



UML Model Transformations for Quality and Correctness

Andrea Baruzzo
e-mail: baruzzo@dimi.uniud.it

*Dipartimento di Matematica e Informatica
Università degli Studi di Udine*

1



Agenda

- Approaches for UML Quality Assurance
- Model Transformations
- Model Transformations for Quality
- Model Transformations for Correctness
- Projects for Thesis

2



Methods for UML Model Quality Assurance

- ❑ Different aspects of a system covered by different types of diagrams
- ❑ **Constructive** methods vs. **Analytical** methods:
Error **Prevention** or **Detection**?
- ❑ Both!
 - ❑ Analytical with **inconsistency detection**
 - ❑ Constructive with the application of **good design principles** and the identification of “critical model patterns” supporting them
- ❑ Quality is more than correctness!




3



Techniques and Tools for Model Quality Assurance

- ❑ Automated Model Analysis is useful for both
 - ❑ Model Validation
 - ❑ Model Verification
- ❑ **Validation** checks the model against the (*explicit or latent!*) needs of the user (customer, stakeholder,...)
 - ❑ Model execution simulations
 - ❑ Visual debuggers (model-level debugging)
- ❑ **Verification** checks the conformmness of the model with respect to selected *quality attributes*
 - ❑ Consistency
 - ❑ Correctness
 - ❑ Software design principles

4



Model reviews


- Walkthroughs**
 - Participants: model author and (usually) domain experts or project leaders (stakeholders responsible for quality)
 - The author inspect the model, describe the architecture
 - ... and mimics its behavior simulating the execution of the dynamics
 - ... eventually gathering feedback from the participants
- Inspections**
 - Manual: using checklist
 - Automated: tool support

Validation - oriented

Verification - oriented

My work is aimed essentially to provide this type of tool support

5



Models and Model Transformations

- A **model** is a **simplified representation** of a part of the world named “the system” [Seidewitz, 2003]
- A model is useful if it helps to:
 - better understand** the system (or the source code)
 - decide the **appropriate actions** needed to reach and maintain the system’s goals
- Source code is a model too!
- Problems:
 - Model and code **synchronization**
 - Abstraction vs. implementation** details
- Partial solution: MDA and Model Transformations

Maintain synchronized two different things!

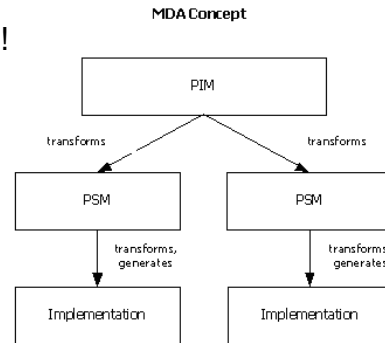
Different levels of reasoning

6



MDA (Model-Driven Architecture)

- ❑ MDA is a *process architecture*!
- ❑ MDA prescribes specific output (models!) for each software development phase
- ❑ The transition from one phase to the next is performed by *automatic transformations*
- ❑ Reduced manual synchronization efforts
- ❑ *Automatic consistency management* between the hierarchy of models



7



MDA, tools support, and definitions...

- ❑ MDA is more a vision than a reality
 - ❑ Lacking of proper tool support...
- ❑ Need of a clear definition for transformation-related concepts
- ❑ **Transformation**: automatic generation of a target model from a source model, according to a t. definition
- ❑ **Transformation definition**: set of t. *rules* that together describe how a model in the source language can be transformed into a model in the target language
- ❑ **Transformation rule**: a description of how one or more constructs in the source language can be transformed in one or more construct in the target language

8

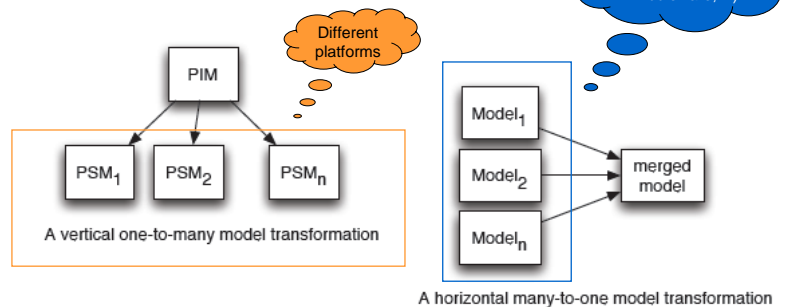
A Taxonomy of Model Transformations

- Q1. Which model transformation **approaches** exists?
- Q2. Which approach is **more appropriate** for a particular problem?
- Transformation *taxonomy* is useful to answer to Q1 and Q2
- The **taxonomy** proposed [Mens&VanGorp, 2005] investigate crucial questions:
 - What needs to be *transformed into what*?
 - What are the *important characteristics* of a model transformation?
 - What are the *success criteria* for a transformation language or tool?
 - What are the *quality requirements* for such languages or tools?
 - Which *mechanism* can be used for model transformations?

9

Question: What needs to be transformed into what?

- Program** transformations vs. **Model** transformations
- Number of source and target models
 - Transformation of **multiple source** models
 - Transformation for **multiple target** models

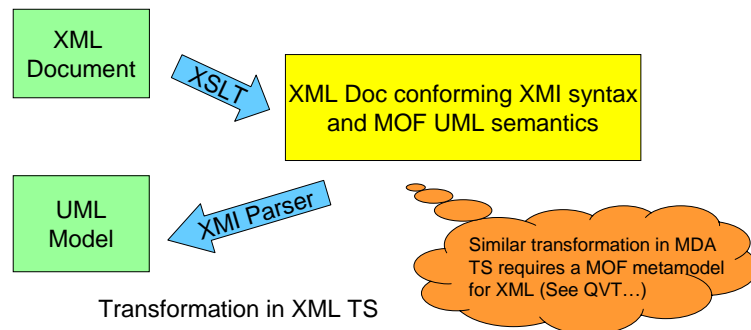


10

Technological Spaces

- ❑ A **Technological space** is determined by the **metamodel** to be used:

- ❑ **XML TS** (XML schema; HTML, XML, XSLT...)
- ❑ **MDA TS** (MOF; UML)



11

Endogenous vs. Exogenous transformations

- ❑ **Exogenous**: t. between models expressed in different languages (**translation**)
 - ❑ *Synthesis* of high-level models (PIM) into implementation models (i.e. Java Program). Also called Code generation
 - ❑ *Reverse Engineering* (the inverse of synthesis)
 - ❑ *Migration* from a program in one language to another, but keeping the same level of abstraction
- ❑ **Endogenous**: t. between models expressed in the same language (**rephrasing**)
 - ❑ *Optimization* (improve certain operational qualities preserving the semantics)
 - ❑ *Refactoring* (change internal structure without changing observable behavior)
 - ❑ *Simplification* and *Normalization* (decrease syntactic complexity)

12



Horizontal vs. Vertical transformations

- ❑ **Horizontal:** source and target model reside at the same abstraction level
 - ❑ *Refactoring* (also endogenous)
 - ❑ *Migration* (also exogenous)
- ❑ **Vertical:** source and target model reside at different levels of abstraction
 - ❑ *Refinement* (a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details)
- ❑ Exogenous/Endogenous and Horizontal-Vertical are orthogonal dimensions!

13

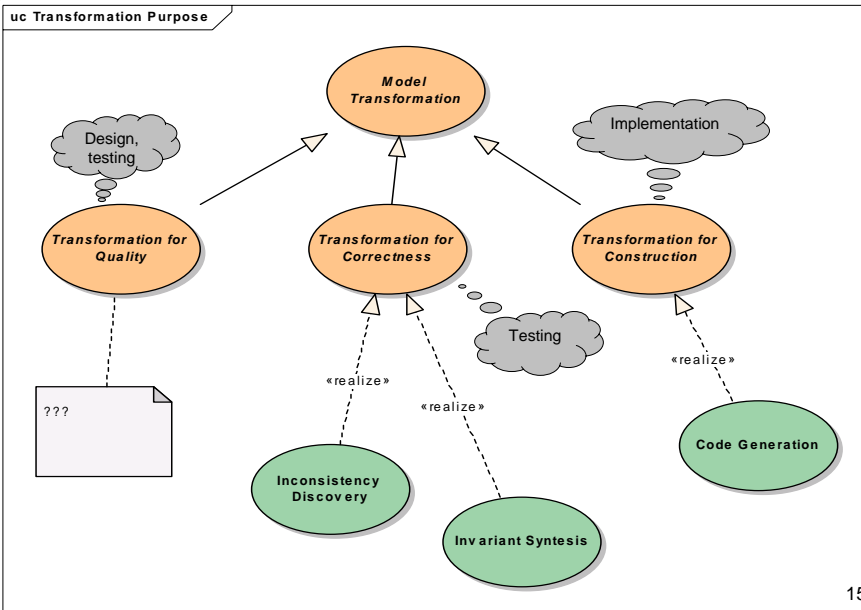


A missed dimension?

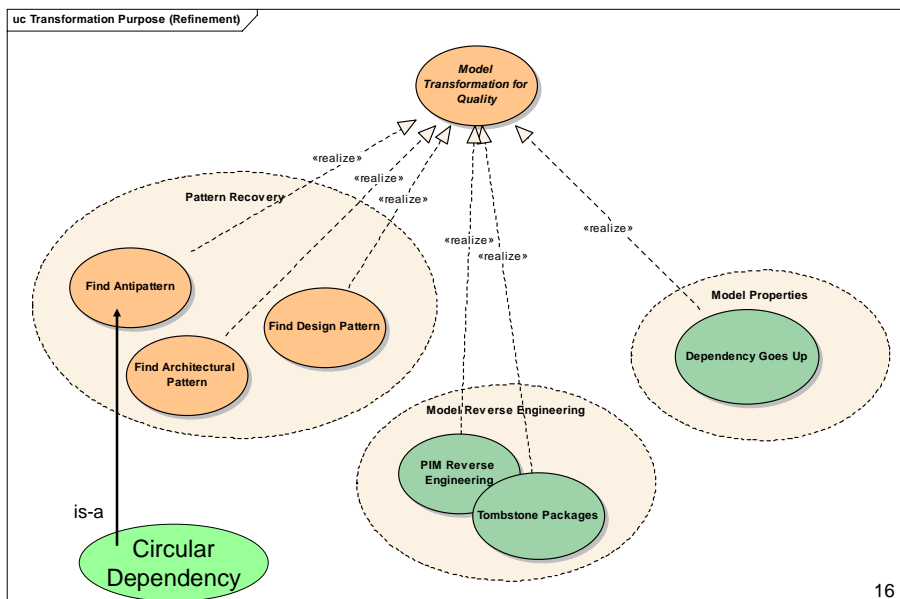
- ❑ The proposed taxonomy seems to not consider an important aspect: the **purpose** of the t.
- ❑ **Purpose** is more useful in order to categorize t. for a user-oriented perspective
 - ❑ Natural mapping with functional specifications
 - ❑ More relevant in order to choose tools
 - ❑ Reflect classical software engineering perspective (important activities in the software life cycle)
- ❑ We propose a new **dimension** called "Purpose"
 - ❑ It is orthogonal to all other levels in the taxonomy

14

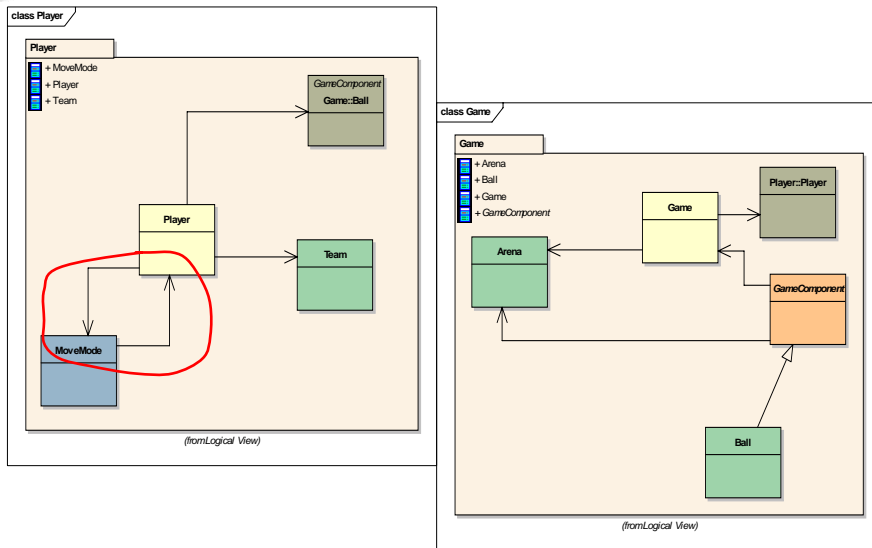
Purpose Level of Abstraction



Transformations for Quality

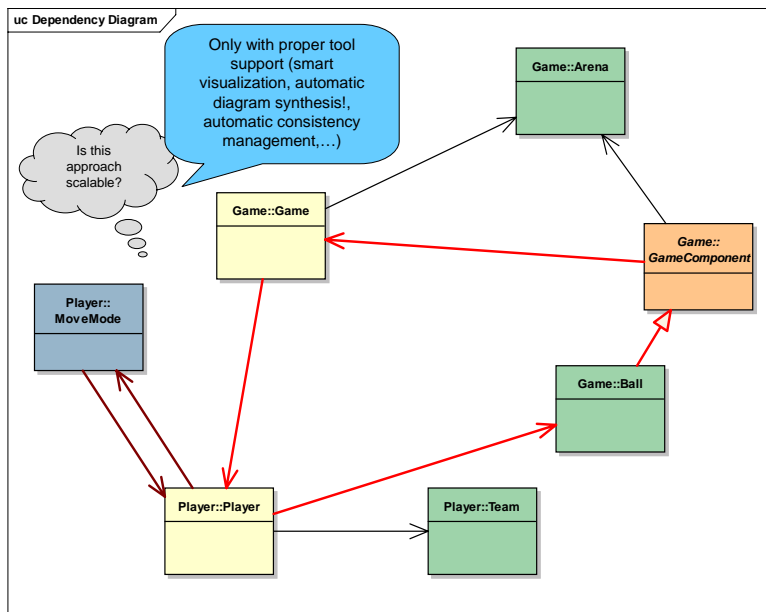


Circular Dependency (Antipattern)



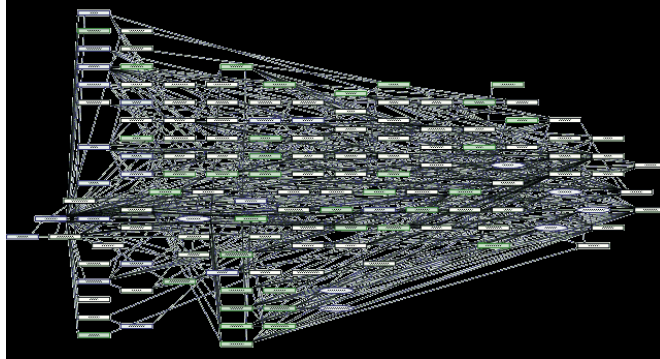
17

Circular Dependency (full diagram)



18

Don't Expect Miracles!

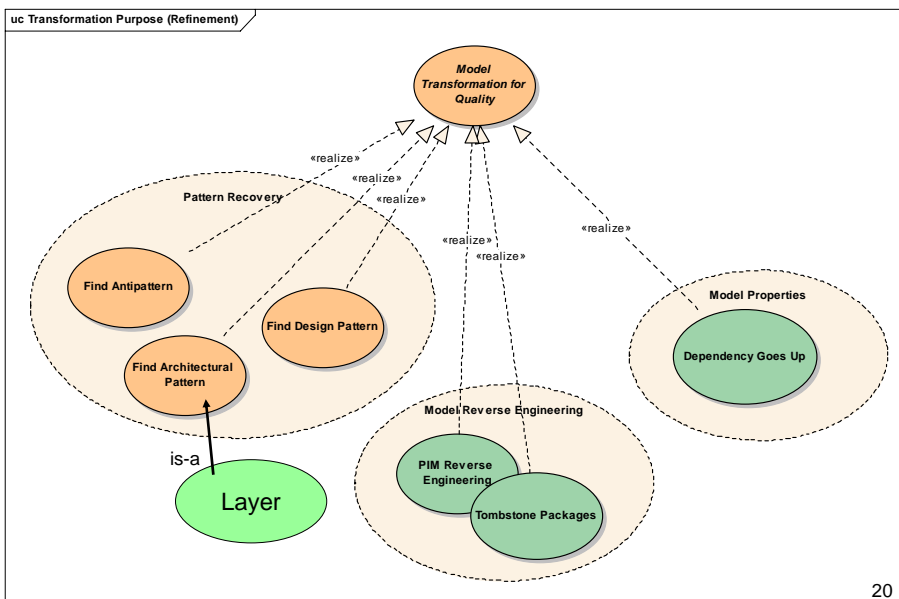


File dependency diagram of a 70,000-line program (138 classes, 470 relations)

- ❑ The tool should:
 - ❑ keep *hidden* the full graph dependency
 - ❑ extract *only relevant dependencies* from the dependency graph in order to reveal the "pattern" we are looking for

19

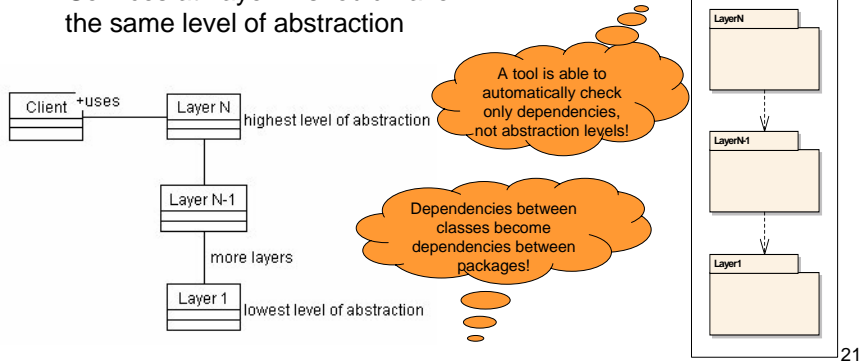
Find Architectural Patterns



20

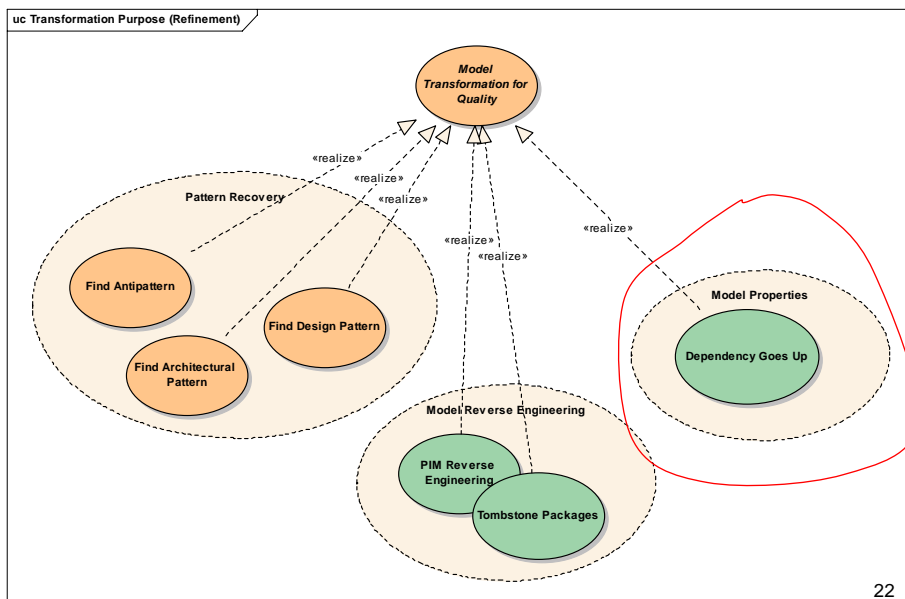
Layer [POSA1]

- ❑ **Intent:** helps to structure applications that can be *decomposed* into *groups of subtasks* in which each group of subtasks is at a *particular level of abstraction*
- ❑ **Structure:**
 - ❑ Layer N's services may depend on other services in Layer N or N-1
 - ❑ Dependencies between Layer N and Layer N-2 are forbidden
 - ❑ Services at Layer N should have the same level of abstraction



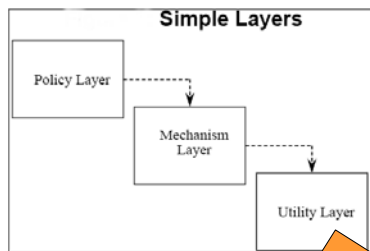
21

Find good Model Properties (inspired by good principles)



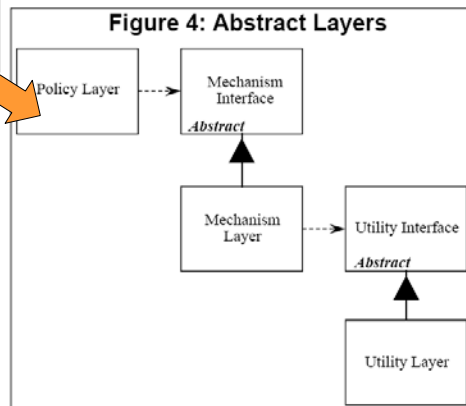
22

Stratification and Dependency Inversion Principle (DIP)



- ❑ It has the **insidious characteristic** that the Policy Layer (more abstract) is sensitive to changes all the way down in the Utility Layer (more concrete)!
- ❑ **Abstractions** should be **more stable** than concretizations!

- ❑ Each of the lower level layers are represented by an abstract class. The actual layers are then derived from these abstract classes
- ❑ Each of the higher level classes uses the next lowest layer through the abstract interface



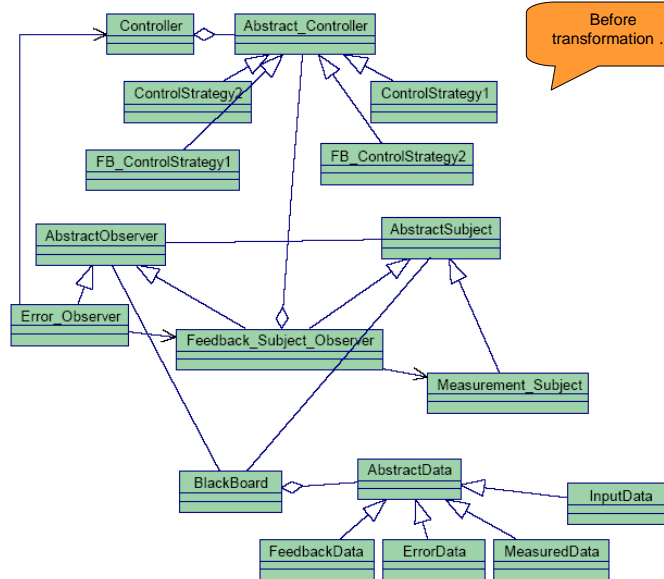
23

A generalized form of DIP

- ❑ DIP decomposes systems by means of abstractions
- ❑ What if we apply DIP to *every dependency* (rather than only to generalization)?
- ❑ Abstract things should not depend from concrete things
- ❑ General (more reusable) things should not depend from specific things
- ❑ Observation: a composite class is less general (reusable) than its components!

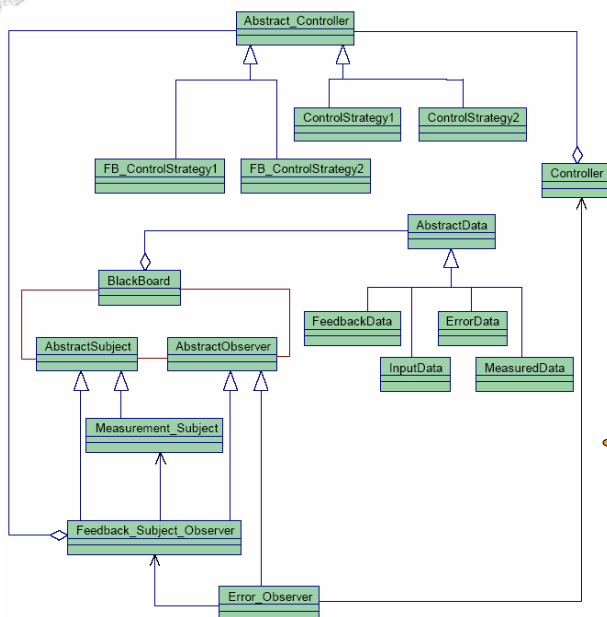
24

Dependency Goes Up (Carlo Pescio)



25

Dependency Goes Up (after transformation)



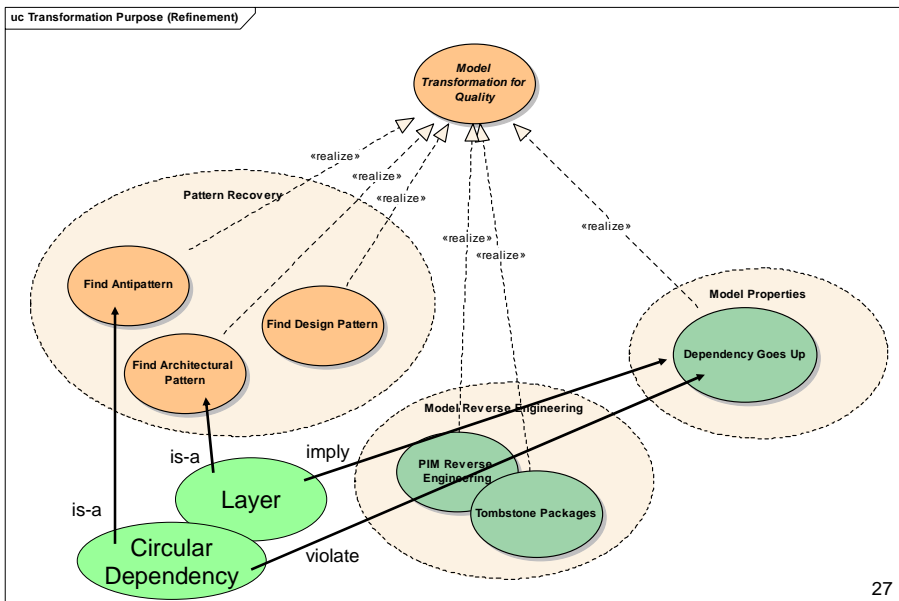
Useful for identify circular dependencies at glance!

Abstraction layers identification cannot be automated!

Not all people perceive this layout as natural, especially with composition.

26

Relation between Patterns and Good Principles



Transformations for Correctness

- Verify the dynamic view* of the system against the static view and its constraints (“software contracts”)
- Identify and *refine specifications* too strong
- Identify *new constraints* (specifications too weak?)
- Build *precise UML models*
 - Class diagrams
 - Sequence diagrams
 - Statechart diagrams
 - (OCL) specifications (the software contract)
- Independence from the specification language* (OCL, Promela, Alloy,...)

28

The notion of class correctness

Definition 1 (Meyer). A class C is correct with respect to its specification if

- For any set of valid arguments e_1, \dots, e_n to a creation procedure p :

$$\{ \text{Default}_C \wedge \text{Pre}_p[\vec{x}/\vec{e}] \} p \{ \text{Post}_p[\vec{x}/\vec{e}] \wedge \text{Inv}_C \}$$
- For every public method m and any set of valid arguments e_1, \dots, e_n :

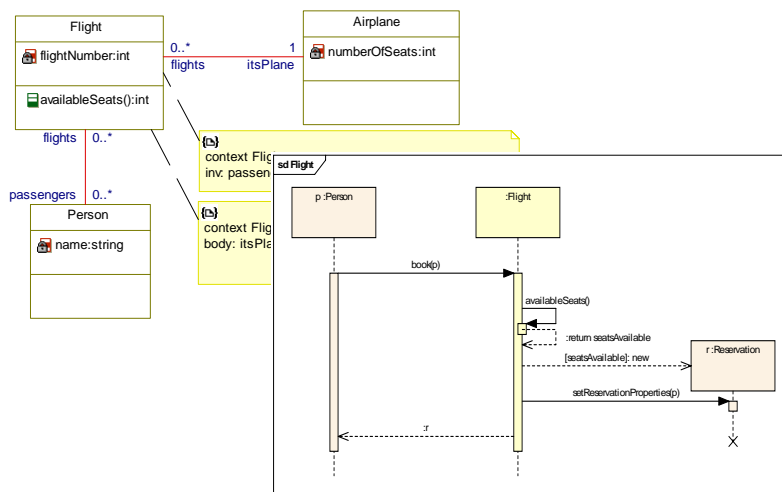
$$\{ \text{Pre}_m[\vec{x}/\vec{e}] \wedge \text{Inv}_C \} m \{ \text{Post}_m[\vec{x}/\vec{e}] \wedge \text{Inv}_C \}$$

where Default_C denotes the assertion expressing that the attributes of C have the default values of their type.

- ❑ But *what happens when this does not hold?* Ab1
 - ❑ Faulty Program or Inconsistent Specification?
- ❑ Due to the size of most systems, bugs in assertions are not so unlikely!

29

The Starting point: an Object Model and its specifications



30

Diapositiva 29

Ab1 This notion clearly states what has to happen when we call a method in a state which satisfies $\text{Prem } [x/e] \wedge \text{InvC}$, but what happens when this does not hold? As already said, failure to meet any of the responsibilities stated in the contract results in a break of the contract, and indicates the existence of a bug somewhere in the design or implementation of the software or in the assertions themselves. Due to the size of most systems, the latter chance is not so unlikely.

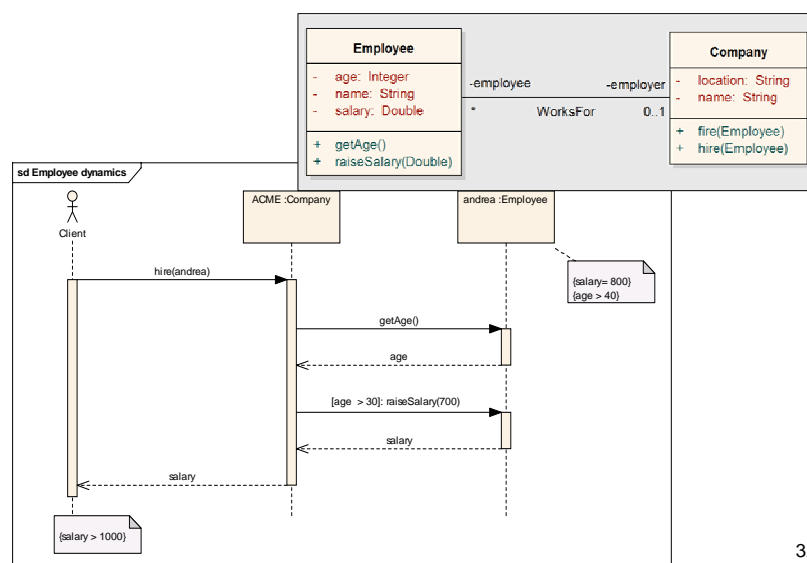
Andrea Baruzzo; 08/03/2007

Our method for Static Verification of UML model consistency – The process (BEDAV) [MoDeV²a06]

- ❑ *Build* the UML model of the system
 - ❑ Build the structure view
 - ❑ Build the dynamic (behavioral) view
- ❑ *Enrich* the model with the (OCL) specifications
- ❑ *Decompose* sequence diagrams in blocks
- ❑ *Annotate* each block with formulas to be imposed and to be checked
- ❑ *Verify* sequence diagrams against the formulas of each block

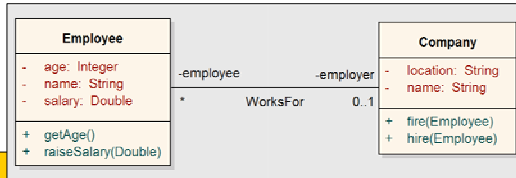
31

The method at work – the model



32

The method at work – the (OCL) specifications



context Company

```
inv: self.employee->size() =
      self.employee->asSet()->size()
```

context Employee

```
inv: (self.age >= 18)
```

pre

pre

post

context Employee::getAge() : Integer

```
pre: true
```

pre

post

```
post: (result = self.age)
```

context Employee::raiseSalary(amount : Double) : Double

```
post: (self.salary = (self.salary@pre + amount))
```

```
post: (result = self.salary)
```

33

Ab3

The method at work - decomposition

We need to impose that

$$\Phi_C = \text{result}(\Phi_A) \wedge \text{Post}_{m_1}[\vec{x}/\vec{e}]$$

$$\Phi_A = \Phi_{\text{link}(A)}$$

$$\Phi_D = \Phi_B \wedge \text{Post}_{m_1}[\vec{x}/\vec{e}]$$

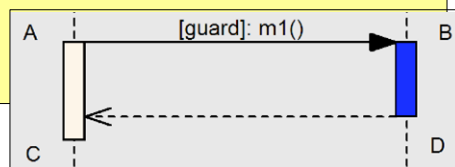
$$\Phi_B = \Phi_{\text{link}(B)}$$

and check that

$$\Phi_A \wedge \text{guard} \wedge \Phi_B \implies \text{Pre}_{m_1}[\vec{x}/\vec{e}]$$

$$\Phi_D \implies \text{Inv}_Y$$

$$\Phi_C \implies \text{Inv}_X$$



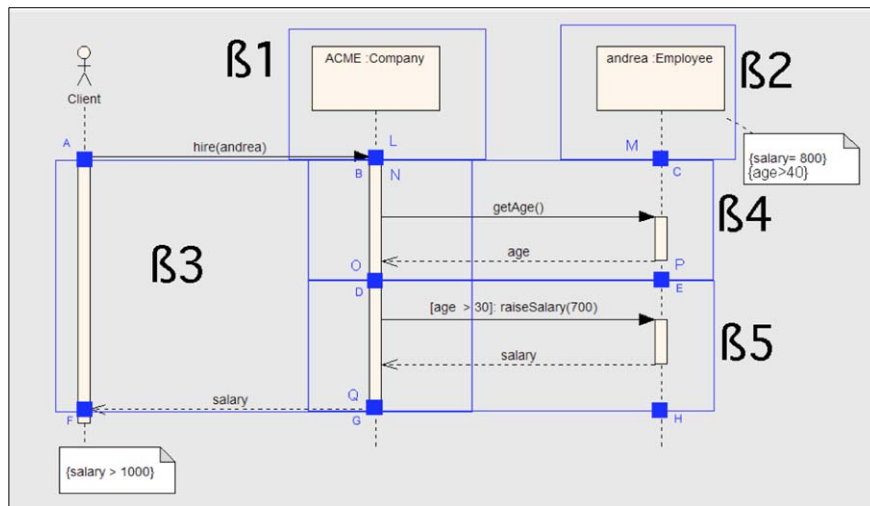
34

Diapositiva 34

Ab3 per ciascuno di questi blocchi costruiamo delle formule che ci garantiscano che l'esecuzione del blocco preservi gli invarianti delle classi e sia consistente con le pre e post condizioni di ciascun metodo

Andrea Baruzzo; 26/09/2006

The method at work – decomposition



35

The method at work – validation (equations checked)

$$Inv_{Company} \wedge Andrea.age \geq 40 \wedge salary \equiv 800 \implies True$$

$$Andrea.age \geq 40 \wedge salary \equiv 800 \wedge Result \equiv Andrea.age \implies Andrea.age \geq 18$$

$$Inv_{Company} \wedge Result \equiv Andrea.age \implies Inv_{Company}$$

$$Inv_{Company} \wedge Andrea.age \geq 30 \wedge Andrea.age \geq 40 \wedge salary \equiv 800 \implies True$$

$$Inv_{Company} \wedge Result \equiv 1500 \implies Inv_{Company}$$

$$Andrea.age \geq 40 \wedge Andrea.salary \equiv 1500 \implies Andrea.age \geq 18$$

$$Company.isDefined \wedge Andrea.isDefined \wedge Inv_{Company} \implies \\ Andrea.isDefined \wedge Company.employee \rightarrow exclude(Andrea) \quad (i)$$

$$Inv_{Company} \wedge Company.employee \rightarrow includes(Andrea) \implies Inv_{Company}$$

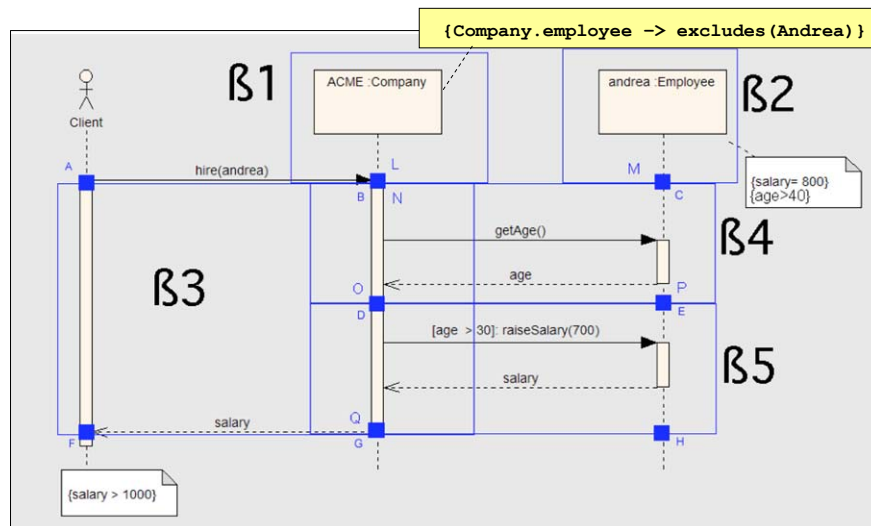
36

Diapositiva 36

Ab4 If we add in the diagram an initial constraint specifying that `Company.employee` \rightarrow `excludes(Andrea)` then we can prove the new (i) and then the diagram becomes consistent.

Andrea Baruzzo; 26/09/2006

... now the diagram becomes consistent!



37

Project for Thesis

- Tools for computer-aided validation of UML models**
 - Visual debuggers, simulators, model animations...
- Tools for computer-aided verification of UML models**
 - Proof engines, formal verification methods (static or dynamic), test case generators, model testing tools
- Metrics tools for UML models**
- UML profiles for quality**
- Formal specifications of Design Patterns**
- Tools for automate the layout of UML diagrams**
 - (i.e. according to Dependency Goes Up rule)
- Tools for model transformations**

38



Bibliography

- [Mens&VanGorp, 2005]
Tom Mens, Pieter Van Gorp – “*A Taxonomy of Model Transformation and its Application to Graph Transformation*”, Université de Mons-Hainaut, Mons, Belgium, 2005
- [Meyer92]
Bertrand Meyer, “*Applying Design by Contract*”, ACM Computer, Volume 25, Issue 10, 1992
- [MoDeV2a06]
Andrea Baruzzo and Marco Comini, “*Static Verification of UML Model Consistency*”, MoDeV2a Workshop, Satellite Event of the MoDELS 2006 ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genoa, Oct. 1-6, 2006
- [POSA1, 1996]
Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal – “*Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*”, Wiley&Sons, 1996
- [Seidewitz, 2003]
Ed Seidewitz – “*What Models Means*”, IEEE Computer, 2003