From Modelling to Meta-Modelling To Concern-Orientation

Jörg Kienzle

Why Modelling?

"Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose."

Jeff Rothenberg
The Nature of Modeling
John Wiley & Sons, August 1989

Why Modelling?

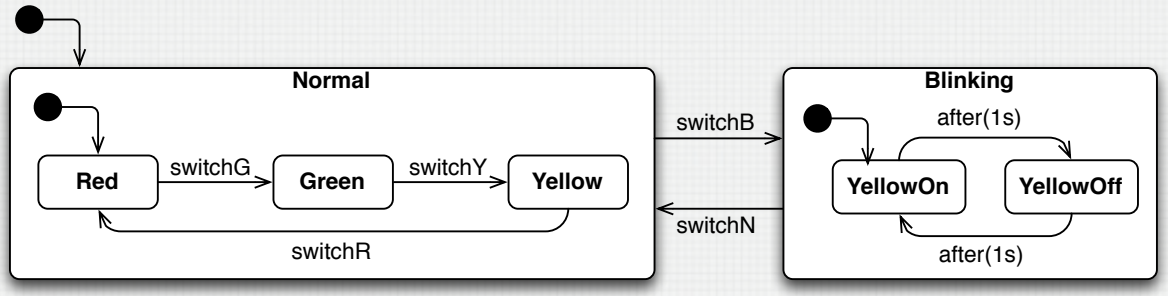
"A model represents reality for a given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality."

Jeff Rothenberg
The Nature of Modeling
John Wiley & Sons, August 1989

Modelling and Software Development

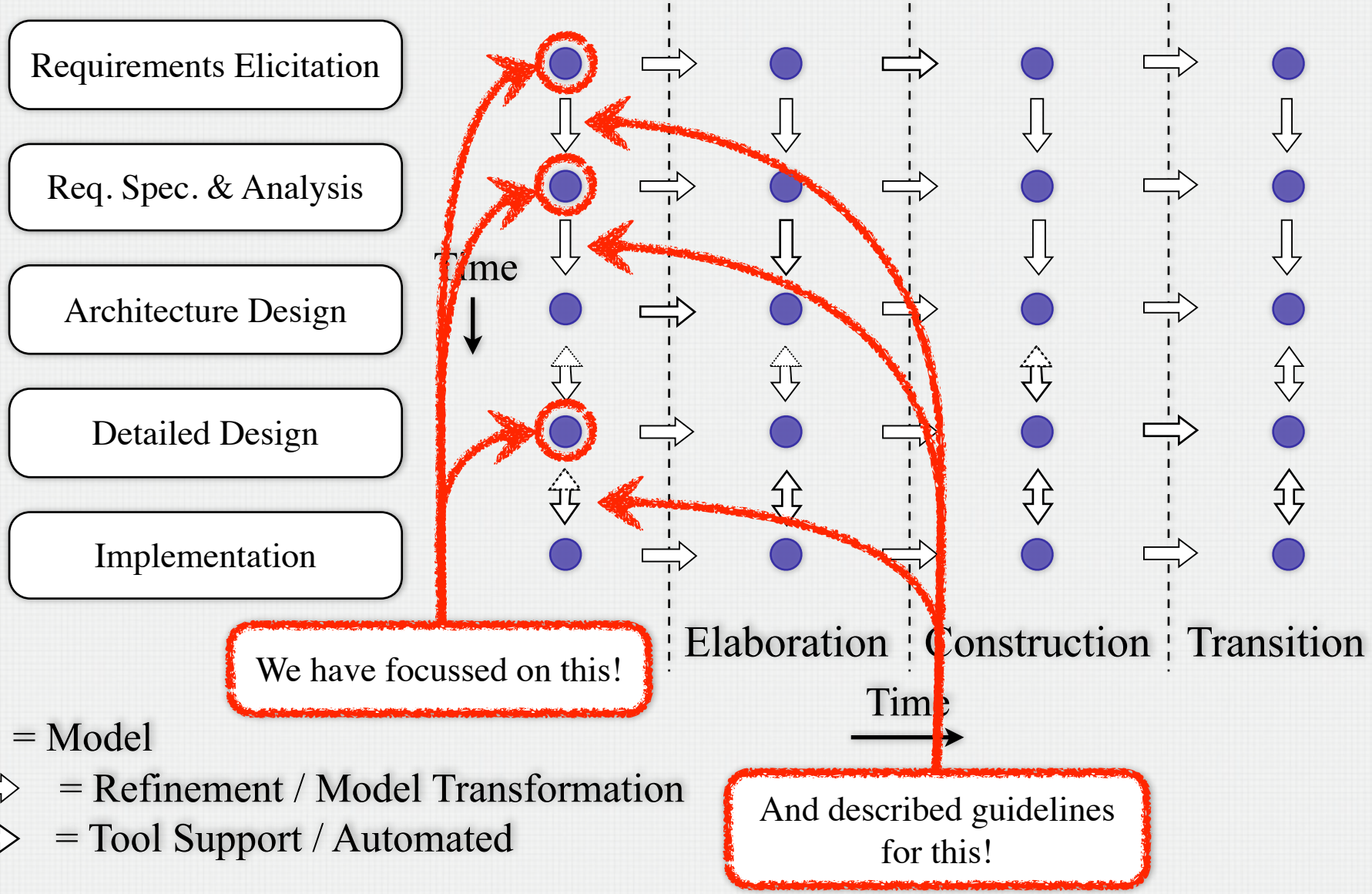
- A Model is a simplified representation of an aspect of the world for a specific purpose
- We use models to better understand a system
  - For a observer A, M is a model of an object O, if M helps A to answer questions about O. (Minsky)
- A model helps to understand, communicate and build
- Modelling and engineering: model something not yet existing!

M1 - Modelling Space



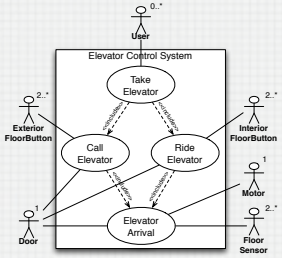
M0 - The World

Model-Driven Development

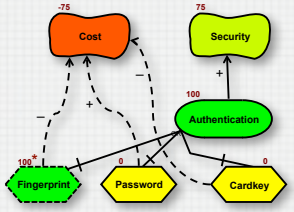


Model-Driven Development

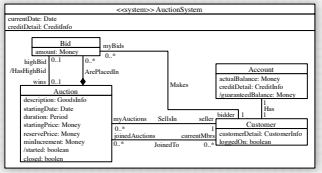
- Requirements Elicitation
- Req. Spec. & Analysis
- Architecture Design
- Detailed Design
- Implementation



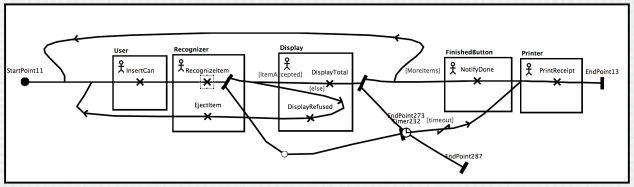
Use Case Model



Goal Model



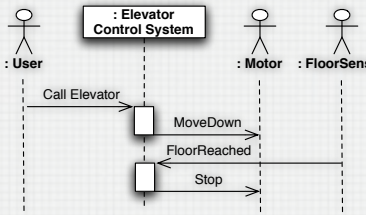
Concept Model



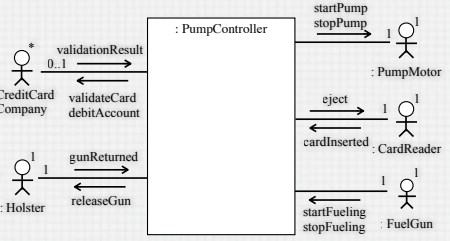
Use Case Maps

```
Operation: EmploymentAgency::jobFilled
  (worker: Person, comp: Company, amount: Money);
Description: Creates a job for a given person and company,
where company must have a budget smaller than or equal to
10 million;
Scope: Person; Company; Job;
New: researchJob: Job;
Pre: self.company→select(budget > 1.0E7)→excludes(comp);
Post:
   researchJob.oclIsNew() &
   researchJob.salary = amount &
   researchJob.employee = worker &
   researchJob.employer = comp;
```

Operation Model



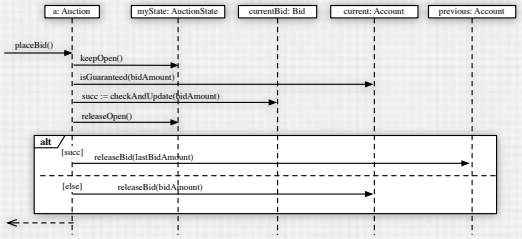
Protocol Model



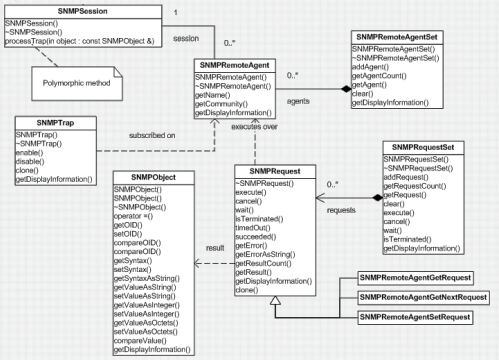
Environment Model

```
public class Asteroid
  extends Model {

  //position
  float xPos;
  float yPos;

  //dynamics
  float speed;

  public Asteroid() {
    xPos = ConstantWORLD_MAX_X;
    yPos = 0;
  }

  public void moveAsteroid() {
    xPos = xPos - speed;
  }

  public boolean outOfBounds() {
    return xPos < 0;
  }

}
```

Java Code



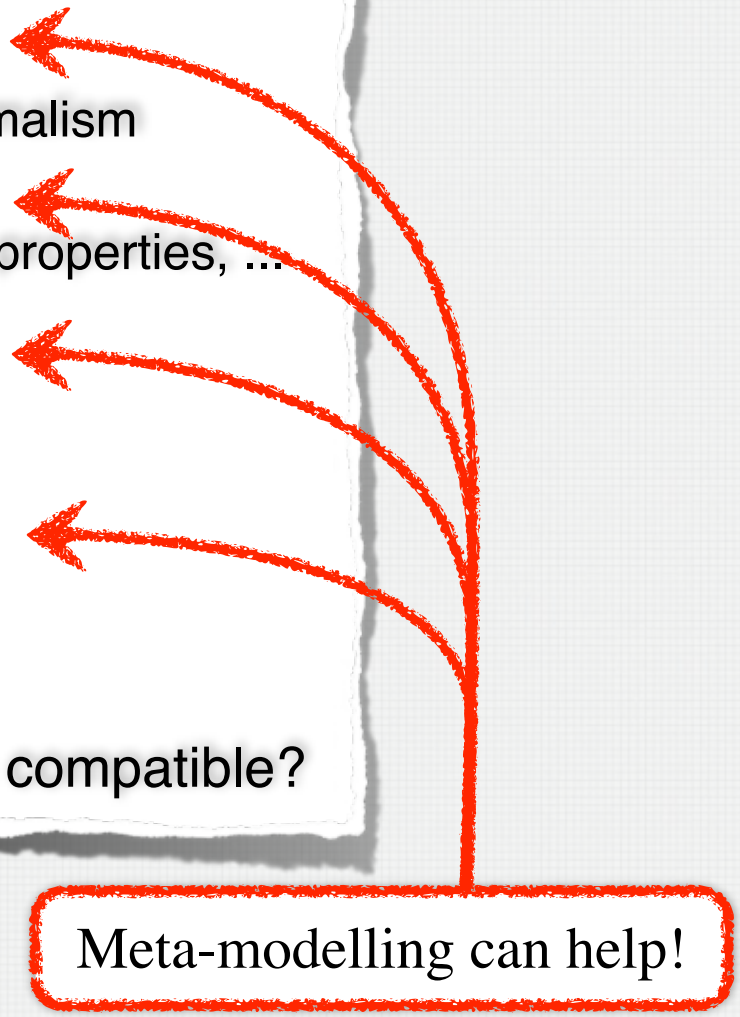
Interaction Model



Design Class Model

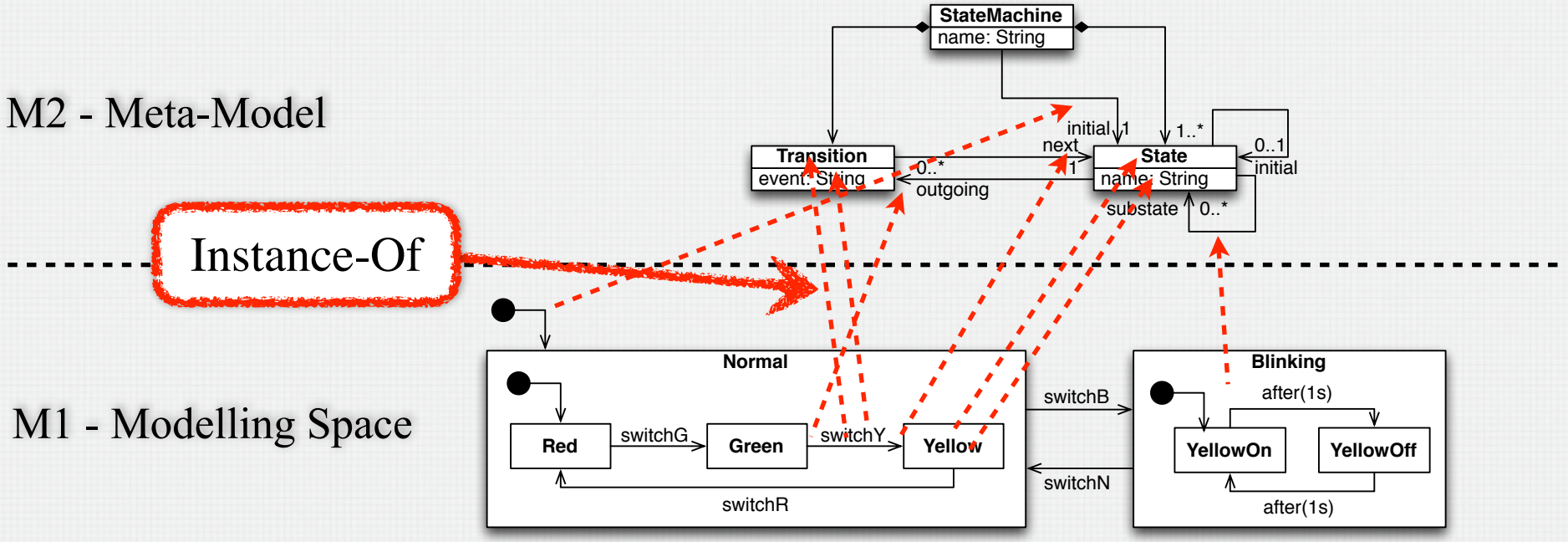
Tools Essential for MDE

- Tools for
  - Creating models
    - Using the most appropriate modelling formalism
  - Analyzing models
    - To find problems / inconsistencies, prove properties, ...
  - Transforming models
    - To other models, to code
- In this class
  - JUCMNav
  - UML tools
  - TouchRAM
  - Wouldn't it be nice if the tools had been compatible?



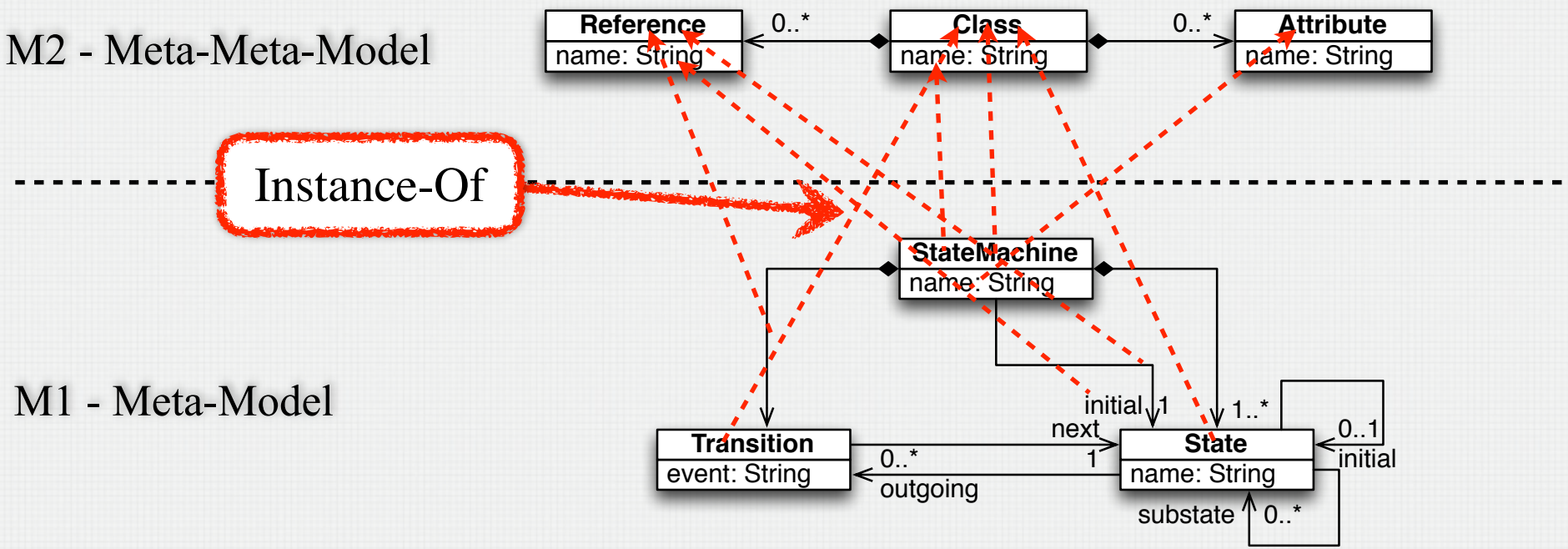
What is a Meta-Model?

- Simple definition
  - A model of a model
- A model M conforms to a meta-model MM iff
  - Each model element in M is an instance of a model element in MM
  - All constraints (e.g. OCL) defined on MM are adhered to in M



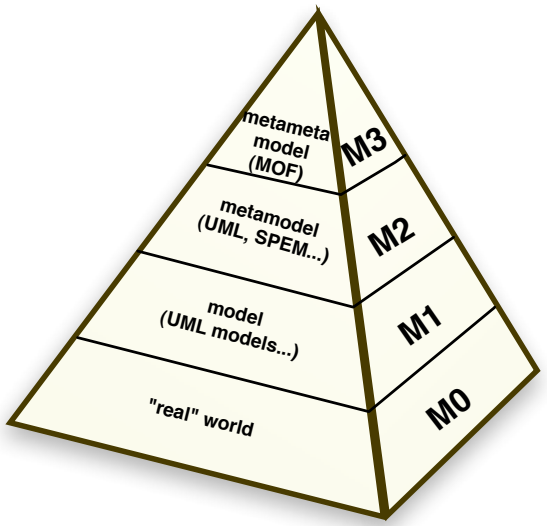
Why Stop at Meta?

- Can we model meta-models?
  - Yes we can!
- Why would we do so?
  - To have a unified framework for defining meta-models

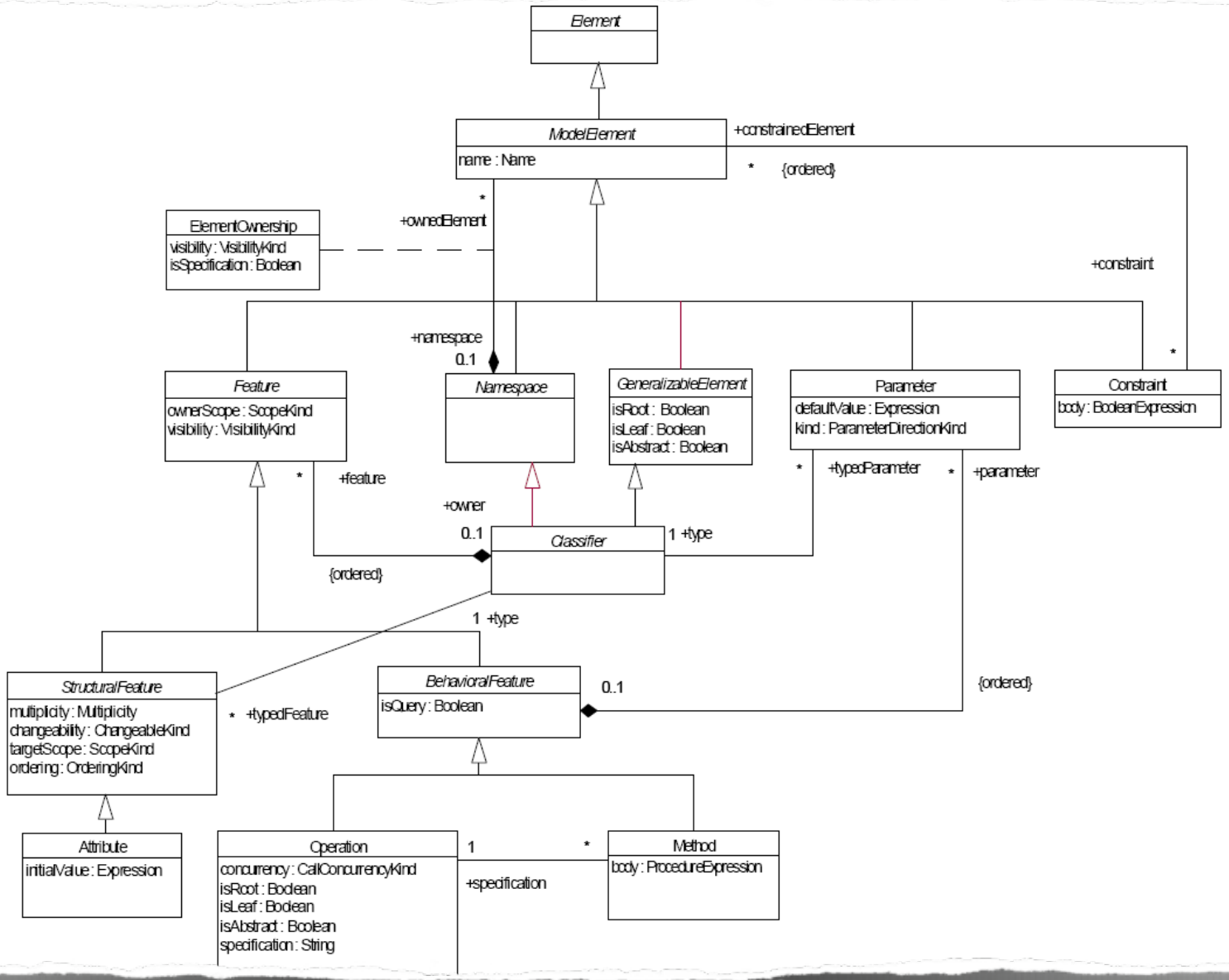


The OMG Modelling Pyramide

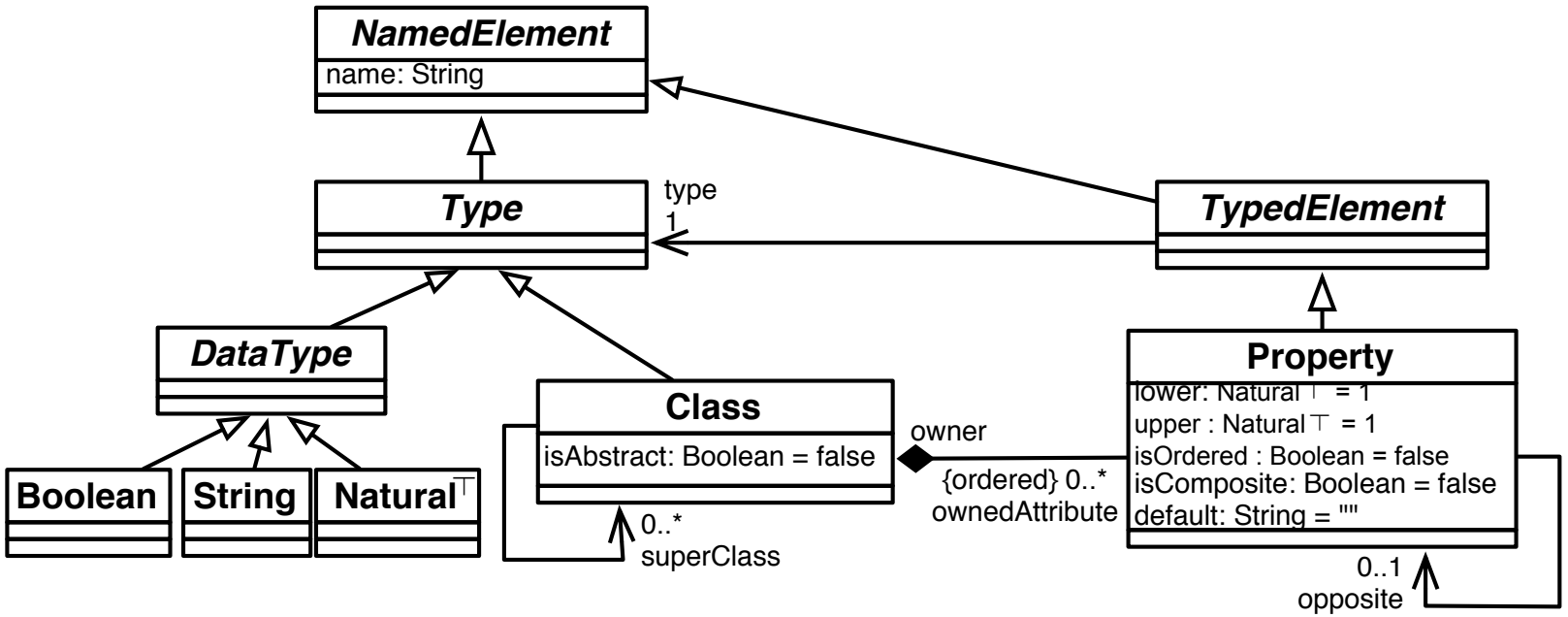
- M3: Meta-meta model
  - Reflexive, i.e. self-describing / bootstrapping
  - Used to define meta-models / modelling languages
  - Examples: MOF (Meta-Object Facility), Ecore (used in Eclipse)
- M2: Meta-model
  - Used to define modelling languages for a specific domain
  - Examples: UML2 (for Software)
- M1: Model
  - Used to define models, i.e. abstractions of "the real thing" for a given purpose
  - Example: a model of a traffic light
- M0: The real world
  - Example: a traffic light



UML2 Meta-Model (parts of)



OMG EMOF (PARTS OF)



http://www.omg.org/spec/MOF/

Meta-Modelling Advantages

- If a modelling language MO has a meta-model MM, then
  - It provides an abstract syntax for MO
    - It is possible to check if a model M is a valid model, i.e. a valid instance of MM
    - It is possible to attach a concrete syntax to it
  - It is possible to define model transformations (i.e. algorithms that manipulate models) that can be applied to *any* MO model
    - Example: re-curring manipulations, refactorings, etc...
- If MO and MO' are two modelling languages with meta-models MM and MM', then
  - It is possible to define model transformations from MO to MO', i.e. algorithms that transform any MO model into a MO' model
  - This is useful, among other things, for
    - Providing semantics for MO
    - Connecting different levels of abstraction (for example from requirement models to design models, or from design models to code)
    - Converting between notations (for example to use existing model analysis tools)

Eclipse Modelling Framework

- Eclipse Modelling Framework is composed of
  - Ecore, the EMF Meta-meta model
    - Can be used to define meta-models (stored in .ecore files)
    - XMI serialization
  - EMF.edit
    - Given a meta-model (.ecore file), generate Java code for manipulating (create/edit) models that conform to the meta-model
    - Command-based editing (provides undo / redo)
  - EMF.Codegen
    - Code generation for building a model editor
    - Integrates with Eclipse GMF (Graphical Modelling Framework)
  - http://www.eclipse.org/modeling/emf/

- Kermeta
  - http://www.kermeta.org
  - OCL-like language to specify model transformations
  - Aspect-oriented

Meta-Modelling and RAM

- We have a meta-model for RAM
  - Every .ram model is an instance of RAM.ecore
- We have defined model transformations on .ram models
  - Structural view weaving
  - Message view weaving
  - State diagram weaving
- TouchRAM uses the meta-model for
  - Loading / saving .ram files
  - Visualizing RAM models
  - Editing RAM models
  - Weaving RAM models

Concern-Orientation

Reuse

The process of creating software systems from existing software artifacts rather than creating them from scratch.

Charles W. Krueger, 1992

Model-Driven Engineering

Models are created at the right level of abstraction using the most appropriate modeling formalism. Model transformations map models at one level to models at another level of abstraction

Separation of Concerns (SoC)

This is what I mean by focusing one's attention upon a certain aspect; it does not mean completely ignoring the other ones, but temporarily forgetting them to the extent that they are irrelevant for the current topic. Such a separation, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts that I know of.

Edsger Dijkstra, 1976

Concern-Oriented Reuse

- A concern
  - Reaches across software development phases
  - Has a root phase
  - Is represented at the root and at each following phase by model(s) which
    - Use the most appropriate formalism to express the properties and composition rules relevant at the current level of abstraction
    - Have well-defined interfaces for all models
    - Are modelled as general as possible including all relevant variations
    - Give guidance on how to choose among existing variations
  - Defines model transformations that link model(s) from one phase to model(s) at the next phase
    - To avoid duplication of effort
    - To minimize accidental model complexity
    - To preserve properties

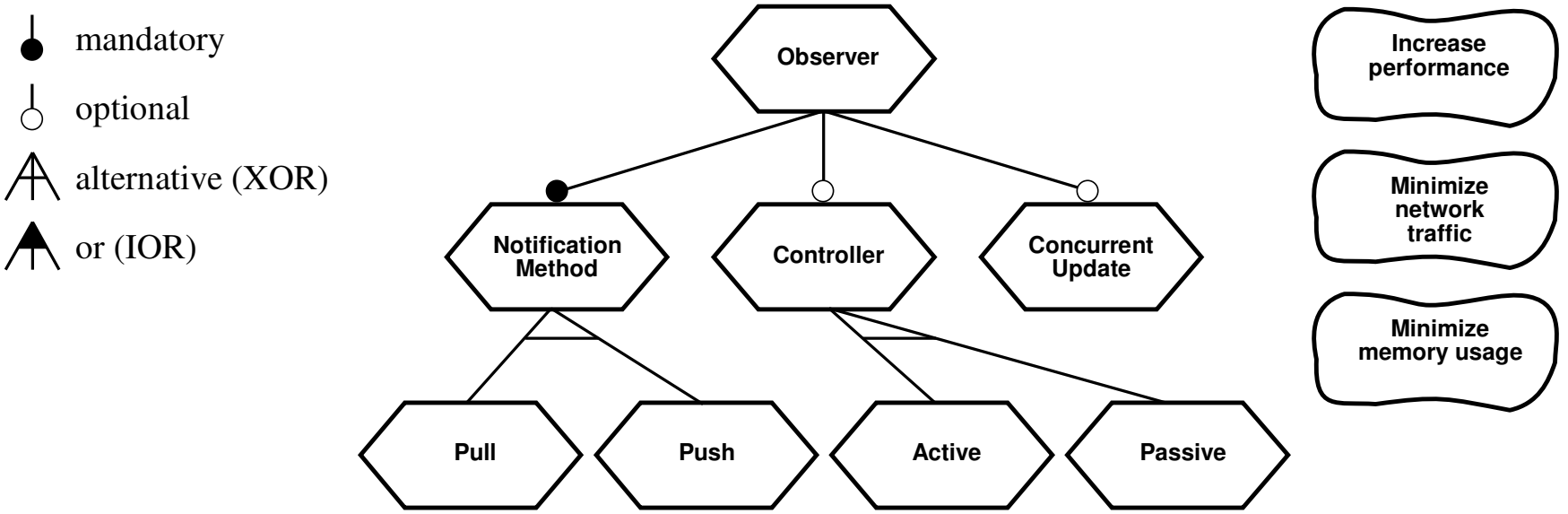
Concern-Oriented Design Reuse

1. Use the variation interface of the concern to select the most appropriate feature(s)
   - That provides the desired functionality
   - That maximizes positive impact on relevant non-functional application properties

   ➡ This generates the detailed generic design for the selected feature(s) of the concern
2. Use the customization interface of the generated design to adapt the generic design elements to the application-specific context

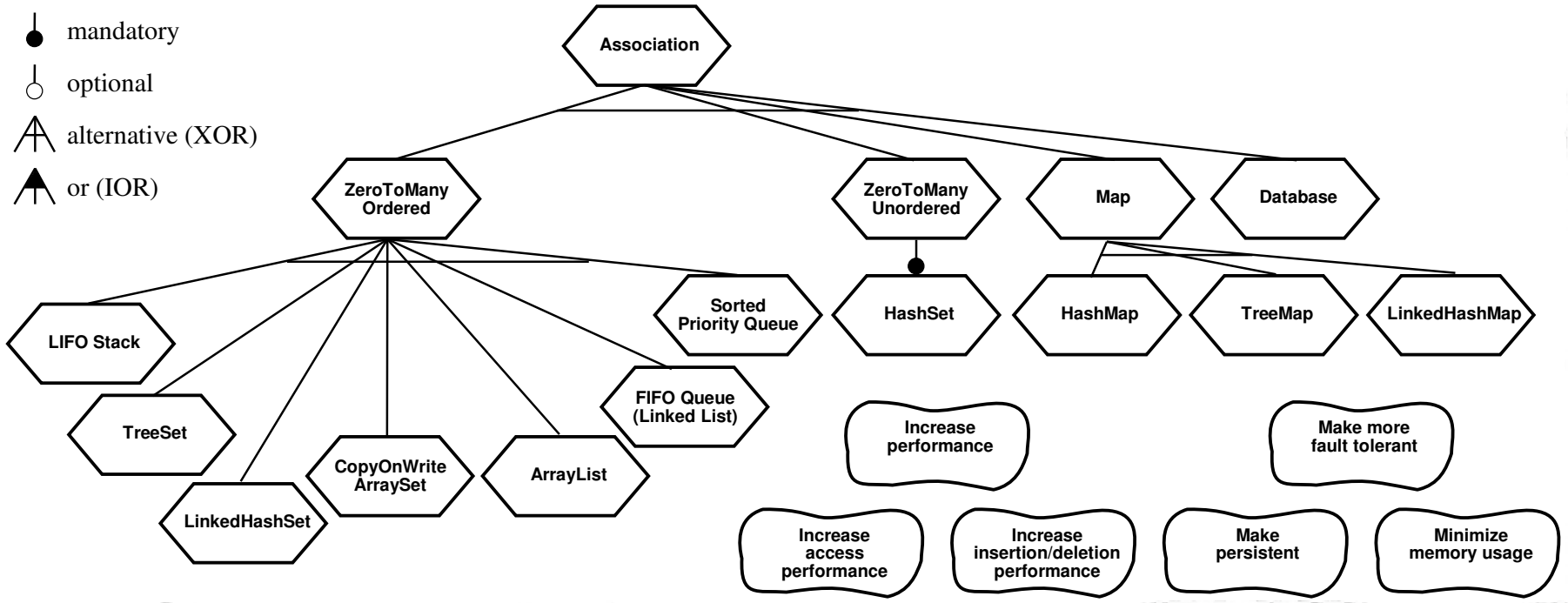
   ➡ This generates the application-specific design for the selected feature(s) of the concern
3. Use the selected concern feature within the application design according to the usage interface

Observer Concern



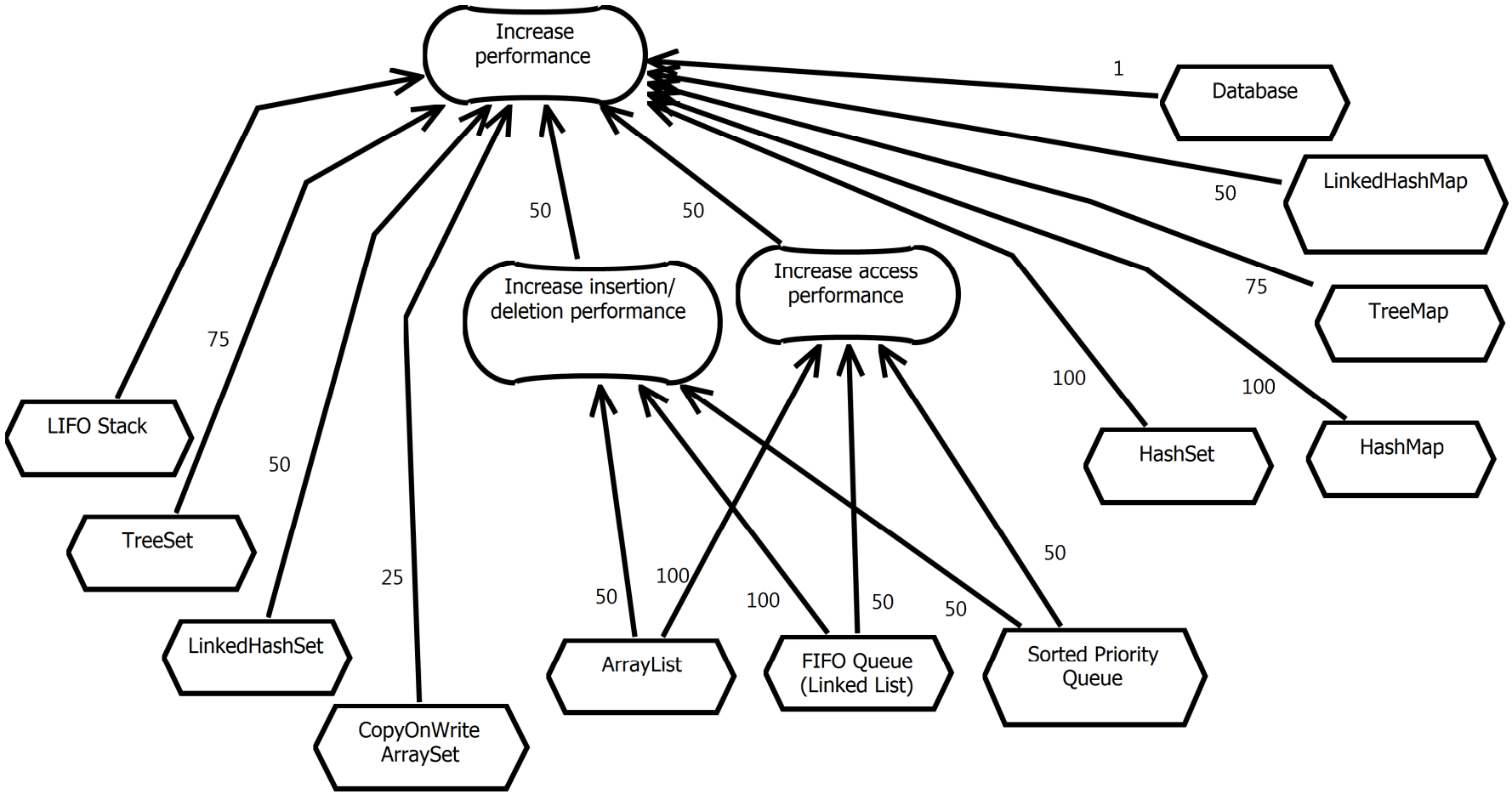
- Variation Model of a Concern
  - Exposes possible design choices
    - Optional, requires, excludes relationships also supported
  - Exposes impact on NFRs / qualities

Association Concern

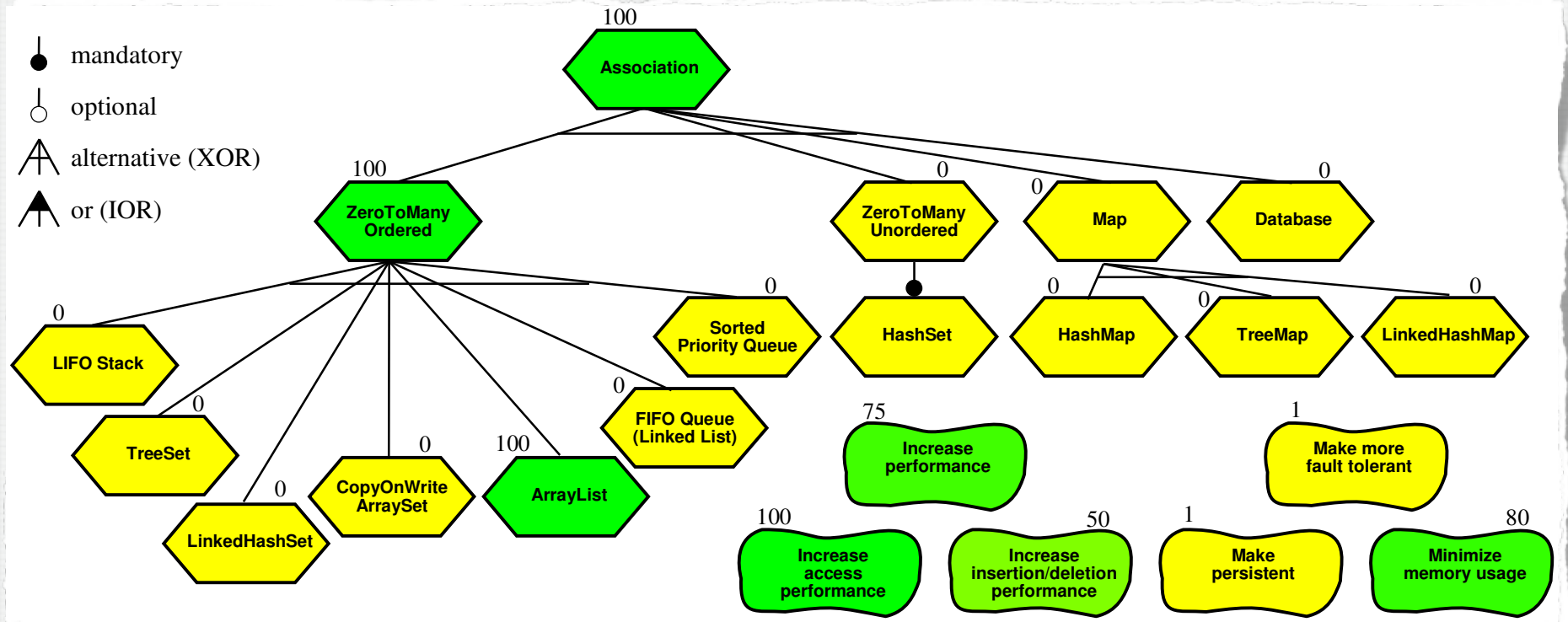


- Variation Model of how associations can be realized
  - Ordered, unordered, volatile, persistent, etc...

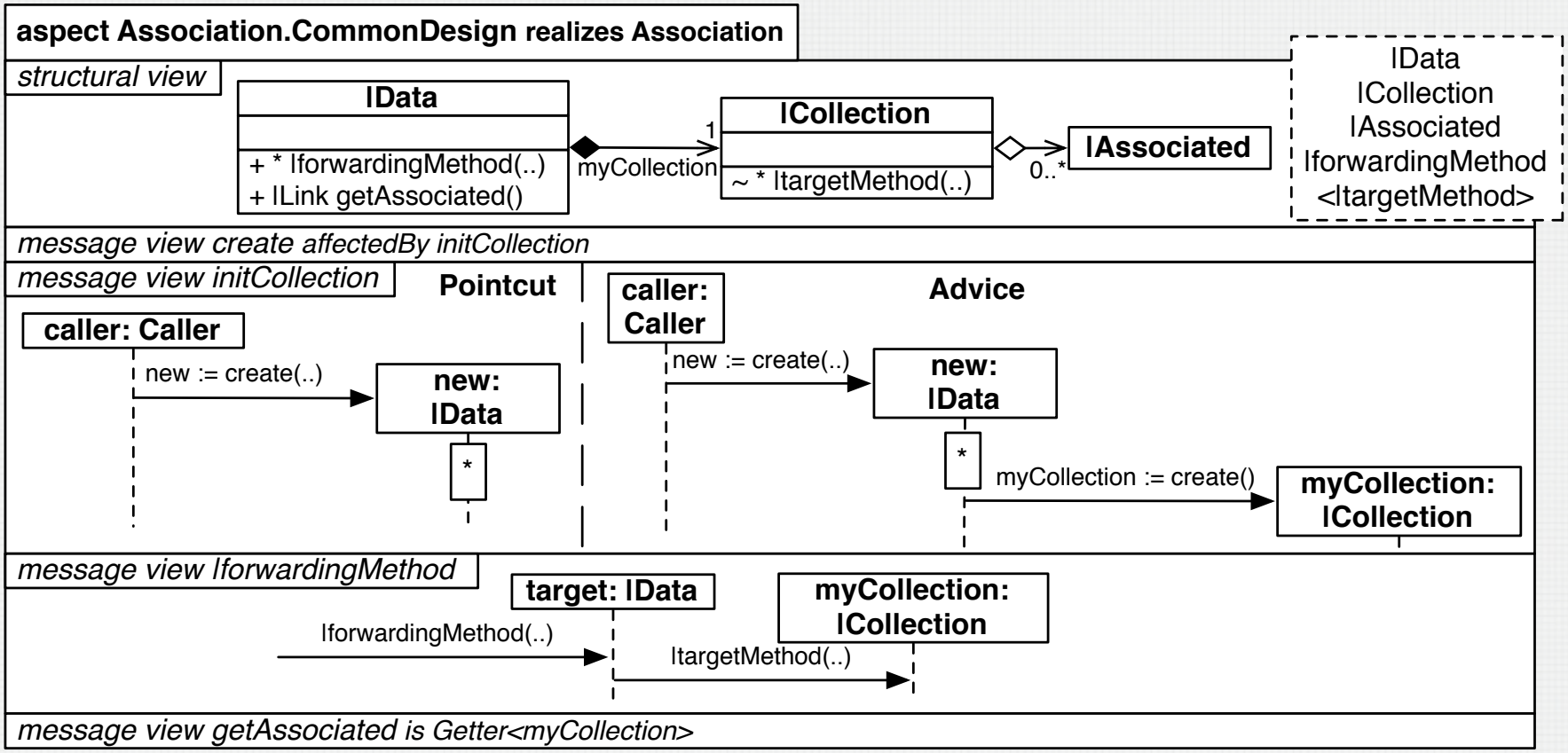
Impact Model - Performance



Impact Evaluation for ArrayList Feature



Feature: Association

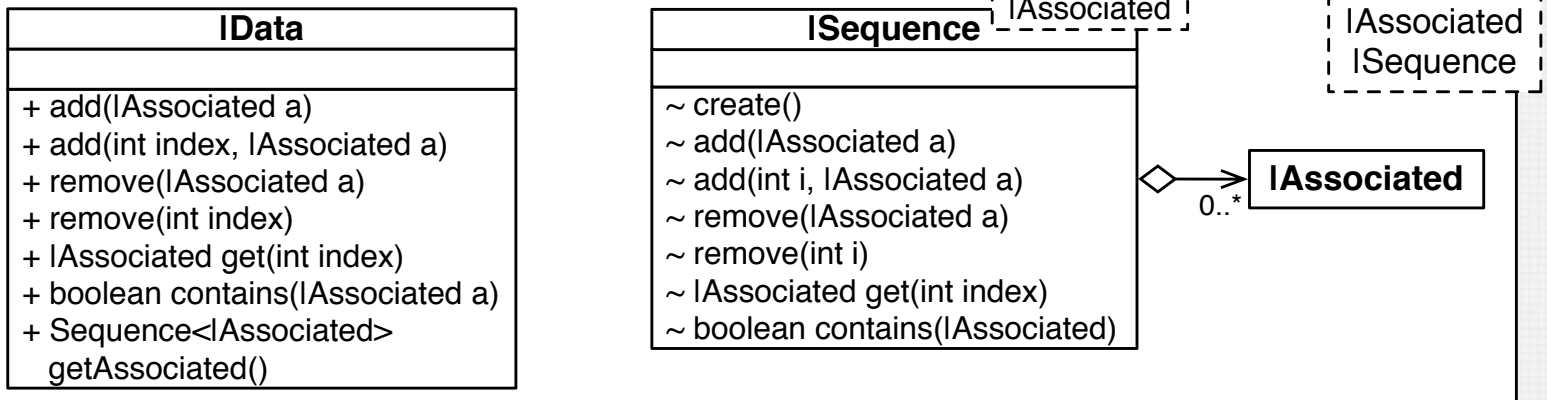


- Features are realized by (zero, one or several) RAM models

Feature: Ordered

aspect Association.Ordered extends CommonDesign realizes Ordered

structural view

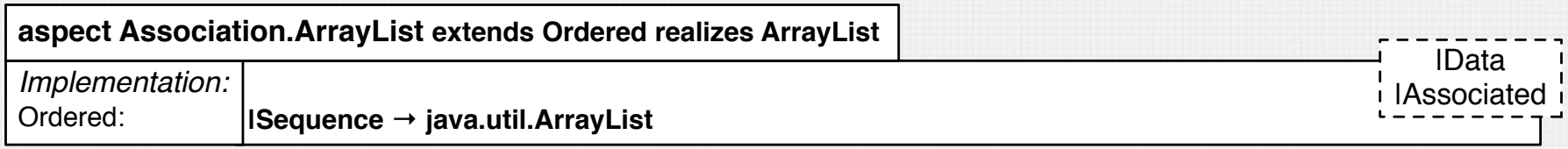


Instantiations:
CommonDesign:

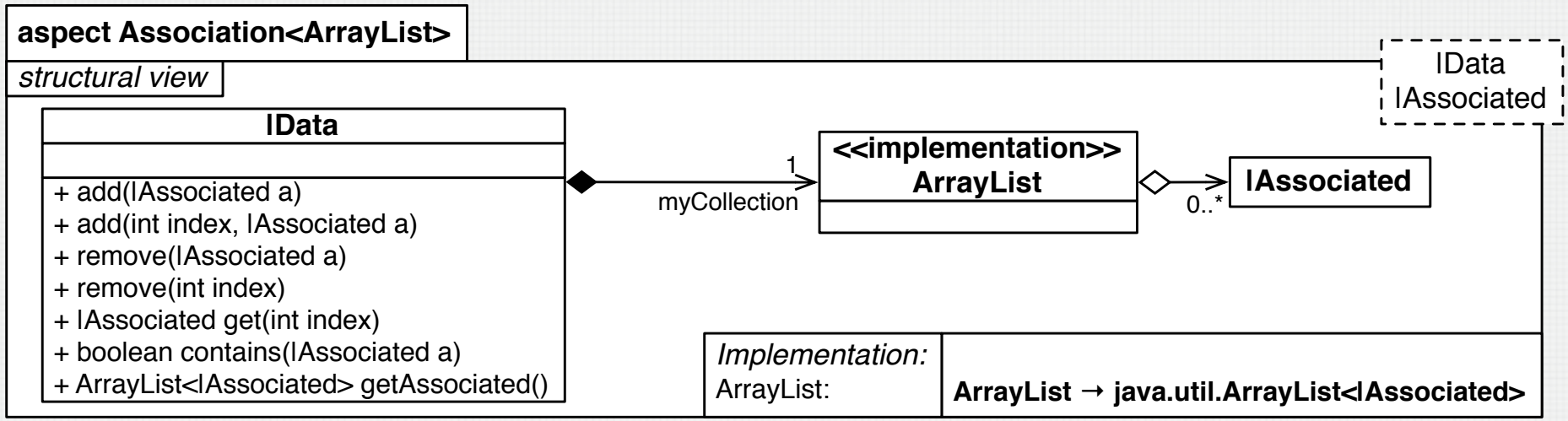
ICollection → ISequence; IforwardingMethod<ItargetMethod> → add(IAssociated)<add(IAssociated)>, add(int, IAssociated)<add(int, IAssociated), remove(IAssociated)<remove(IAssociated)>, remove(int)<remove(int)>, get<get>, contains<contains>

- RAM models within the same concern extend each other

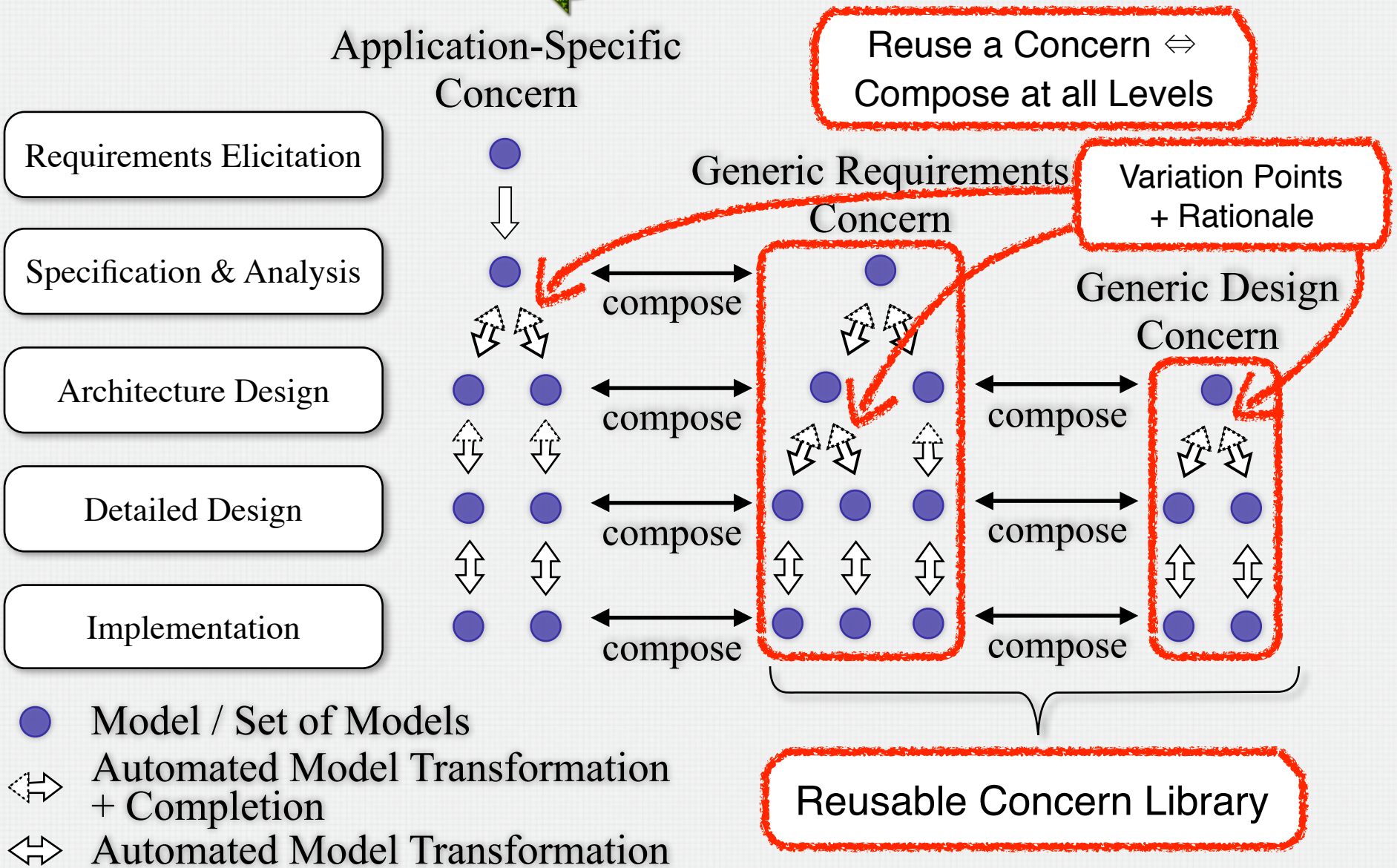
Feature: ArrayList



- If the user chooses to instantiate the Association concern with features Ordered and ArrayList, the following RAM model is generated:



The CORE Vision



The CORE Approach

- Requirements Elicitation
- Specification & Analysis
- Architecture Design
- Detailed Design
- Implementation

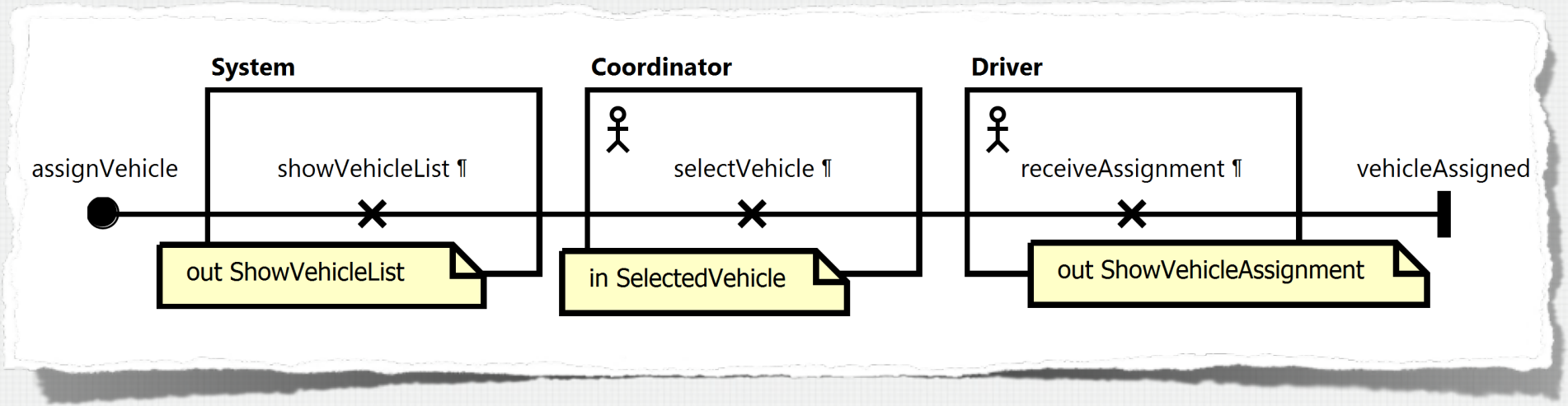
Aspect-oriented User Requirements Notation (AoURN)
- Goal Models (GRL)
- Workflow Models (Use Case Maps)

Reusable Aspect Models (RAM)
- Application-Specific Design Models
- RAM Workflow Concern
- Reusable Design Concern Models

Java Code

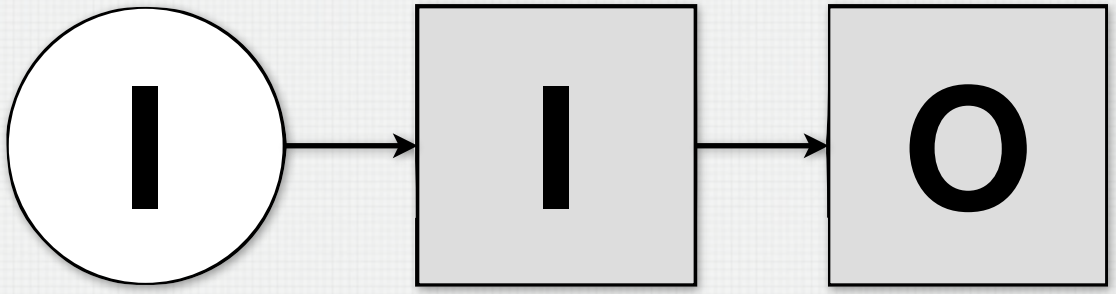
All of these Models of a Concern can be reused!

Assigning a Vehicle



System Operations / Workflow Steps

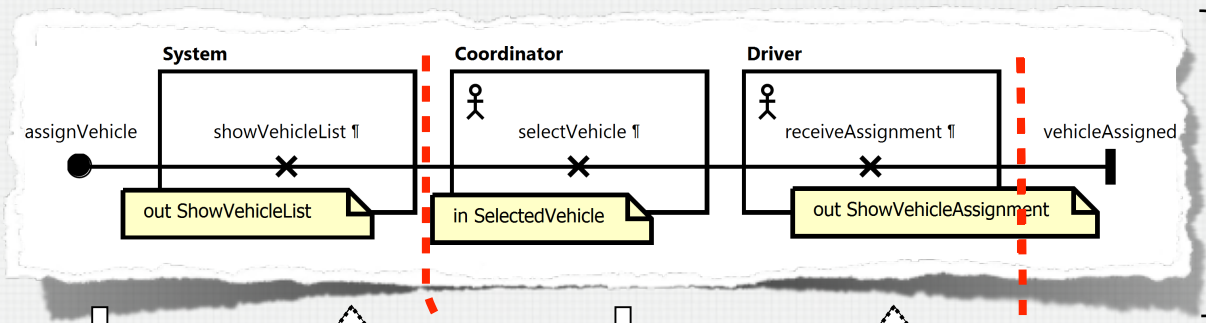
- System receives input
- System processes input
- System optionally produces output

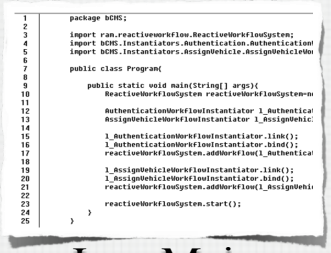


We call this a Step

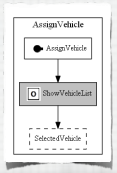
- Steps do not overlap!
- Complex steps can be triggered by several inputs and may produce multiple outputs

AVC: Transformations

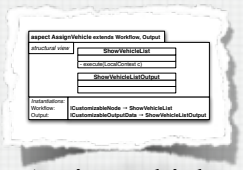




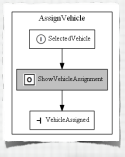
Java Main Program



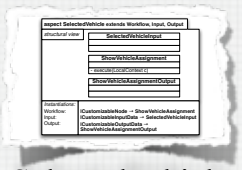
AssignVehicle Step View



AssignVehicle RAM Skeleton



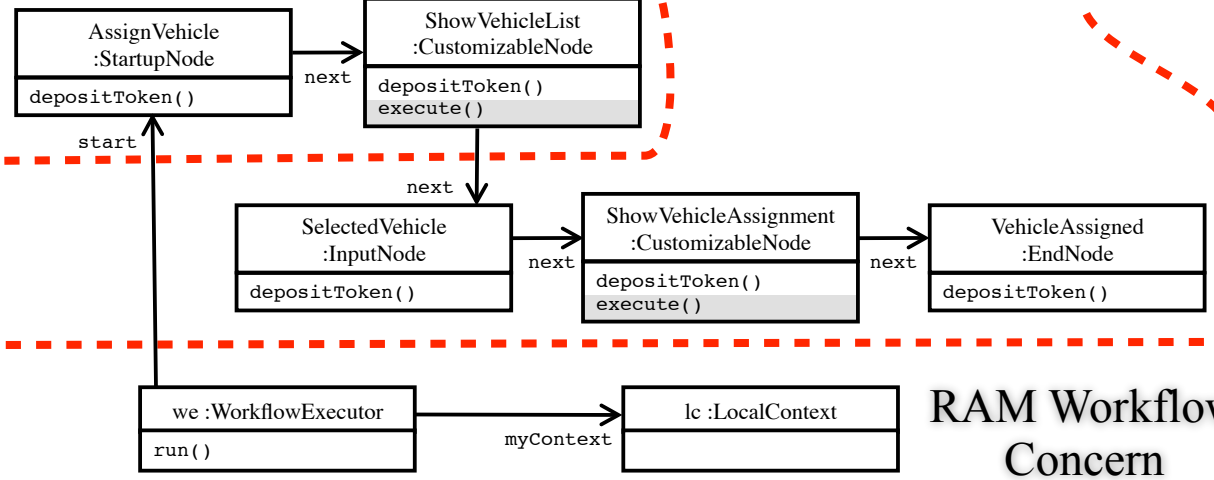
SelectedVehicle Step View



SelectedVehicle RAM Skeleton

compose

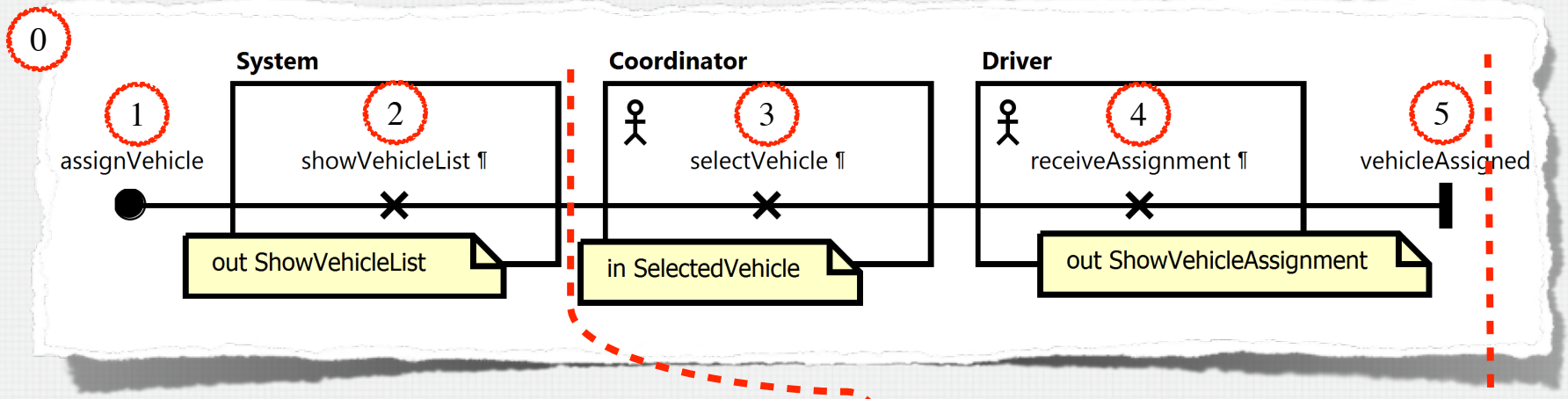
compose

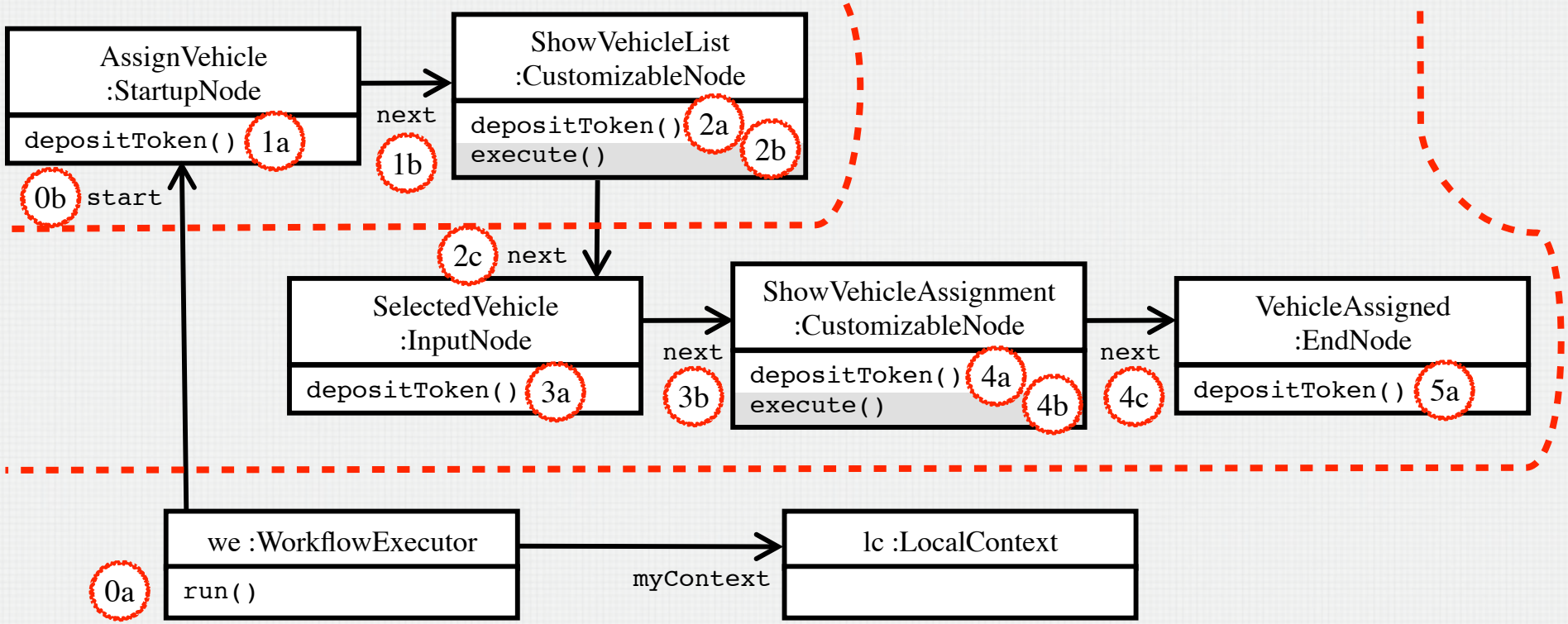


RAM Workflow Concern

Java Instantiation Code

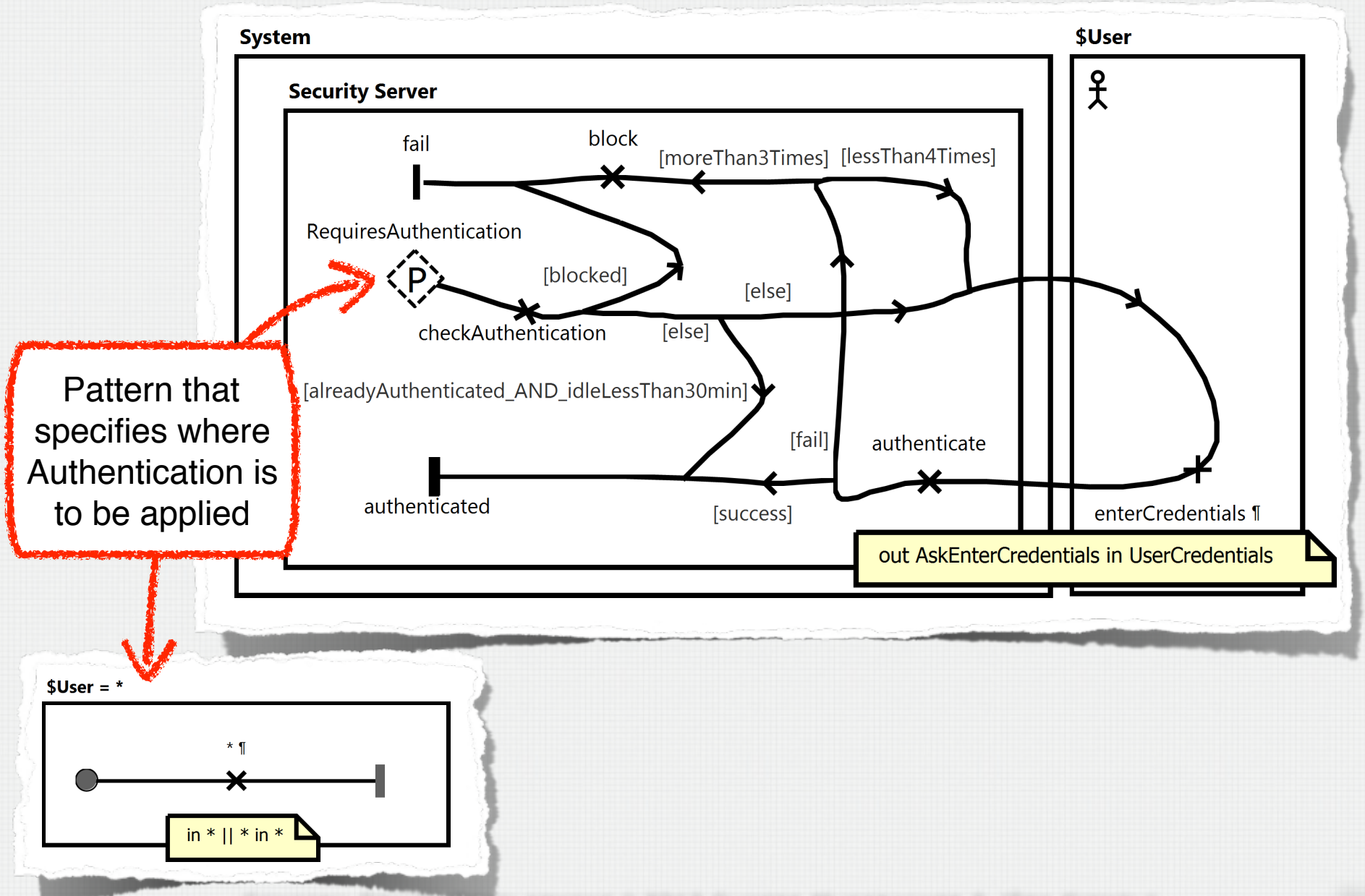
AVC: Transformations



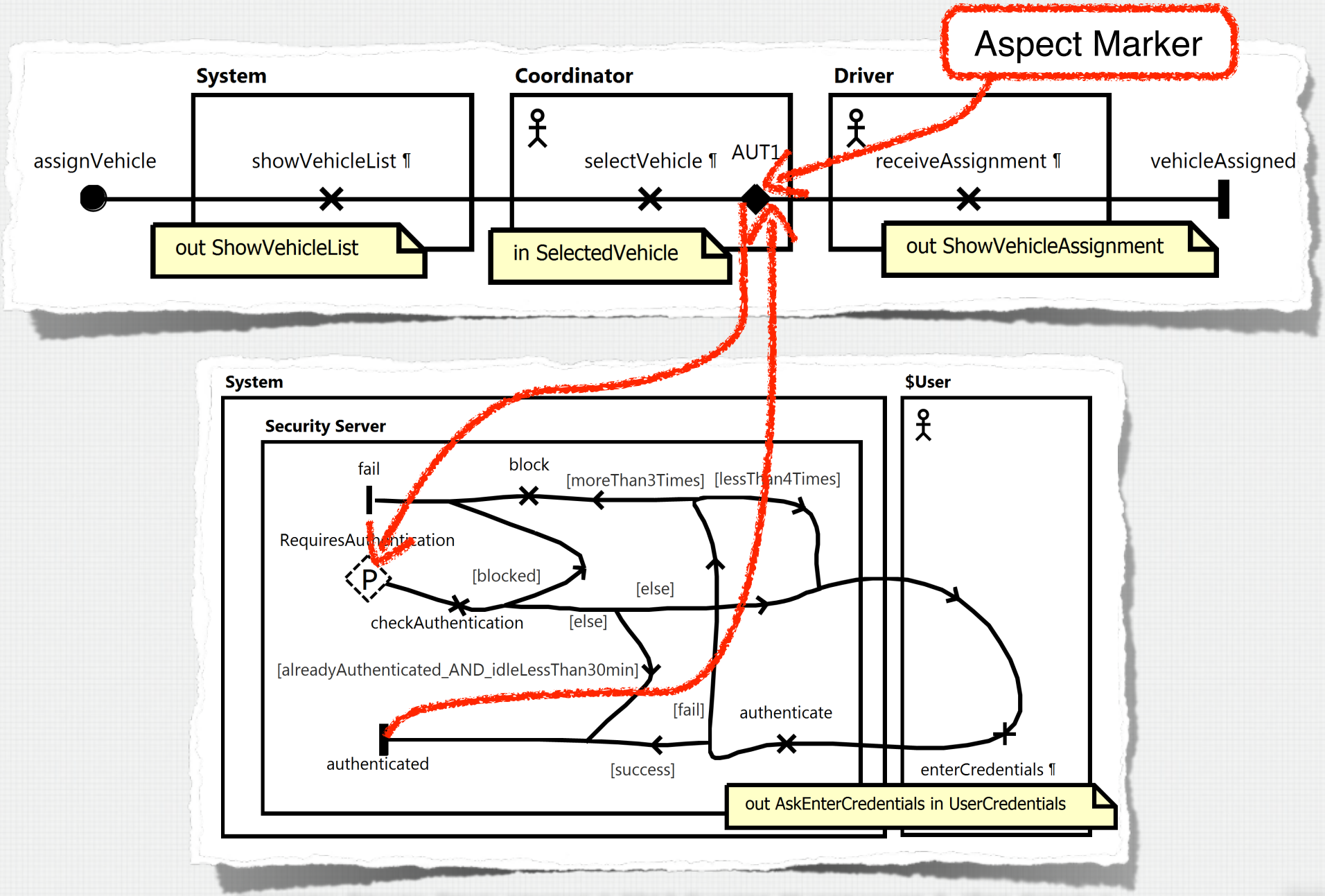


Authentication Concern (AC)

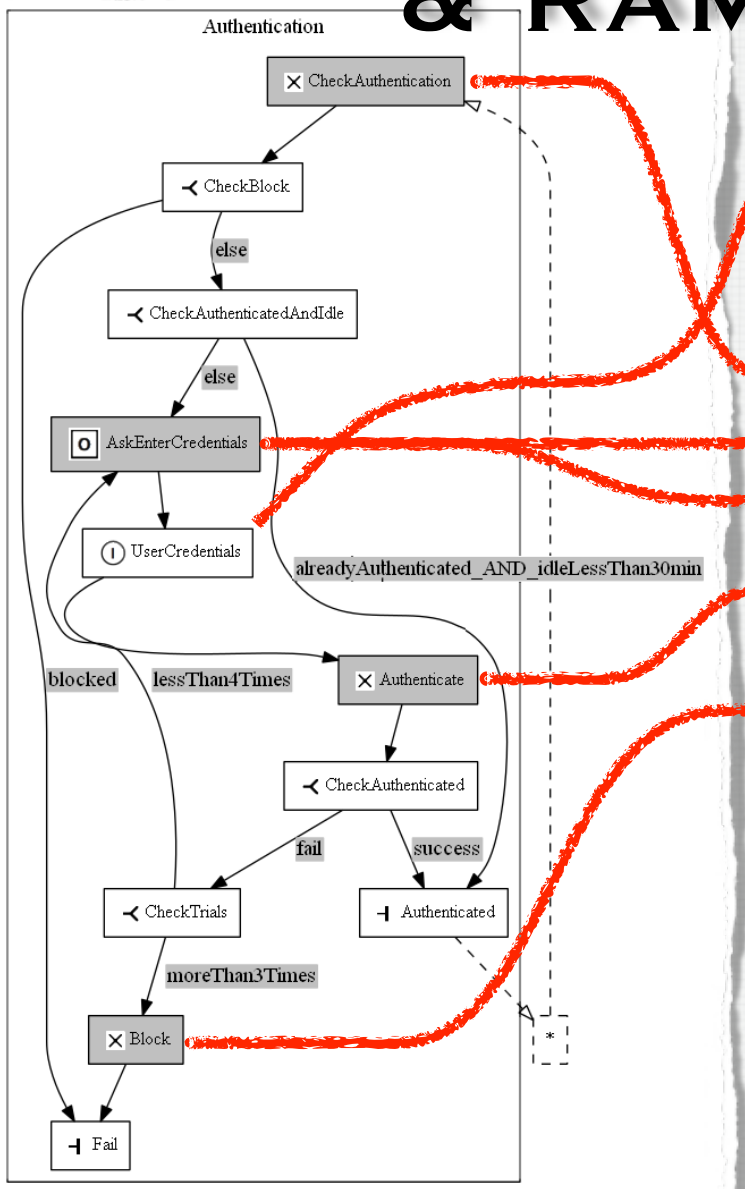
AC: AoURN Workflow Model



AC: Composed AoURN Workflow Model



AC: UserCredentials Step View & RAM Skeleton



aspect UserCredentials extends Input, Output, Workflow

structural view

CheckAuthentication
- execute(LocalContext c)

AskEnterCredentials
- execute(LocalContext c)

Authenticate
- execute(LocalContext c)

Block
- execute(LocalContext c)

UserCredentialsInput

AskEnterCredentialsOutput

Instantiations:

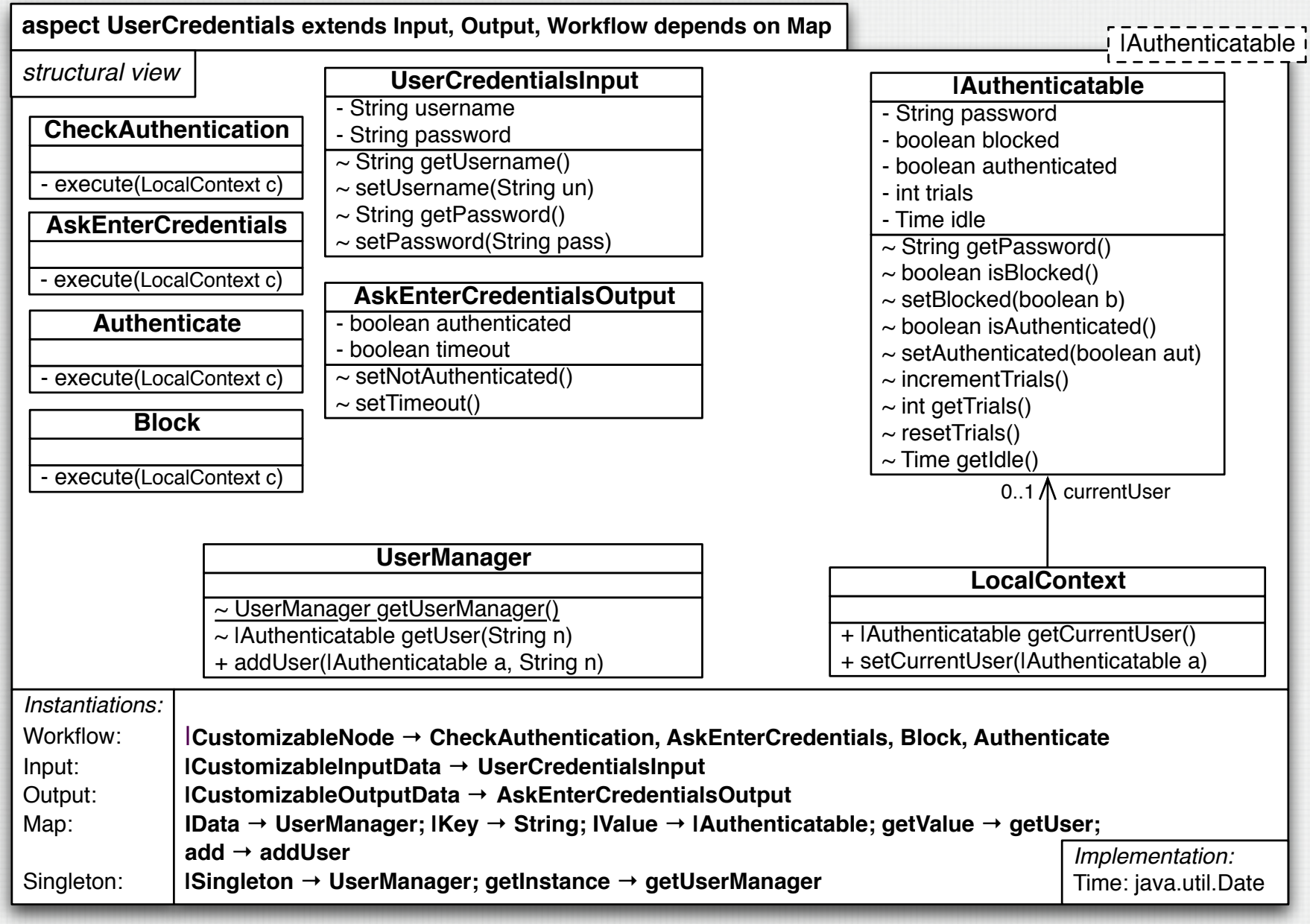
Workflow: **ICustomizableNode → CheckAuthentication, AskEnterCredentials, Block, Authenticate**

Input: **ICustomizableInputData → UserCredentialsInput**

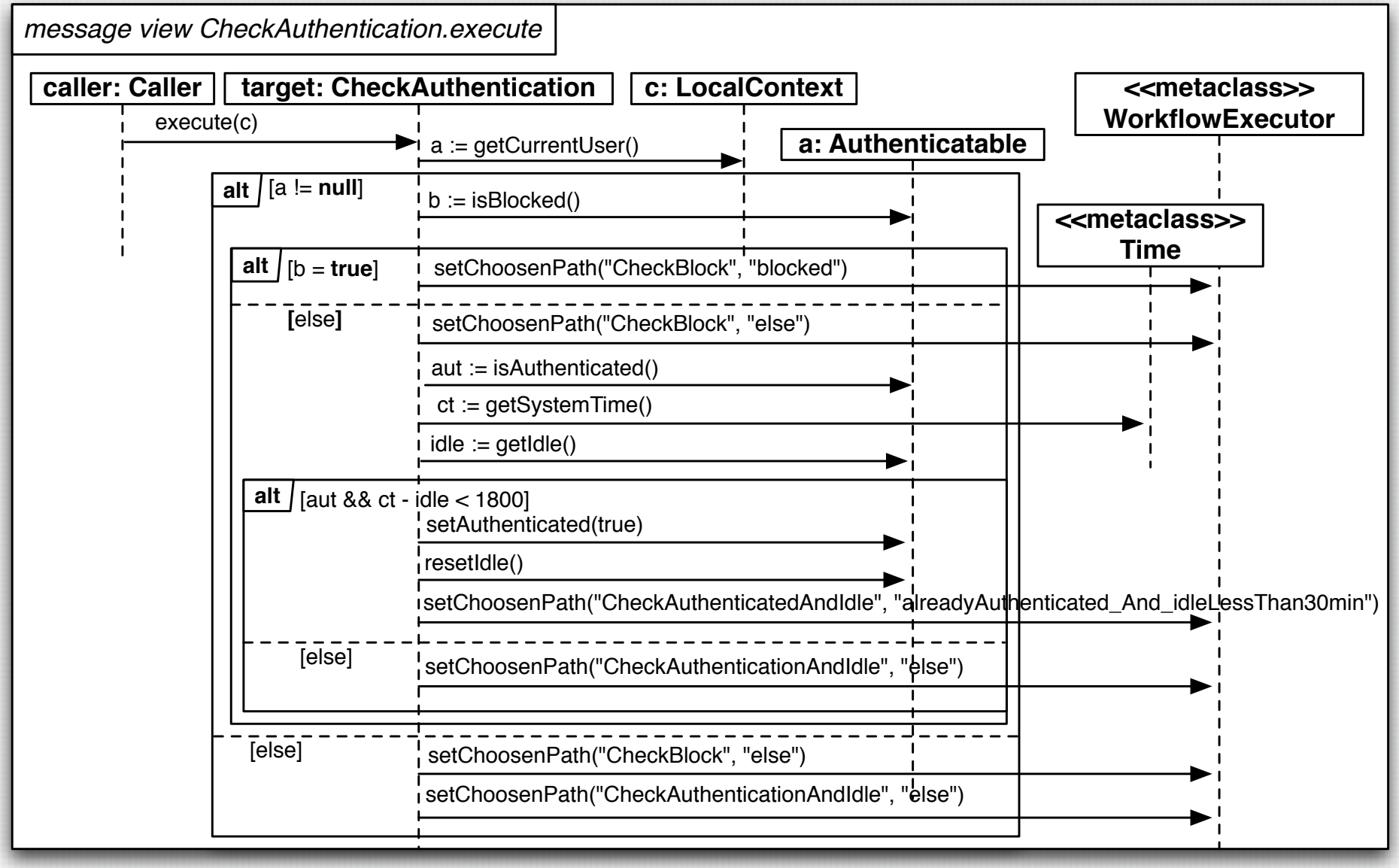
Output: **ICustomizableOutputData → AskEnterCredentialsOutput**

Workflow Design Concern Composition Rules

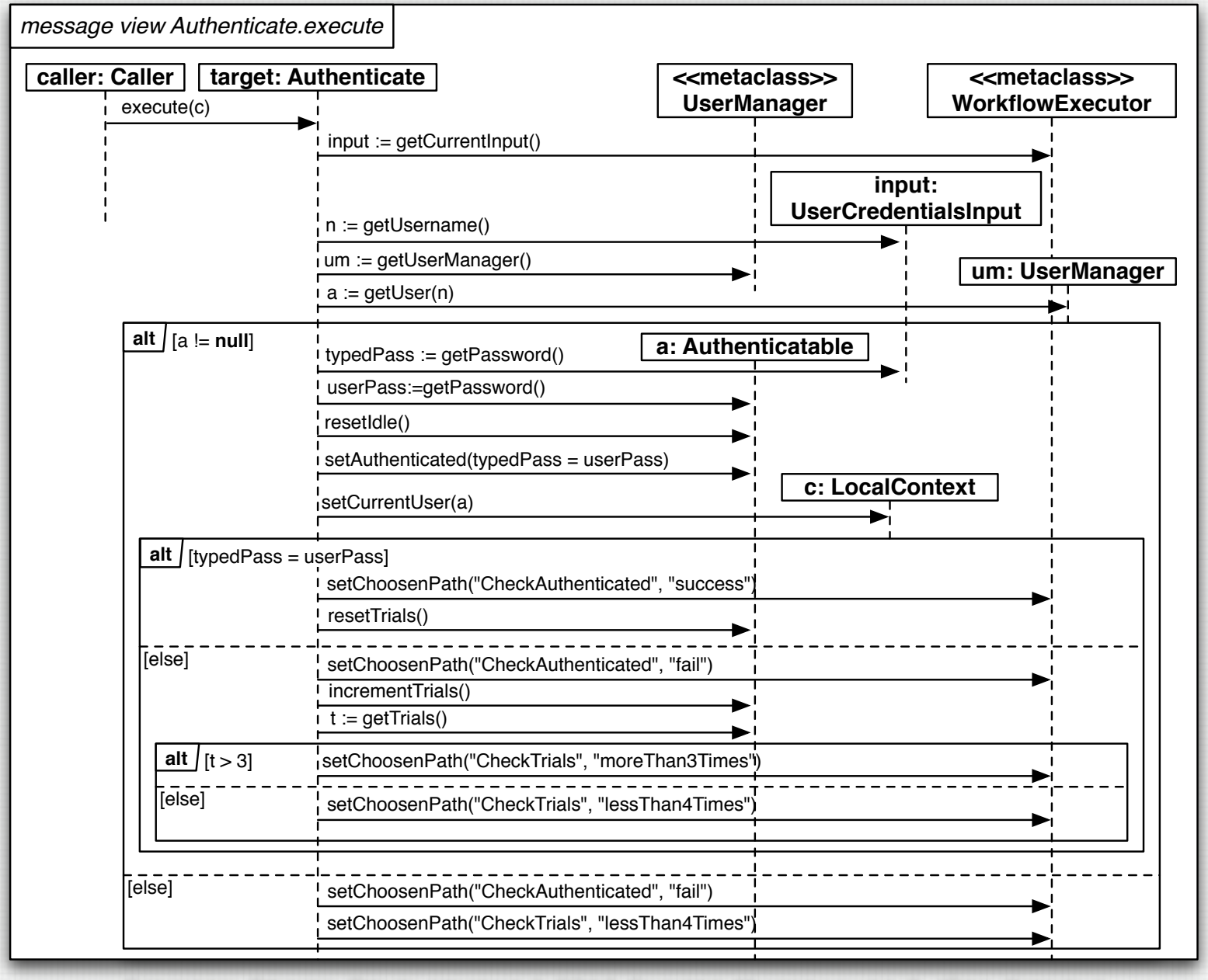
AC: Completed RAM Design



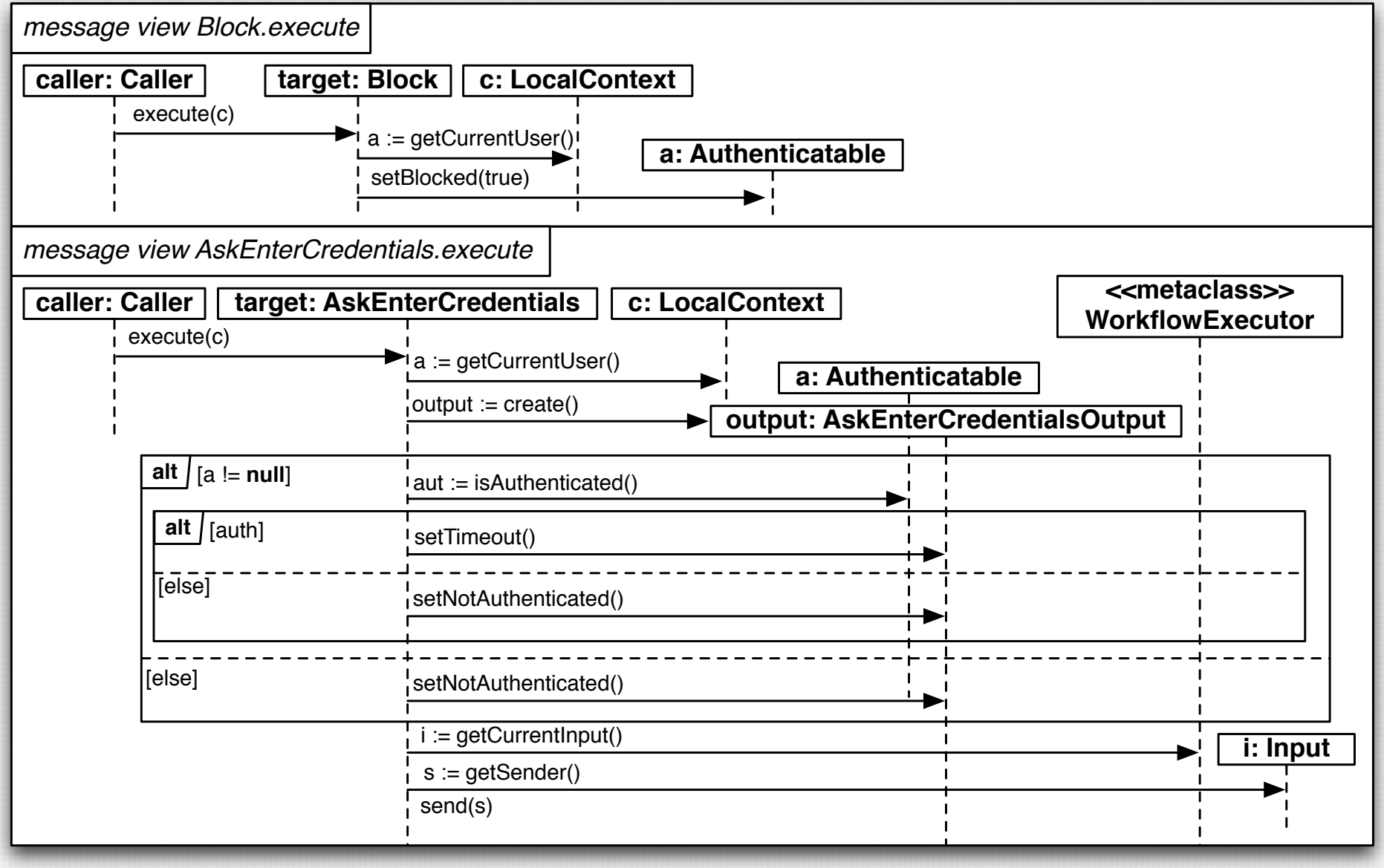
AC: Completed RAM Design



AC: Completed RAM Design



AC: Completed RAM Design



AC: Composable Reusable Concern

- At the RAM level (design phase), the customizable data defined by the Authentication concern (IAuthenticatable) must be composed with the application-specific entities that need to be authenticated

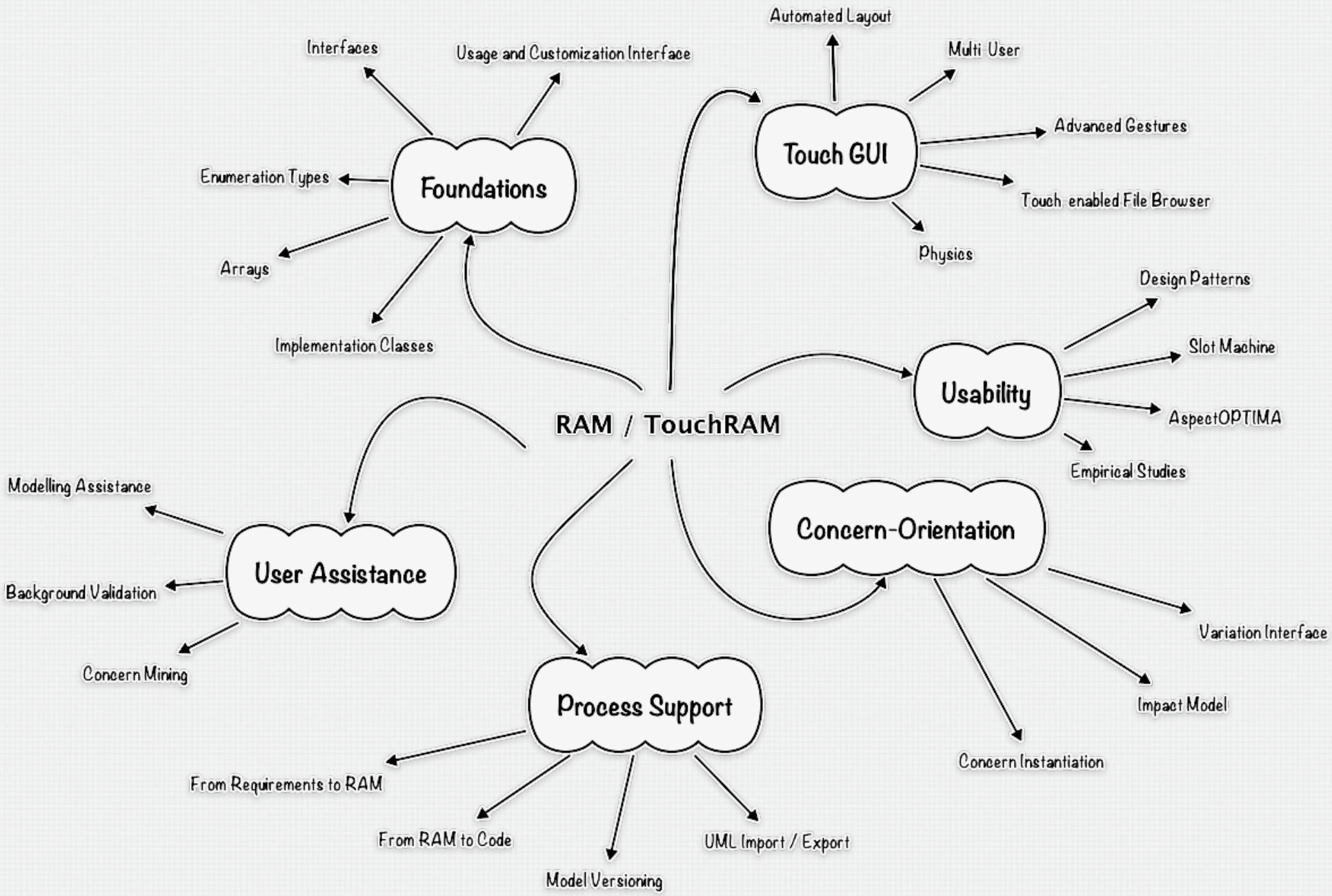
aspect bCMSDomainWithAuthentication extends bCMSDomain depends on UserCredentials	
structural view	
Instantiations: UserCredentials:	**IAuthenticatable → Coordinator, Driver**

- As the bCMSDomain concern grows, more entities might be defined that communicate with the system, and therefore also need to be authenticated

Concern-Orientation Summary

- Concern-oriented development applies model-driven engineering principles using concerns as the main means of structuring and organizing models throughout software development phases
- Allows software engineers to become concern-specialists

- CORE-RAM applies concern-orientation to the design and implementation phases of software development
- CORE is one example for concern-driven development
  - Workflow models for requirements
  - Design models (structural/state/message views)


RAM / TouchRAM
Foundations
Interfaces
Usage and Customization Interface
Enumeration Types
Arrays
Implementation Classes
Touch GUI
Automated Layout
Multi User
Advanced Gestures
Touch enabled File Browser
Physics
Usability
Design Patterns
Slot Machine
AspectOPTIMA
Empirical Studies
User Assistance
Modelling Assistance
Background Validation
Concern Mining
Concern-Orientation
Variation Interface
Impact Model
Concern Instantiation
Process Support
From Requirements to RAM
From RAM to Code
Model Versioning
UML Import / Export

Basic GUI for TouchRAM

- General
  - Resize buttons in accordance with Screen Size
  - Extend touch keyboard
  - Back button
  - Touch-enabled File Browser
- Structural View (Class Diagrams)
  - Support for re-ordering of Attributes
  - Support for re-ordering of Operations
  - Automated layout of class diagrams
  - Support for Physics
- General Split View

GUI-related TouchRAM Projects

- Multi-Touch
  - Recognizing advanced gestures
  - Using advanced gestures for model construction
  - Recognizing hands
    - Leap Motion
    - Kinect
- Multi-User
  - Recognizing users
    - Leap Motion
    - Kinect
  - Adapt gesture recognition for multi-users

TouchRAM Modelling Projects

- Structural View
  - Hide/Show Usage and Customization Interfaces
  - Add Support for Implementation Classes
  - Add Support for Arrays
  - Add Support for Enumeration Types
  - Add Support for Interfaces
  - Direct Support for Many Associations
- Message View Editor
  - Split-view editing
  - Selective weaving of operations
  - Operation refactoring
- State View Editor
  - Layout for State Views
  - Selective weaving of state machines

TouchRAM Modelling

- Support for Concern-Orientation
  - Variation Interface
    - Feature Diagram Visualization
    - Feature Diagram Editing
  - Impact Model
    - Goal Model Visualization
    - Goal Model Editing
    - Goal Model Evaluation
  - Concern instantiation
- UML import / export
- Model versioning
- Collaborative Modelling
  - TouchRAM Server
  - Concern Model Library “in the cloud”

TouchRAM Process Support

- Linking with Requirements
  - Aligning AoURN and RAM metamodels
  - From Domain Models to RAM Designs
  - Transformation of point cut expressions from AoUCM to RAM
  - Traceability
- Linking with Code
  - Generate Object-Oriented Code from Message Views
  - Generate Aspect-Oriented Code
    - AspectJ
    - Scala
  - Comparing OO and AO code
  - Interfacing with existing libraries and frameworks
    - Define concern-oriented interfaces
      - For GUI frameworks
      - For Spring

Using TouchRAM

- Designing Concern Models (with Impact Models)
  - Association
  - Networking
  - Workflows
- Complete Case Studies
  - Complete Concern-Oriented Design of the Slot Machine
  - Concern-Oriented Design of all Design Patterns
  - Concern-Oriented Design of AspectOPTIMA (transaction framework)
- Empirical User Studies
  - Usability
  - Understandability
  - Reuse
  - Agile design exploration

User Assistance

- Background validation
  - Consistency within a view
  - Consistency between views
    - Model-check message views and protocol views
- Modelling assistance
  - In popups, present only model elements that are applicable
    - Only lifelines that are visible
    - Only messages that correspond to protocol
  - Suggest splitting of models into separate features if appropriate
- Concern mining
  - Suggest concerns that seem applicable

Questions