An integrated Multi-View Model Evolution Framework

Volker Stolz

United Nations
University

UNU-IIST
International Institute for
Software Technology

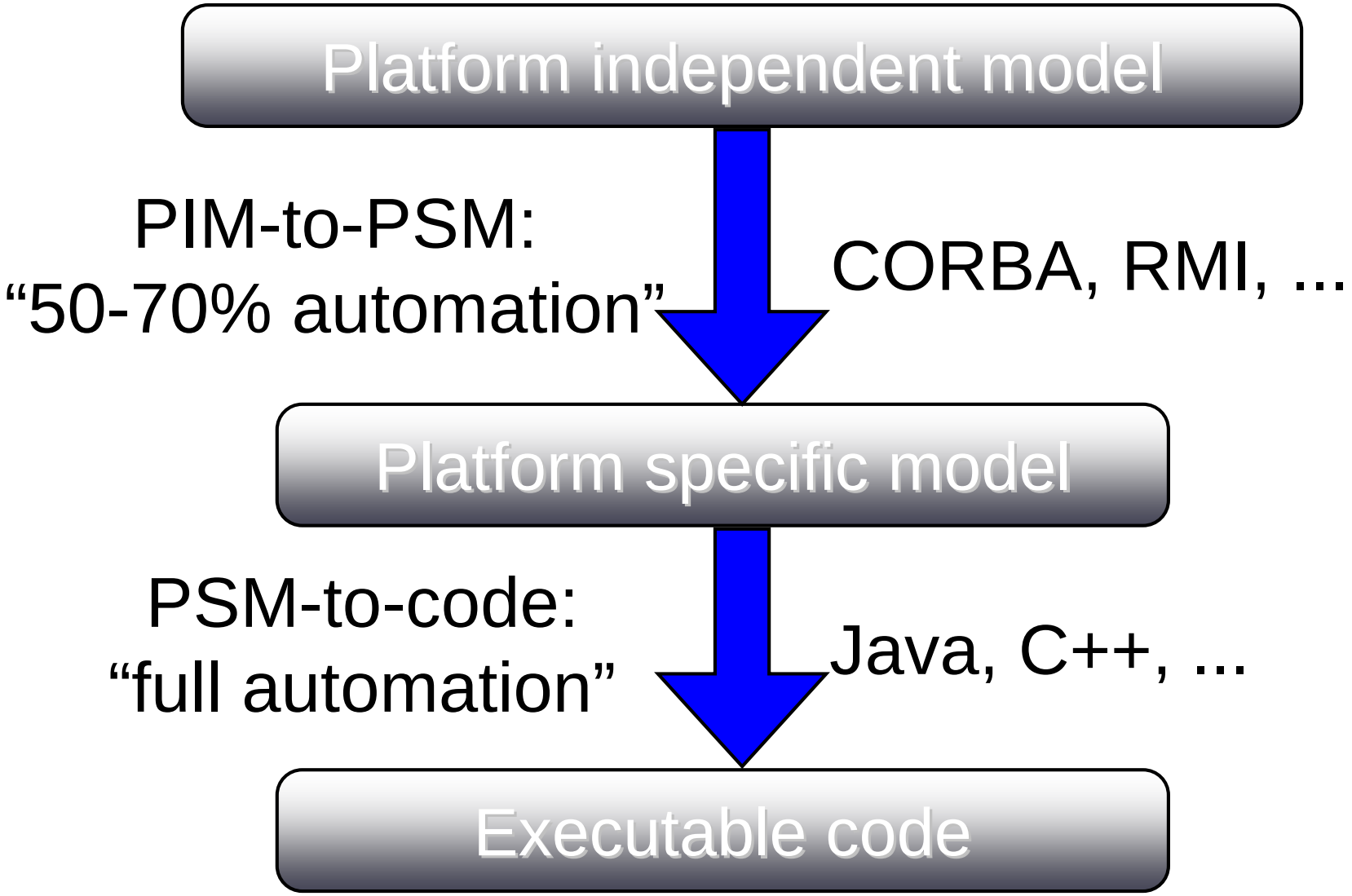
Problem Description

- rCOS Modeler for use case-driven, component based, development
- Uses UML (+ rCOS profile)
- Development process through refinement/transformation
- Verification (model checking), code-gen (Java), test-case generation (MBT)

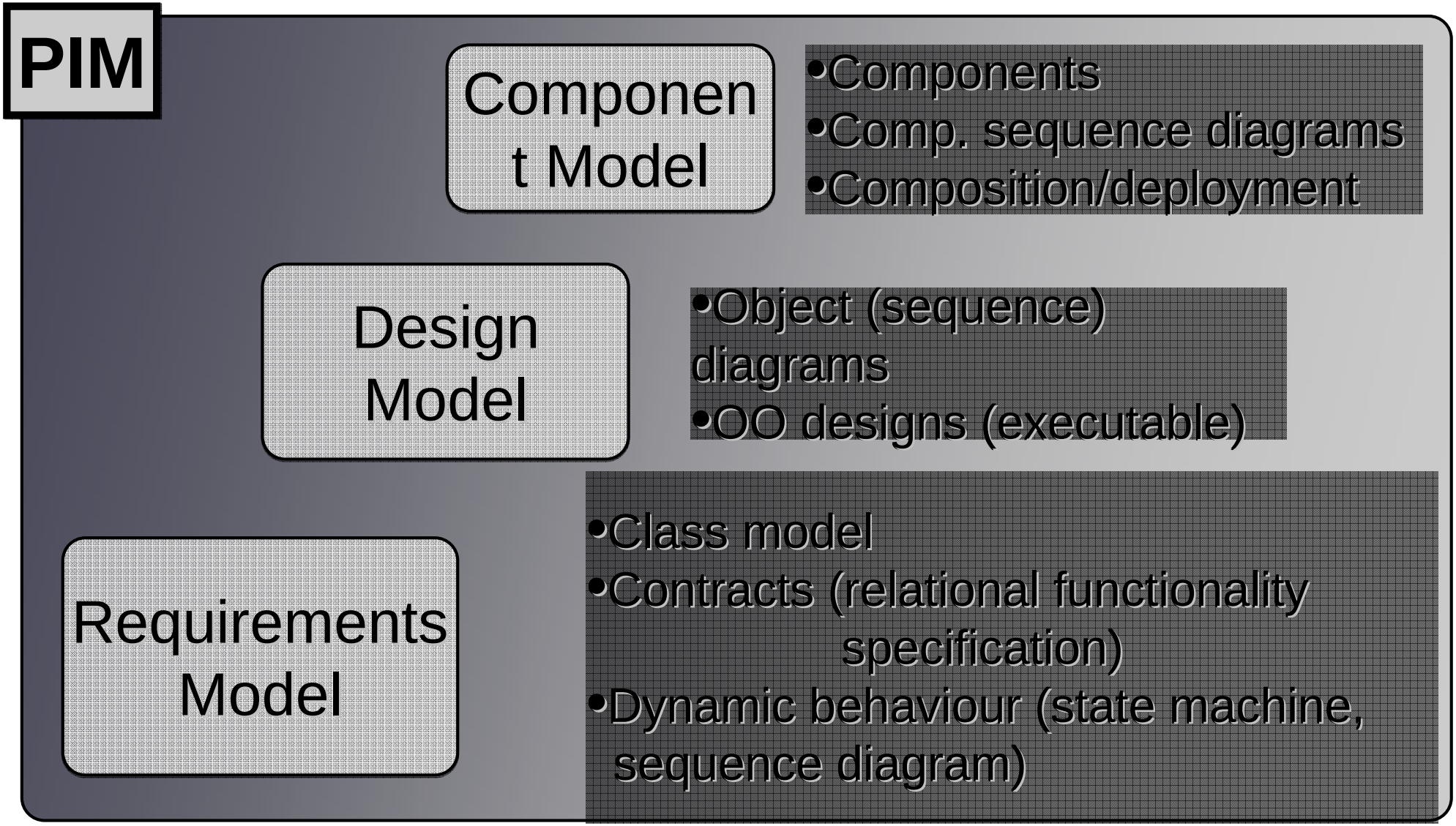
Where are the models?

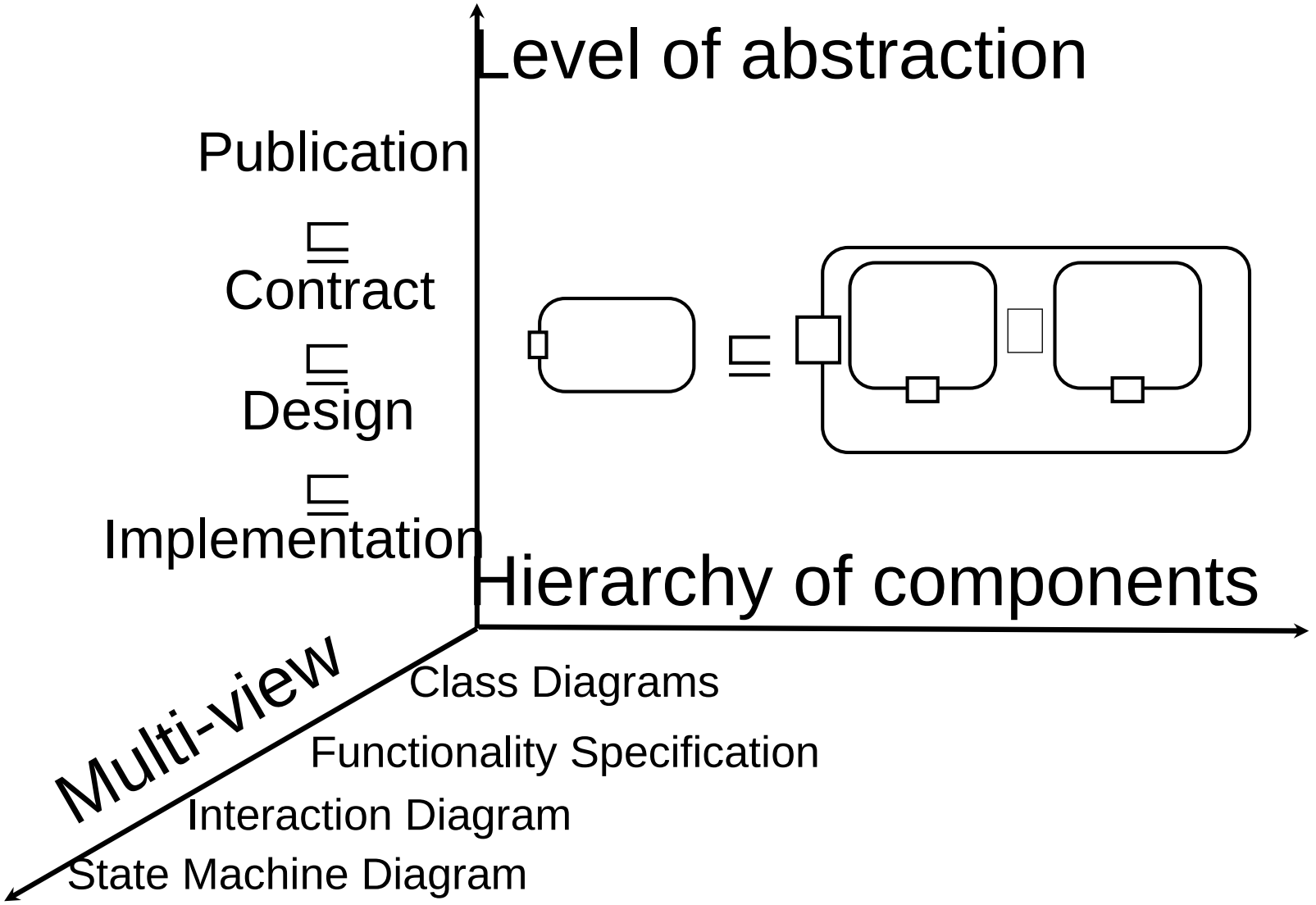
- Possible views:
  - There is no model—at least not until you're finished.
  - There a (too) many models—many models along the way to the result.
  - There is only one model—but not the one you expected!

Modeling Levels: OMG



PIM: sub-divisions




Level of abstraction
Publication
⊑
Contract
⊑
Design
⊑
Implementation
⊑
Hierarchy of components
Multi-view
Class Diagrams
Functionality Specification
Interaction Diagram
State Machine Diagram

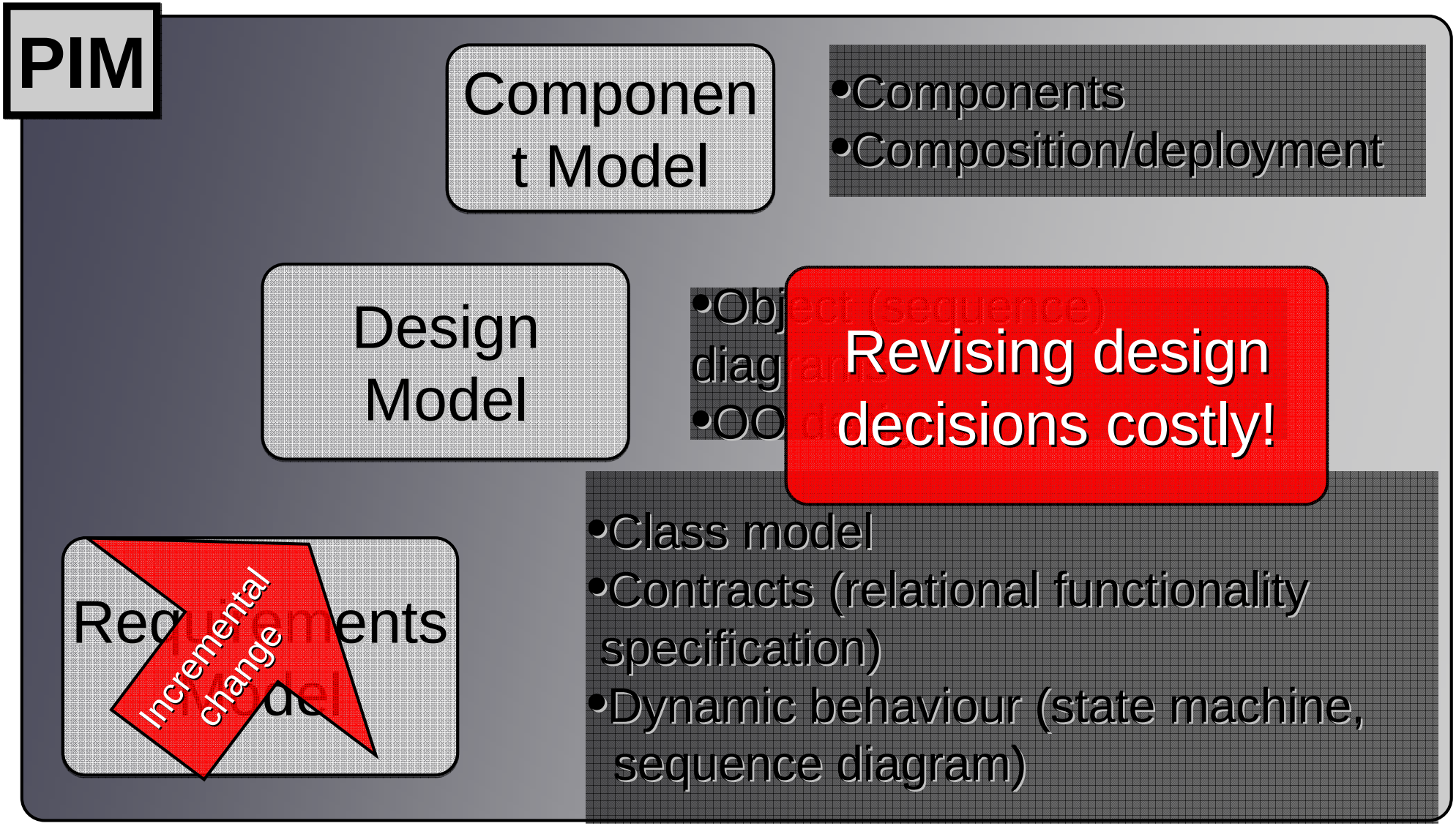
Creating Models
(UML style)

- Atomic actions:
  - add child element
  - update attribute
  - delete child element
  - (plus "pocket" changes like primitive types, multiplicities etc.)

Traditional model editing,
lather, rinse, repeat.

What about design decisions?
Applied patterns:
refactoring/refinement)?
(Concurrent editing)

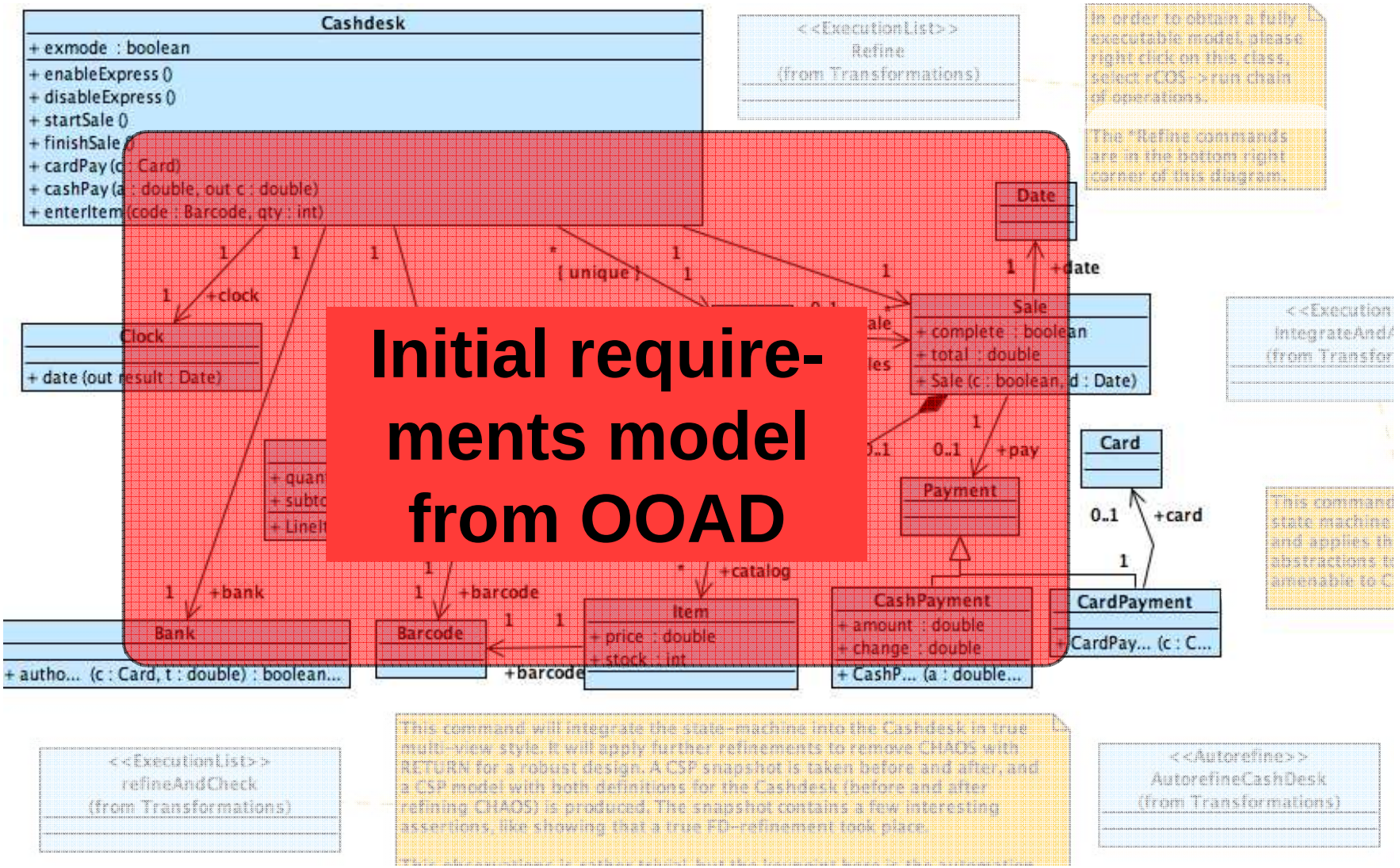
PIM: sub-divisions



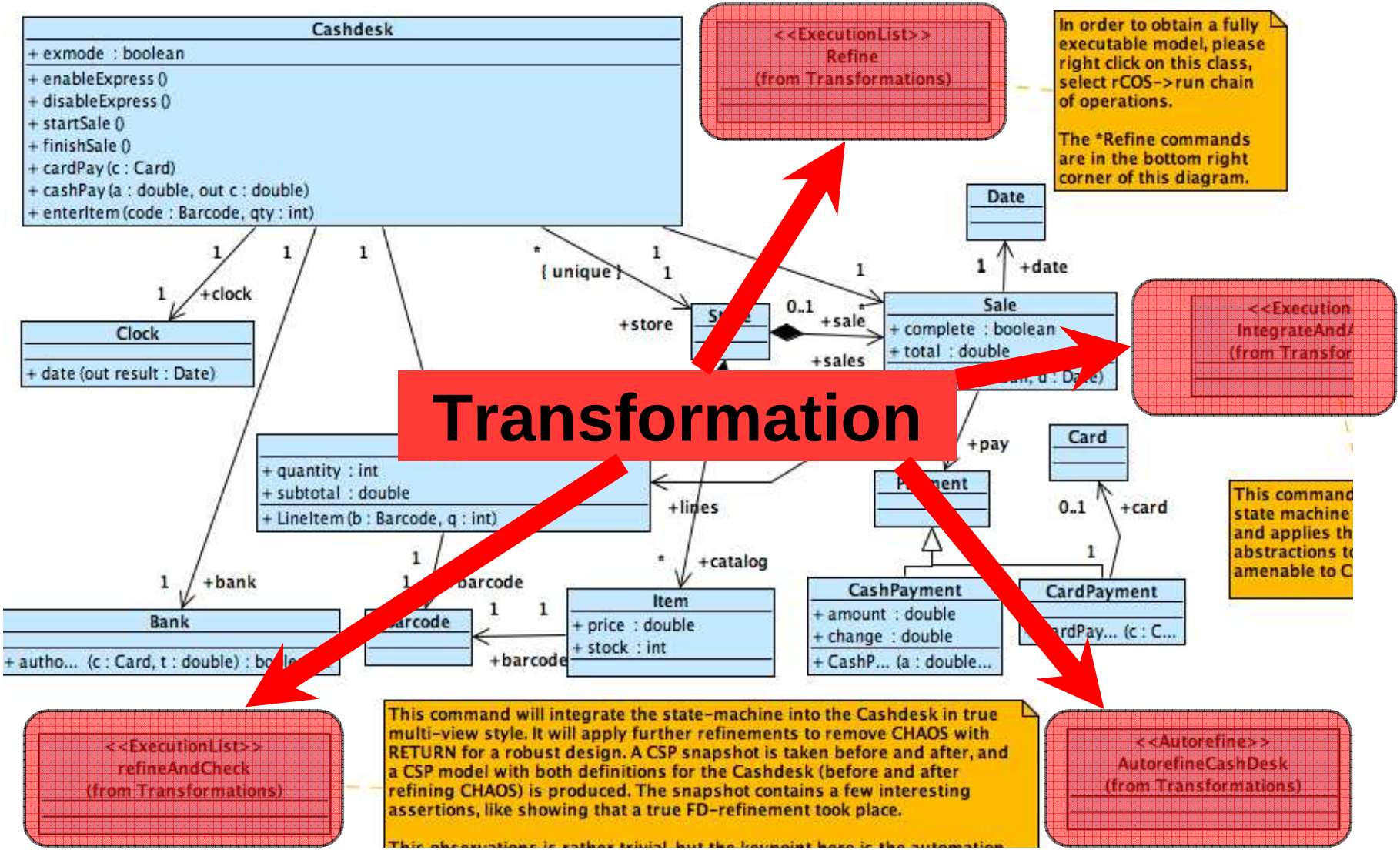
Transformations in rCOS

- Class model:
  - “Usual suspects”, implemented:
  - Expert Pattern, simple Auto-refine
  - Manual refinement steps
- Abstraction (on the method-level)
- Integration (merge state machine into “guarded designs”)
- Component/Use case: splitting into sub-components/use cases (in progress)

Current setup (1)



Current setup (2)



Transformations...

- ...are based on meta-model with classes and associations/attributes.
- ... can be primitive actions (add, delete, update).
- ...have formal parameters.
- ...can be chained: *output* of one transformation is input for another one.
- ...have prerequisites.
- ...are refinements (proof!).

Updating a model through a transformation

- Workflow:
  - User selects transformation and arguments. Example:
    Expert pattern applied on sub-expression with [illegible]ation.
  - Chec[illegible]
    Find [illegible]me in target class.
  - Compute and execute primitive actions.

Model updated destructively!

Alternative

- GUI only represents *current state* of model.
- Allow user to browse model before/after transformation.
- Chaining and re-playing comes naturally.
- Develop independent transformations independently.

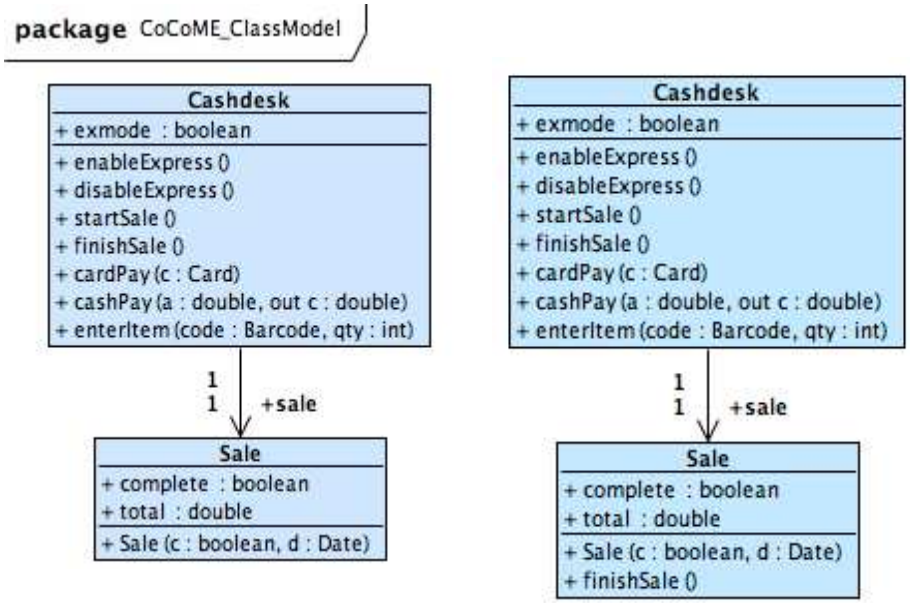
Transformations become the model!
(+ initial model)

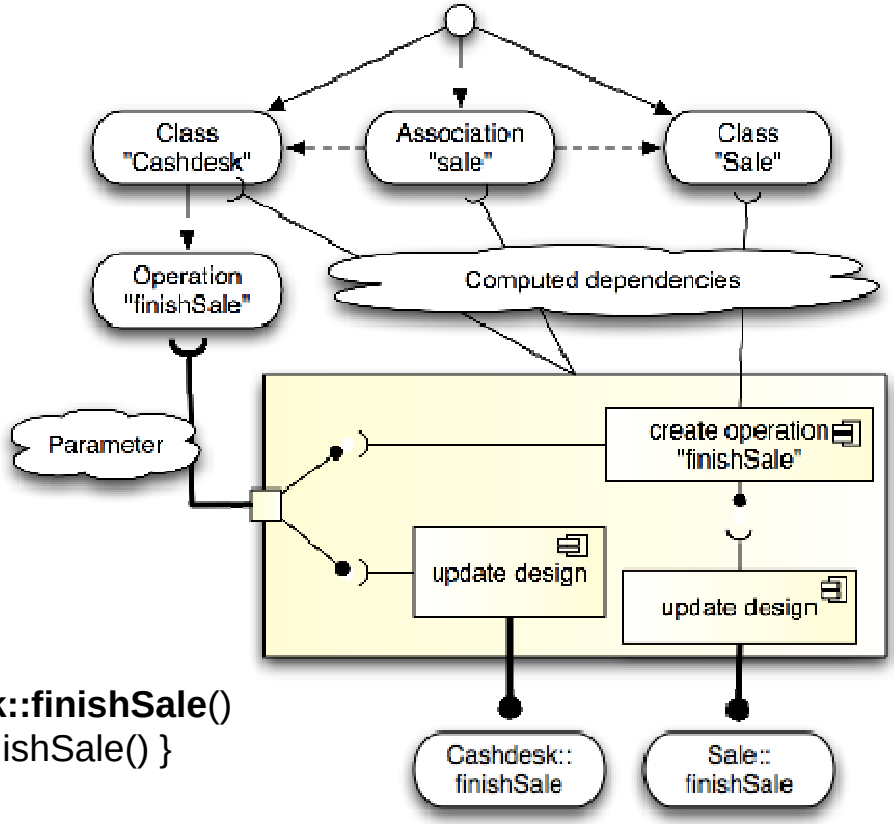
- Primitive actions are singleton transformations, corresponding to traditional editing steps!
- Final model described by (final) “chain” of transformations.

Editing chains of transformations

- Transformation as batch job.
- Achieving the desired output model means changing the transformations.
- Inserting/removing a step influences subsequent step because of dependencies.
- Especially so when there are proof obligations/proofs.

Example: Expert Pattern





Cashdesk::finishSale() {
[sale != null ⊢ sale.complete' = true];
var double sum;
sum := 0;
for (LineItem l : sale.lines) {
    [⊢ sum' = sum + l.subtotal]
};
[⊢sale.total' = sum];
end sum
}

Cashdesk::finishSale()
  { sale.finishSale() }

Sale::finishSale() {
[⊢ complete' = true];
var double sum;
sum := 0;
for (LineItem l : lines) {
    [⊢ sum' = sum + l.subtotal]
};
[⊢ total' = sum];
end sum
}

Things to consider

- Change to the (initial) spec of `Cashdesk::finishSale()`
  → re-apply transformation.
- Change in the hierarchy of `Sale` requiring to re-check prerequisites.
- There was no `Sale::finishSale()` before—now other transformations/editing actions may depend on it.

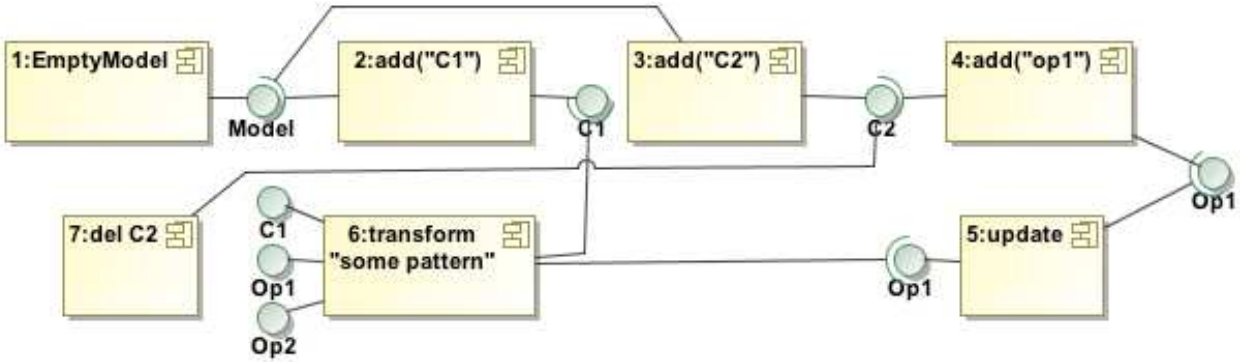
Some observations

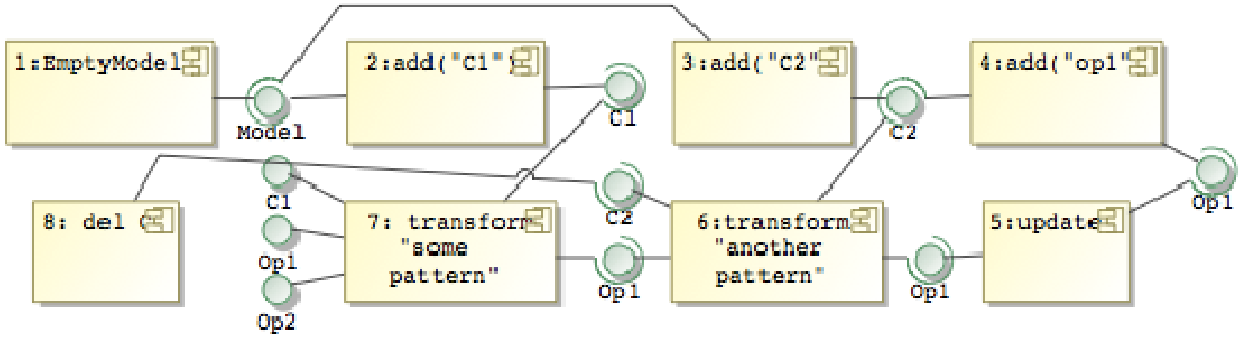
- Refinement-based modeling means developing a program that will transform your model (what about strategies?).
- Model becomes data structure, transformation becomes the model (Composition of transformations).
- Inherits many (interesting) problems and solutions from programming/program analysis;
  dependency tracking may have huge overhead?
- “Semantic overhead” from refinement setting.
- Presentation issues (GUI just a view; trans-formation should be in-memory instead of on-disk).
- Multi-view: you may not be seeing whole model/entire impact.

A quick word on proof obligations

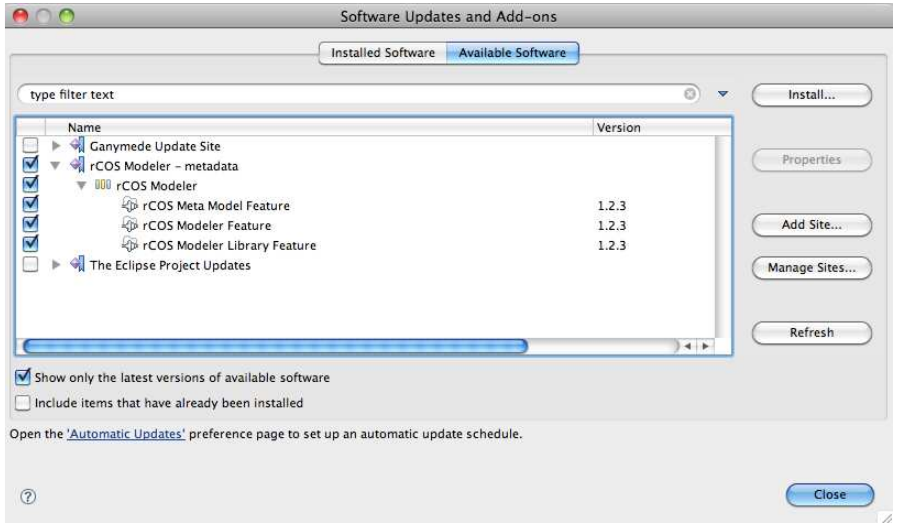
- Source of proof obligations:
  - Prerequisites of transformations (semantic!).
  - NOT for refinements themselves...
  - ...except for manual refinement steps (which syntactically substitute one design for another).
  - May be updated when re-running transformations.
- Proofs:
  - Stored together with transformation.
  - Re-checked when transformation is re-run.
  - Needs to be updated/re-done when it fails.

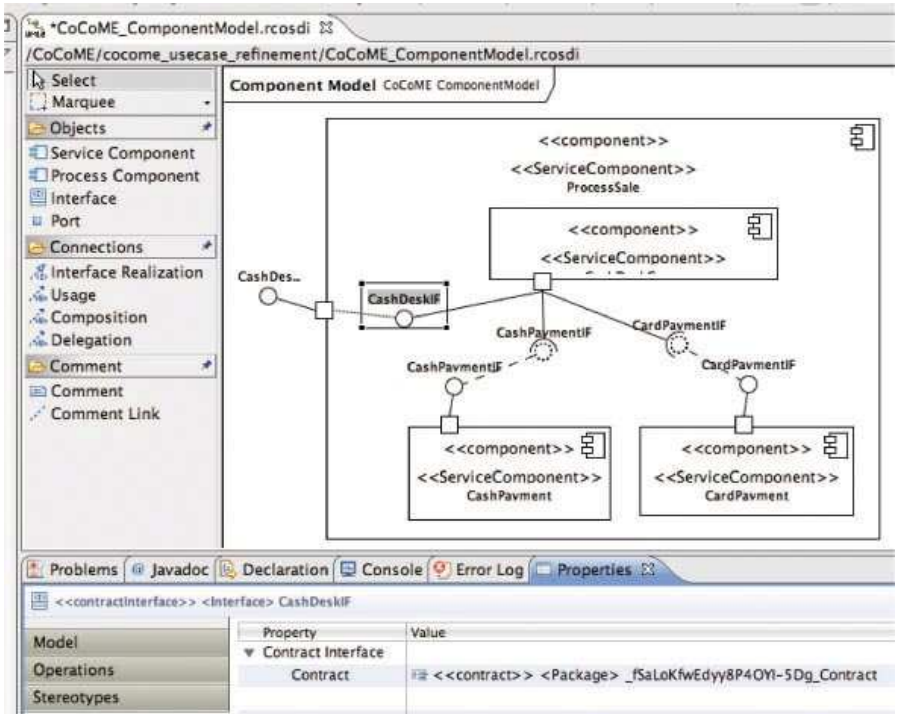
Appealing visual notation?





Trying out rCOS





- Eclipse Ganymede (3.4.2)
- rCOS Eclipse update site:
- http://rcos.iist.unu.edu/eclipse/
- Download the example models

TOPCASED