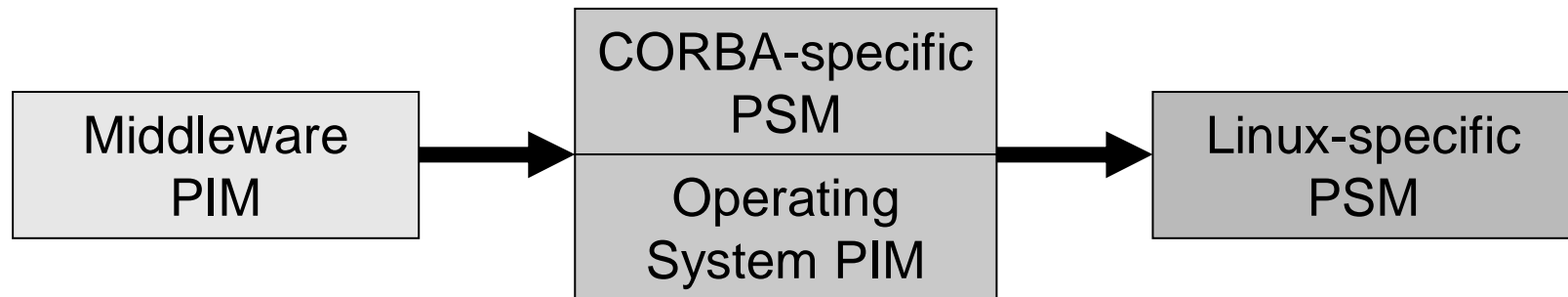
- Standard specification languages for describing the models
- Models can be tailored to accurately match the domain's vocabulary
- Metamodels
 - domain concepts and their relationships
 - semantics and constraints associated with the concepts

MDE Tools

- Are a unifying vehicle to document, analyze, and transform information systematically at many phases
- Generate code or a more specific model (semi-) automatically from a more abstract model
 - transformation generators analyze & synthesize models
- Impose domain-specific constraints and perform checks that can prevent errors early in the lifecycle
- Need not be complicated
 - they target higher-level systems (frameworks) instead of lower-level systems as in the past (e.g. OS APIs)

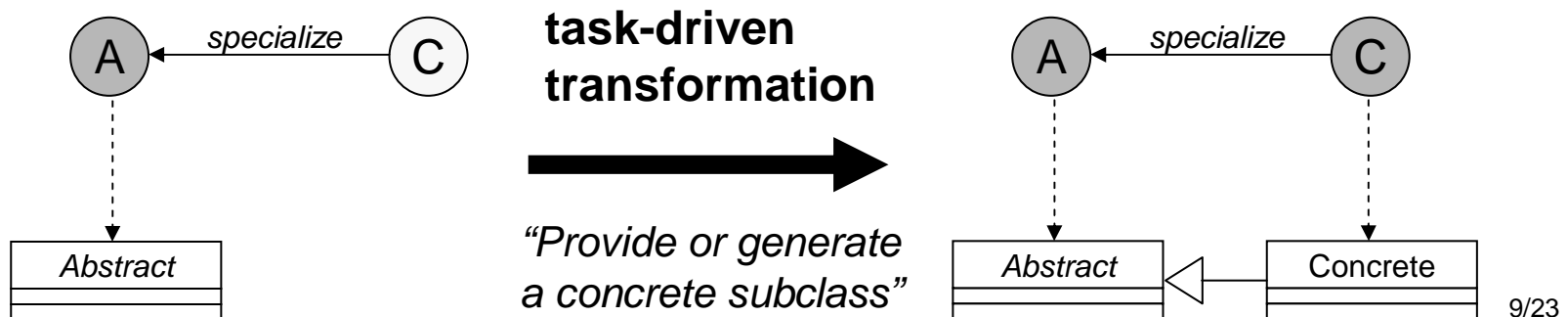
OMG's MDA Vocabulary

- Four kinds of models
 - ~~Computation Independent Model (CIM)~~
 - Platform Independent Model (PIM)
 - Platform Specific Model (PSM) described by a Platform Model (PM)
 - ~~Implementation Specific Model (ISM)~~
- Note: “platform” is always relative to a particular point of view



Specialization Patterns

- Descriptions of structural configurations of (modeling) elements
- Provide a mechanism to implement MDE
 - can be thought of as catalysts or specifications of transformations from a PIM to a PSM
- Consist of roles
 - *input roles* bound to source model before transformation
 - *output roles* depend on input roles and generate the target model during the transformation

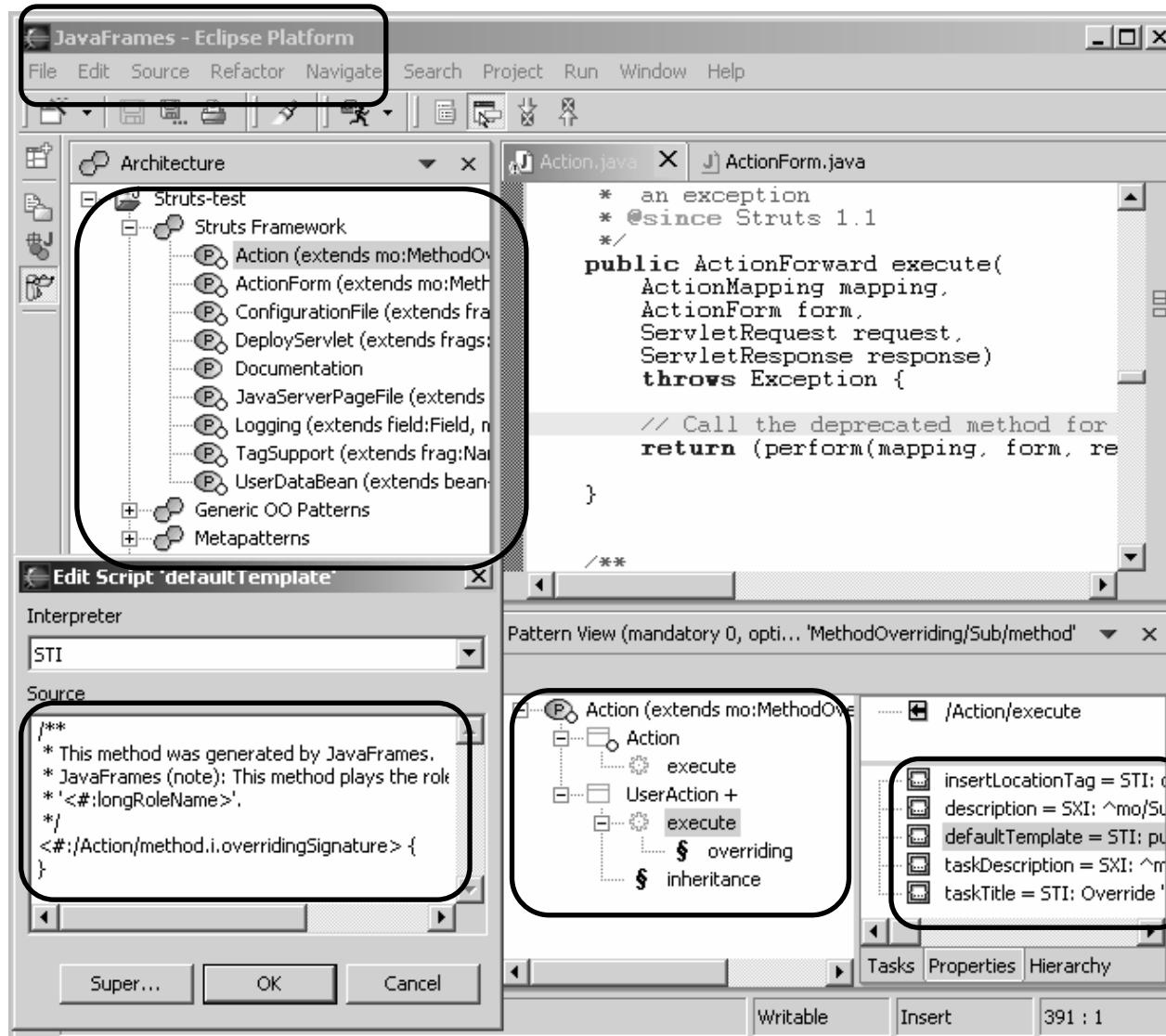


Roles, Role Instances, Tasks

- Roles are bound to concrete software systems (e.g. UML model or source code) through *role instances*
- A role has
 - a *type* that determines the kind of elements they can be bound to (e.g. UML class role or Java operation role)
 - a set of *constraints* (depending on type), constraints can also refer to other roles
 - a set of *scripts* to enable code generation
- Interactive role binding through *tasks*
 - unbound role instances & constraint violations generate tasks

Specialization Patterns

Tool Support: JavaFrames



Specialization Patterns

Tool Support (cont'd): MADE

The screenshot displays the Eclipse IDE interface for a project named 'JavaFrames'. The main workspace is divided into several views:

- Architecture View (Left):** Shows a project structure with 'Instances' (FileAccess (extends Security)), 'Patterns' (Security), and 'Extended Patterns' (Security: Aspectual pattern for managing system security at different levels of abstraction).
- UML Diagram (Center):** A class diagram showing 'BasePermission' and 'AbstractSecurityManager' as abstract classes. 'Permission' inherits from 'BasePermission', and 'SecurityManager' inherits from 'AbstractSecurityManager'. 'SecurityManager' has a method 'checkPermission (arg : Permission)'.
- Code Editors (Center):** Shows the source code for 'SecurityManager.java' and 'Permission.java'. The 'SecurityManager.java' code is visible, showing the abstract class definition and the 'checkPermission' method.
- Descriptor.xml (Center):** Shows the XML representation of the model, including the 'security' element and the 'strategy' element.
- Diagramview (Bottom Left):** Shows the 'Main' diagram.
- XML Parser View (Bottom Left):** Shows the 'Parse required' status.
- Pattern View (Bottom Center):** Shows the 'Pattern View (mandatory 0, optional 2) - 'FileAccess' extends 'Security'' with a tree structure: 'FileAccess (extends Security)' containing 'XML Descriptor.xml', 'Permission', and 'SecurityManager'. A box labeled 'Pattern instance' points to this view.
- Palette (Right):** Shows the 'Palette' with options like 'Select', 'Note', 'UML Common', 'Use Case Diag...', 'Composite Str...', 'Deployment Di...', 'Component Di...', 'Class Diagram', and 'Createable Cha'.
- Tasks (Bottom Right):** Shows a list of tasks: 'Provide a concrete security manager' and 'Provide a new resource permission'. A box labeled 'Tasks' points to this list. Below the tasks is a 'Task description' box containing the text 'Concrete security manager implementation.'.

Annotations on the left side of the image point to various components:

- Pattern library:** Points to the 'Patterns' section in the Architecture view.
- Pattern description:** Points to the 'Extended Patterns' section in the Architecture view.
- UML-specific tool:** Points to the UML Diagram view.
- XML-specific tool:** Points to the XML Parser View.

Annotations on the right side of the image point to various components:

- Editors for: UML Java XML:** Points to the code editors for 'SecurityManager.java' and 'Permission.java'.
- Tasks:** Points to the list of tasks in the bottom right.
- Task description:** Points to the 'Concrete security manager implementation.' text in the bottom right.

Specialization Patterns

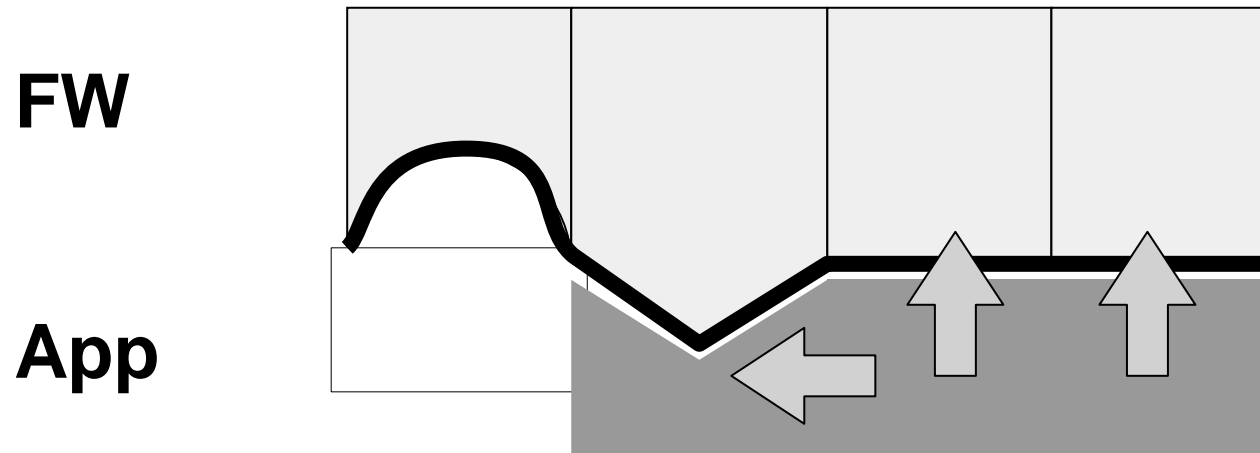
Applications

- UML model transformations
 - from UML to UML & from UML to code
 - MADE & INARI [*Hammouda et al., 2004*]
- Framework usage
 - from abstract code (FW) to concrete (app) code
 - JavaFrames [*Hakala et al., 2001*], [*Viljamaa, 2004 & 2006*]
 - Design fragments [*Fairbanks, Garlan, Scherlis, 2006*]
 - an approach very similar to specialization patterns
- Web service development
 - JavaFrames & MADE [*Jiang, Ruokonen, Systä, 2005*]

Example 1

[Viljamaa, 2004] & [Viljamaa, 2005]

Framework-Based SW Development

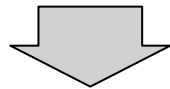


reuse interface = a collection of hot spots or variation points that enable specialization of FW, e.g. by subclassing (*white-box reuse*) or by combining and customizing ready-made components (*black-box reuse*)

Example 1 (cont'd)

Framework Usage Problems

- FWs are abstract, complex, large systems
- Implementation languages don't fully support FW-based development
- Typical FW documentation consists only of informal descriptions and small sample apps



- Locating hot spots & understanding dependencies?
- Steep learning curve, high training costs

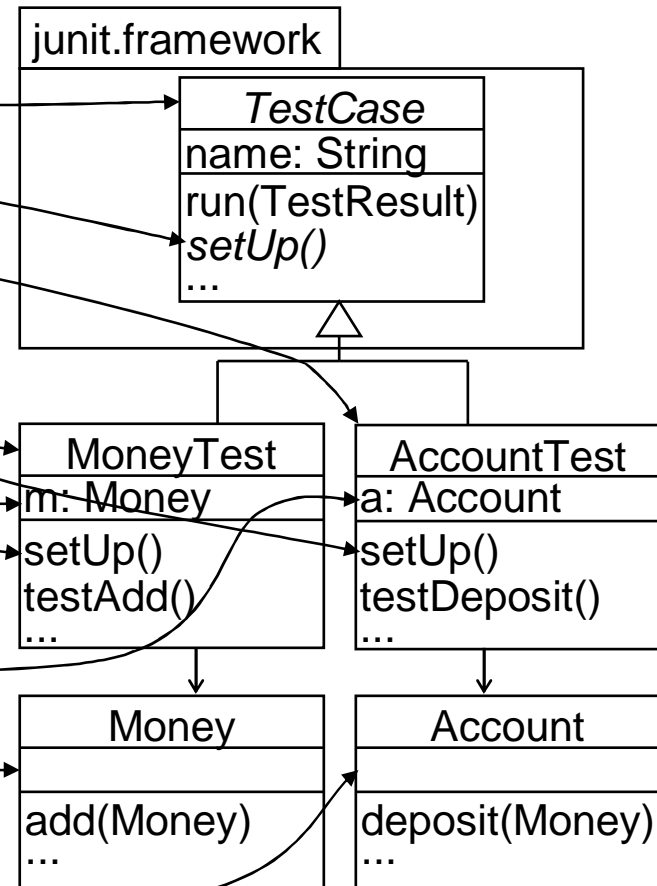
Example 1 (cont'd)

Task-Driven FW Usage Assistance

specialization pattern

```
class TestCase {  
  method setUp {}  
}  
class UserTestCase* {  
  inherits TestCase;  
  method setUp {  
    overrides TestCase.setUp;  
    fragment fixtureCreation {  
      sets fixtureAttr;  
    }  
  }  
  field fixtureAttr* {  
    isTypeOf FixtureClass;  
  }  
}  
class FixtureClass* {}
```

UML, Java, etc.



Example 1 (cont'd)

JavaFrames

- Provides
 - context-sensitive and adjusting documentation
 - parameterized code generation
 - validation of app code against the FW requirements
- Implemented as an Eclipse perspective

Reuse in Web Service Development

- Traditionally
 - reuse as web service composition (building services from other services)
 - e.g. choreography languages
- Our approach adds to this
 - reuse of WSDL descriptions
 - validation of rules defined for WSDL descriptions (e.g. Basic Profile 1.1)
 - Reuse in service implementations

Example 2 (cont'd)

Modeling Variation in WS Development

	Variation points	Variation mechanisms
WSDL document	<ul style="list-style-type: none">• Service name & address• SOAP action string• WSDL character encoding• Namespaces• Array declarations• Binding to transportation• ...	<p><i>WSDL2Code</i></p> <ul style="list-style-type: none">• Parameterized WSDL template <p><i>Code2WSDL</i></p> <ul style="list-style-type: none">• Parameterized code generation
Service endpoint	<ul style="list-style-type: none">• Names and types of operations and their parameters	<p><i>WSDL2Code</i></p> <ul style="list-style-type: none">• Managed in WSDL design <p><i>Code2WSDL</i></p> <ul style="list-style-type: none">• Tool guides framework specialization
Business logic	<ul style="list-style-type: none">• Depends on the service framework in question	<ul style="list-style-type: none">• A framework and tool support for its specialization

Example 2 (cont'd)

Solution: Tool-Assisted WS Development

- JavaFrames & MADE provide
 - context-sensitive and adjusting documentation
 - parameterized UML model and Java code generation
 - validation against WSDL & Basic Profile 1.1 specifications
- Integrated to Eclipse, Rational Rose & Rational Software Architect

Conclusion

- MDE (automatic model transformations and code generation) is becoming increasingly important in SE
- Specialization patterns provide a viable way to implement task-driven transformations
- Empirical evidence shows the benefits of tool support (at least in the context of framework specialization) *[Fairbanks, Garlan, Scherlis, 2006]*

Questions?

- Thank you!
- <http://practise.cs.tut.fi> (downloads)

References

- Fairbanks G., Garlan D., Scherlis W., Design Fragments Make Using Frameworks Easier. In: Proc. *OOPSLA 2006*.
- Hakala M. et al., Annotating Reusable Software Architectures with Specialization Patterns. In: Proc. *WICSA 2001*.
- Hammouda I. et al., Adaptable Concern-Based Framework Specialization in UML. In: Proc. *ASE 2004*.
- Jiang J., Ruokonen A., Systä T., Pattern-Based Variability Management in Web Service Development. In: Proc. *ECOWS 2005*.
- Schmidt D., Model-Driven Engineering. *IEEE Computer*, Feb 2006.
- Viljamaa A., Specifying Reuse Interfaces for Task-Oriented Framework Specialization. PhD Thesis, 2006.
- Viljamaa J., Applying Formal Concept Analysis to Extract Framework Reuse Interface Specifications from Source Code. PhD Thesis, 2004.