UNIVERSITY
OF OSLO

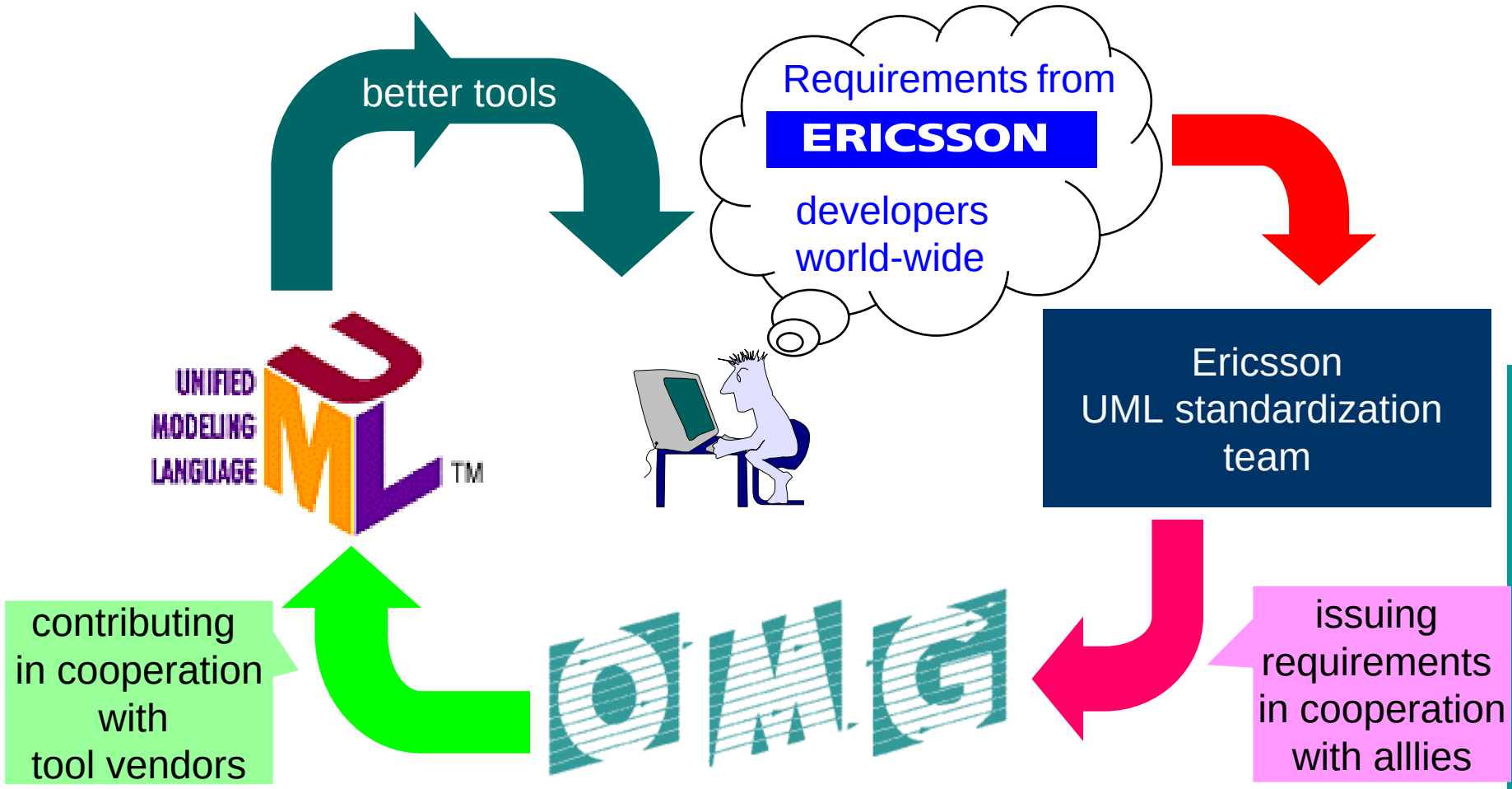
UML 2.0

Components,
System Family Modeling,
Domain Specific Languages,
and MDA

Birger Møller-Pedersen

University of Oslo, Norway

UML standardization within OMG – for Ericsson



U2 Partners

- Submitters
  - Alcatel, CA, Ericsson, Fujitsu, HP, IBM, I-Logix, IONA, Kabira, Motorola, Oracle, Rational, SOFTEAM, Telelogic, Unisys
- Supporters
  - Advanced Concepts Center LLC, Ceira Technologies, Commissariat à L'Energie Atomique, Compuware, DaimlerChrysler, Embarcadero Technologies, Enea Data, France Telecom, Fraunhofer FOKUS, Gentleware, Intellicorp, Jaczone, Kennedy Carter, Klasse Objecten, KLOCwork, Lockheed Martin, Mercury Computer, MSC.Software, Northeastern University, Popkin Software, Proforma, Sims Associates, Syntropy Ltd., Sun Microsystems, University of Kaiserslautern, University of Kent, VERIMAG, WebGain, and 88solutions

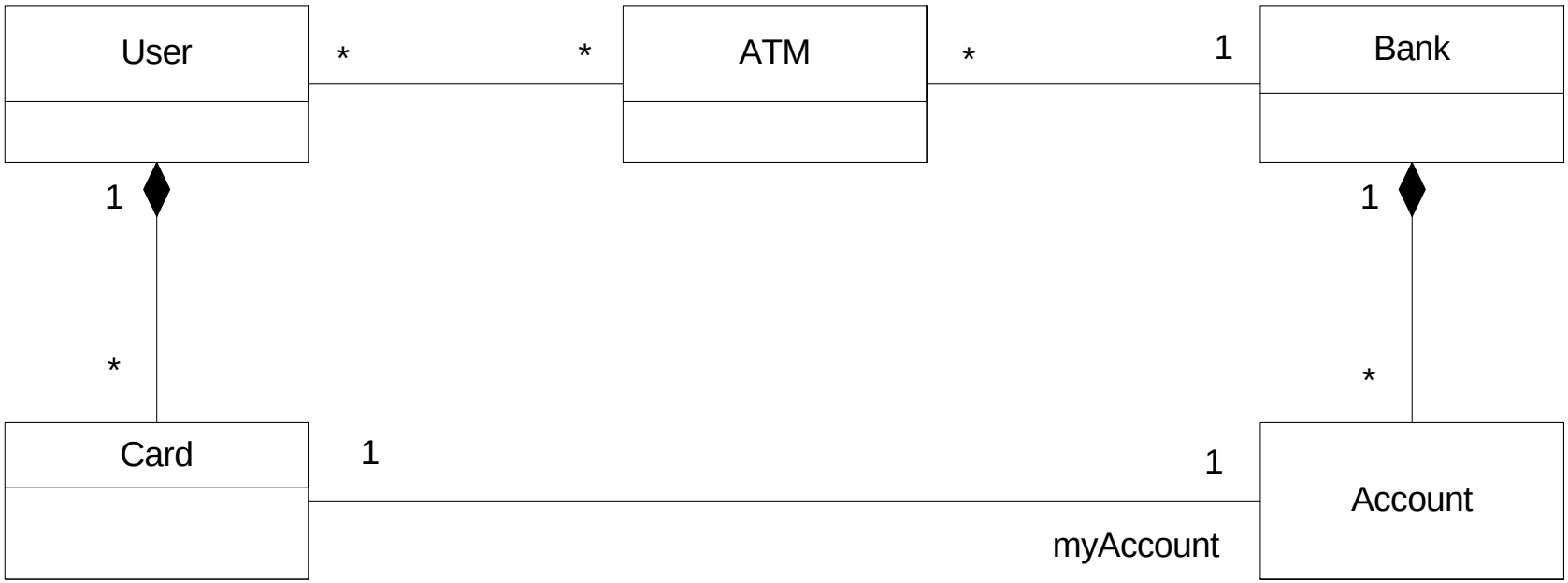
Background – modeling architecture of systems

- BBD: Blackboard Block Diagrams
- SDL (agents, gates, channels, ...)
- ROOM/UML-RT (capsules, ports, connectors, ...)
- ADLs (components, intefaces, connectors)

- for Ericsson, Motorola, Alcatel, Nokia (telecom, realtime)
  - SDL only one vendor
  - UML-RT (RoseRT) only one vendor

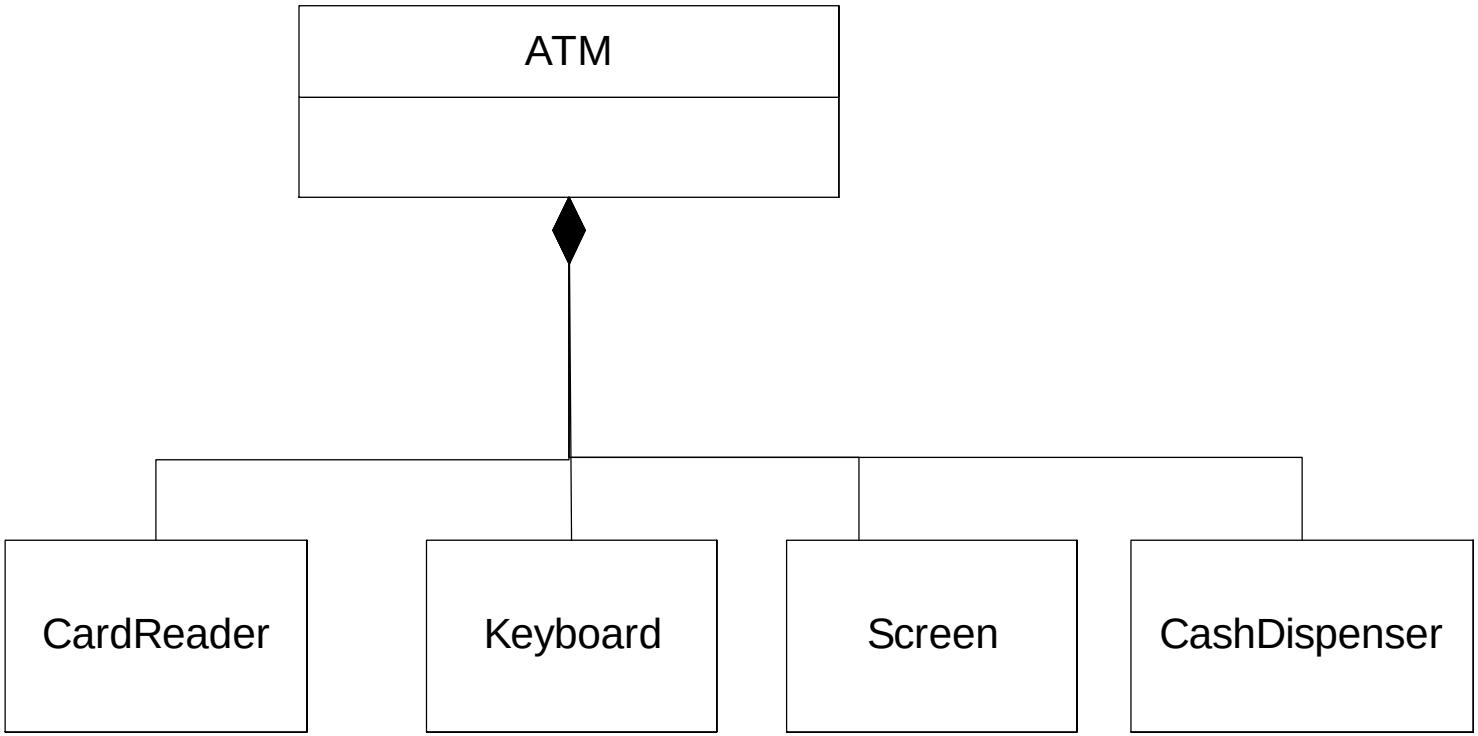
- UML2.0 combining features from these

ATM: Domain Model I



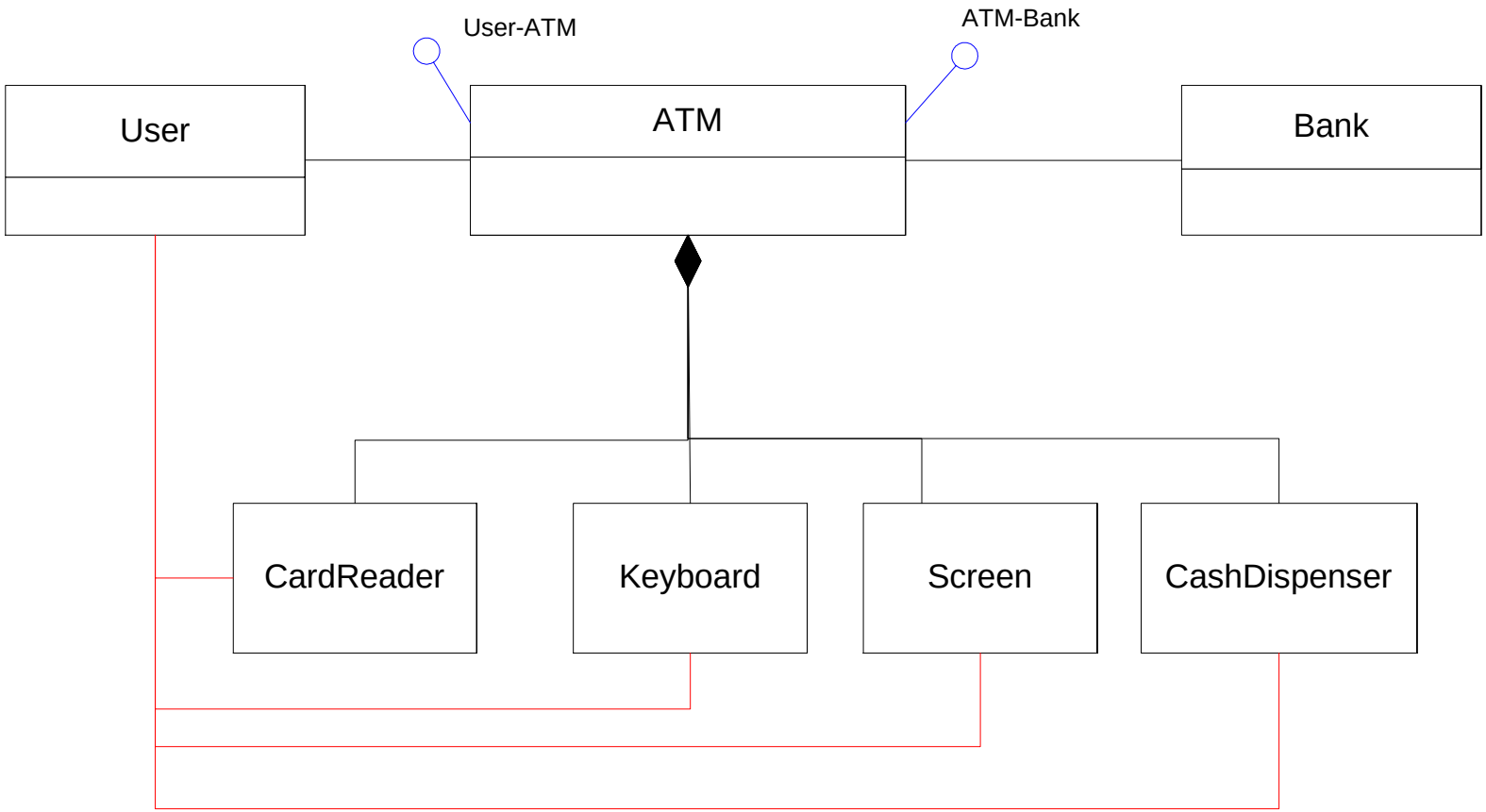
From
Haugen, Ø., B. Møller-Pedersen, and T. Weigert, Modeling Embedded Systems in UML 2.0, in *The Embedded Systems Handbook*, Richard Zurawski, Editor. 2005, CRC Press.

ATM: Domain Model II



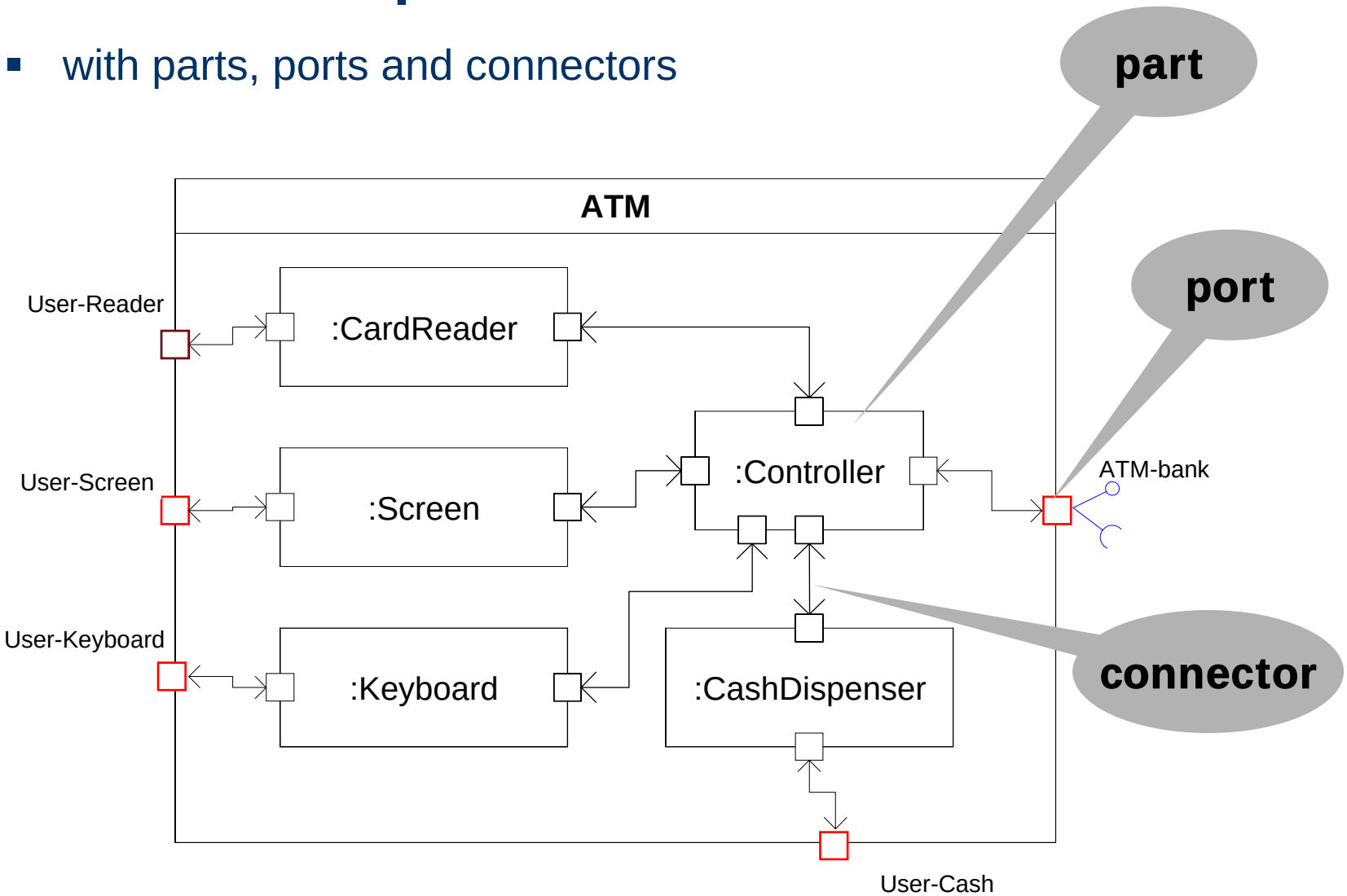
Context model with UML1.x class diagrams

- with plain composition and no encapsulation
- with only provided interfaces on classes



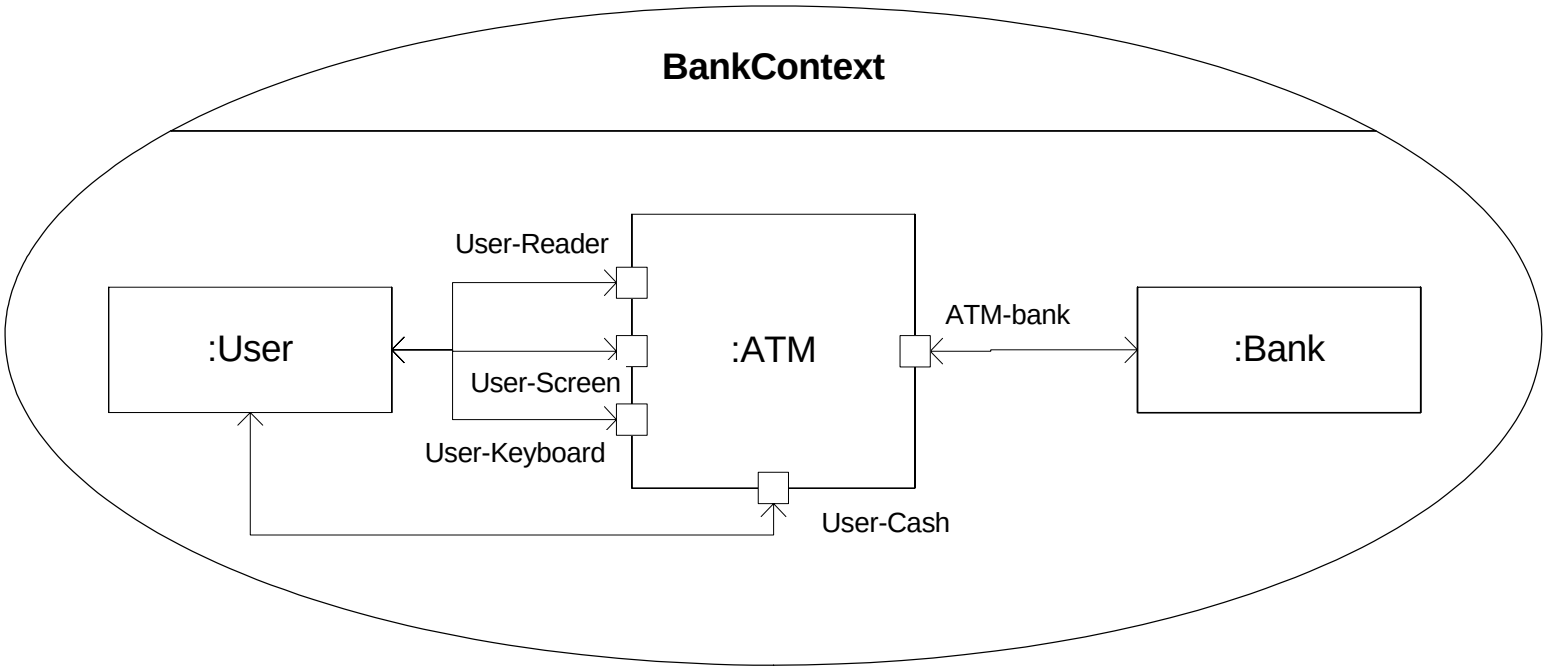
UML2.0: Composite Class

- with parts, ports and connectors



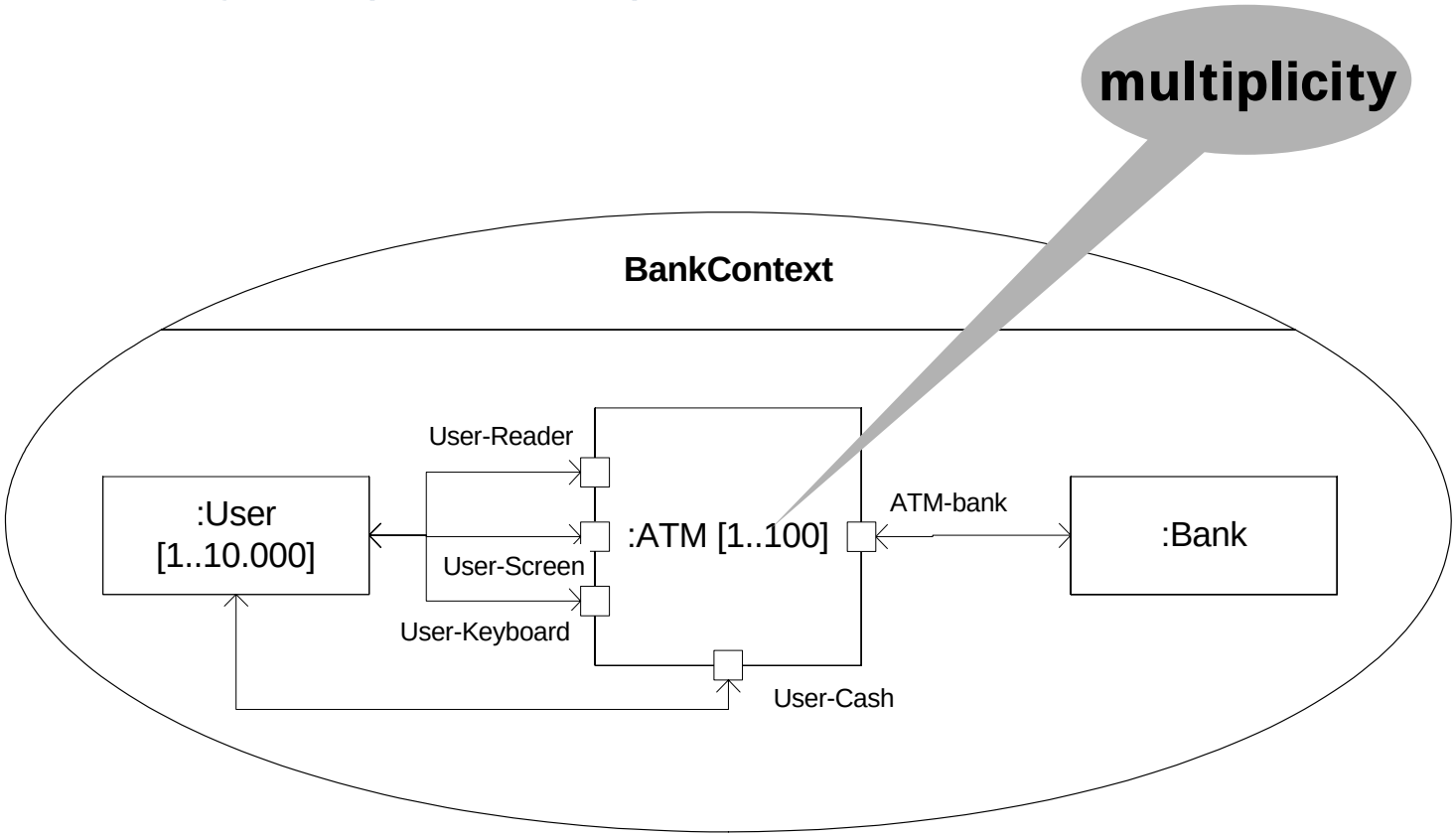
Context Models in UML2.0 - I

- composite structures also as part of Collaborations

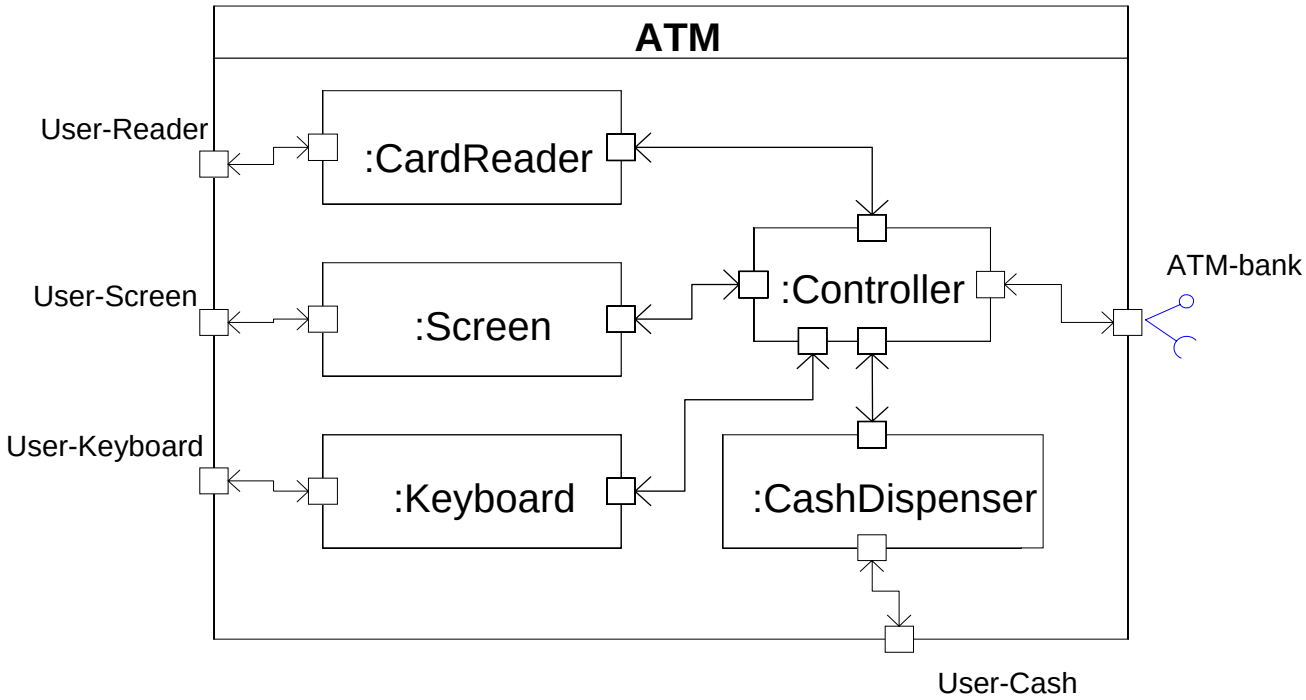


Context Models in UML2.0 - II

- including multiplicities on parts



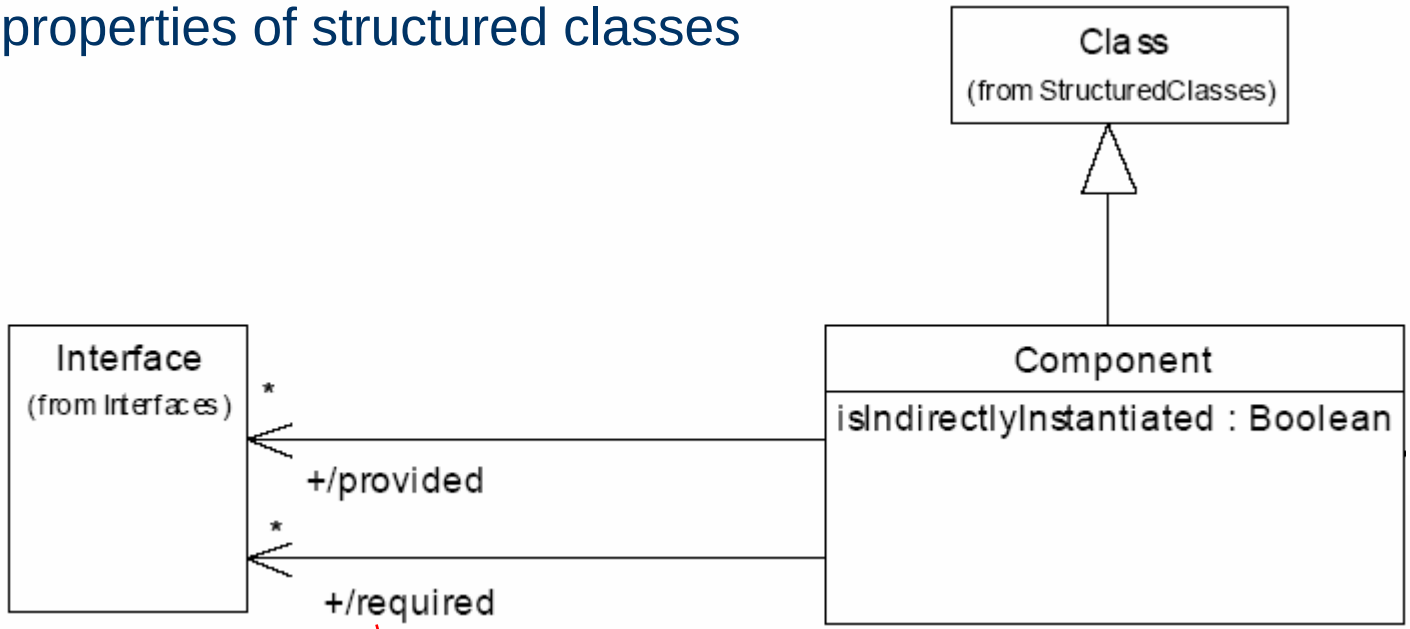
Structured Classes are like other Classes



- Structured Classes may have
  - attributes & operations, interfaces, …
- Internal structure is inherited, inherited parts may be redefined by extension

What about Components?

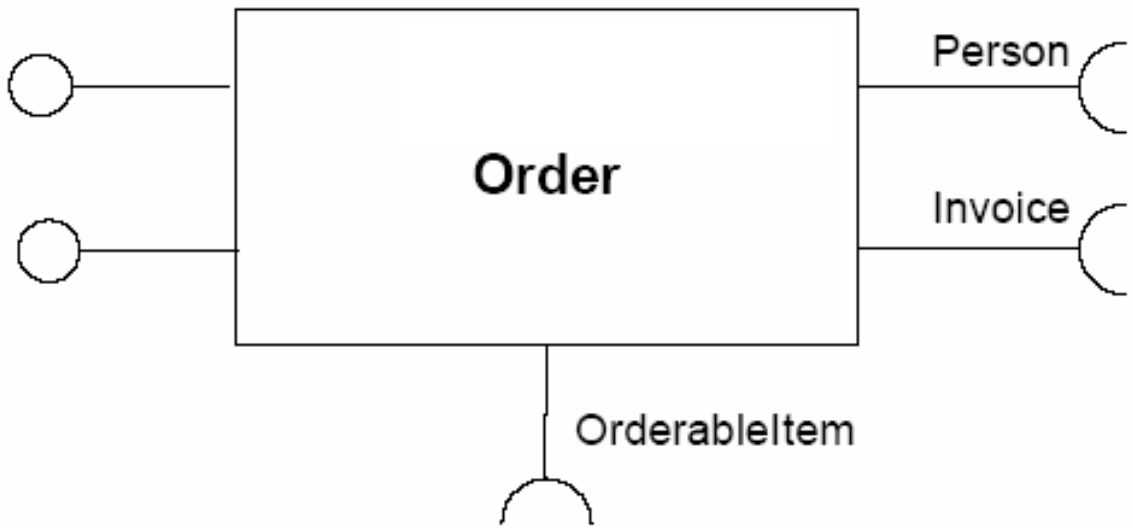
- Components have all the properties of structured classes



Note that these are just derived, that is they are also defined for classes

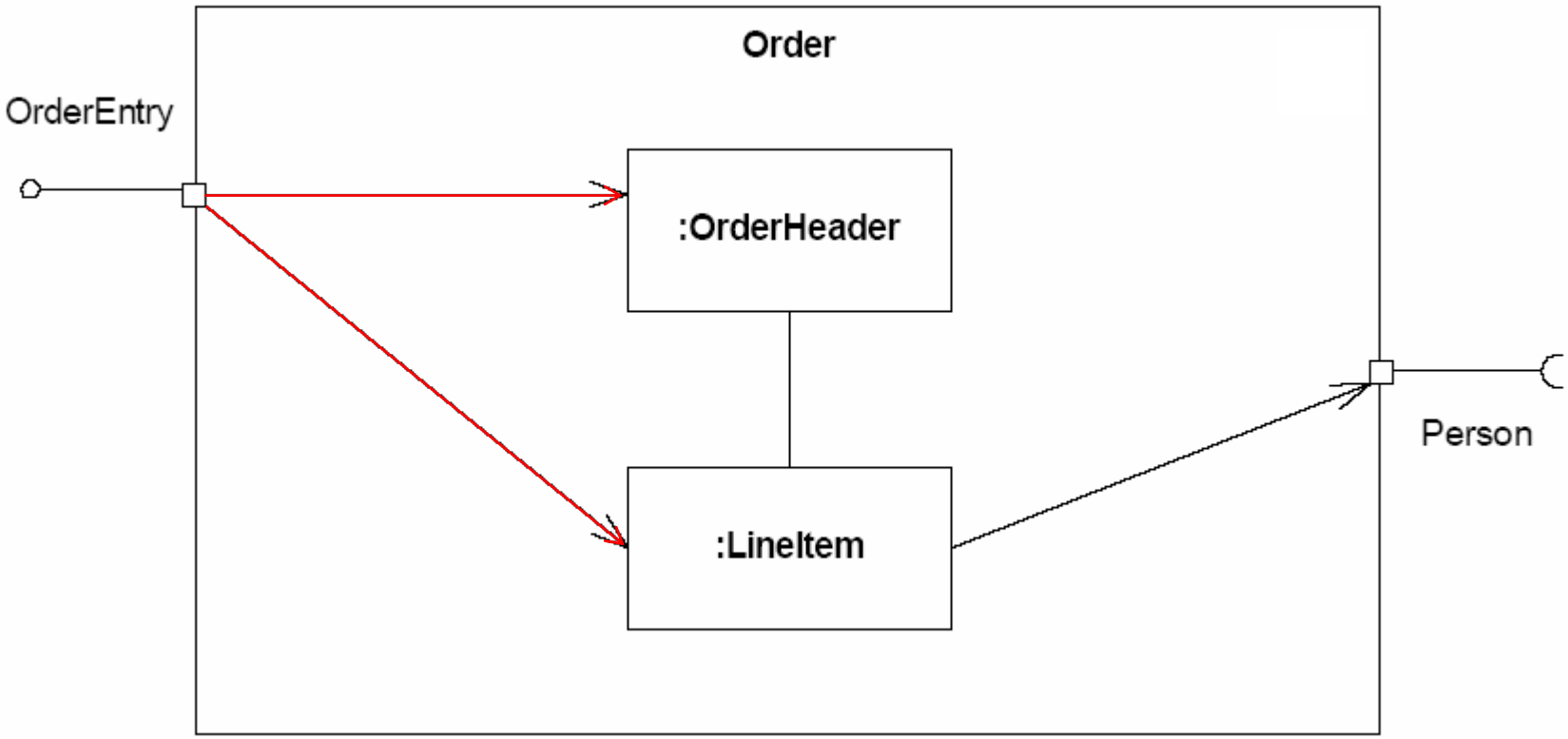
Notation

- ... and for classes



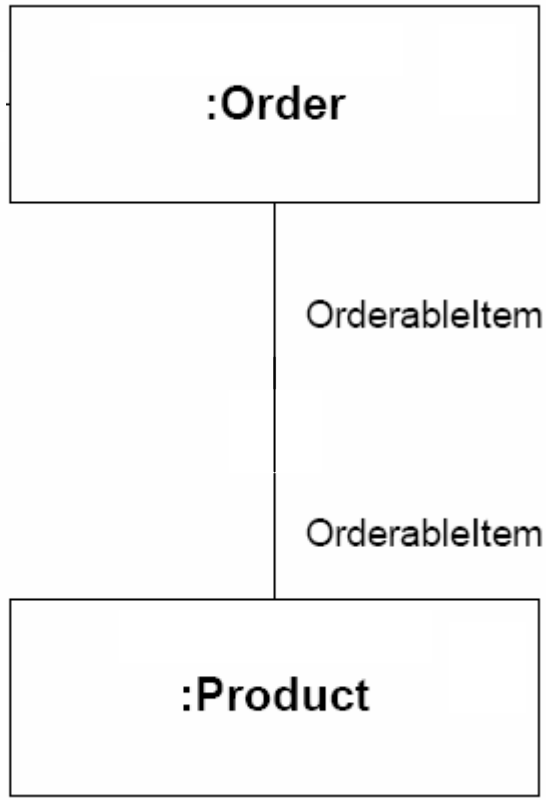
Delegation connector

- ... and for classes



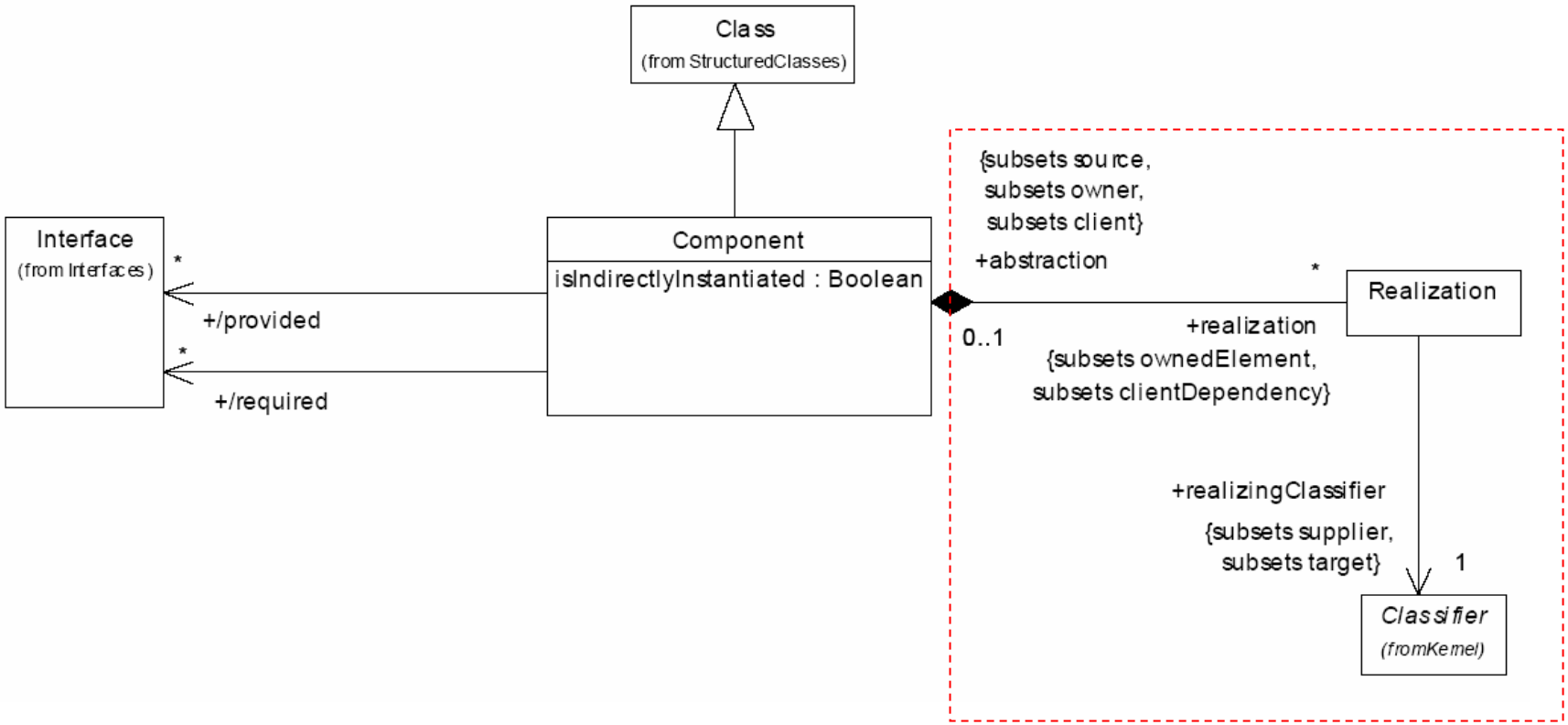
Assembly connectors

- ... and between class-defined parts



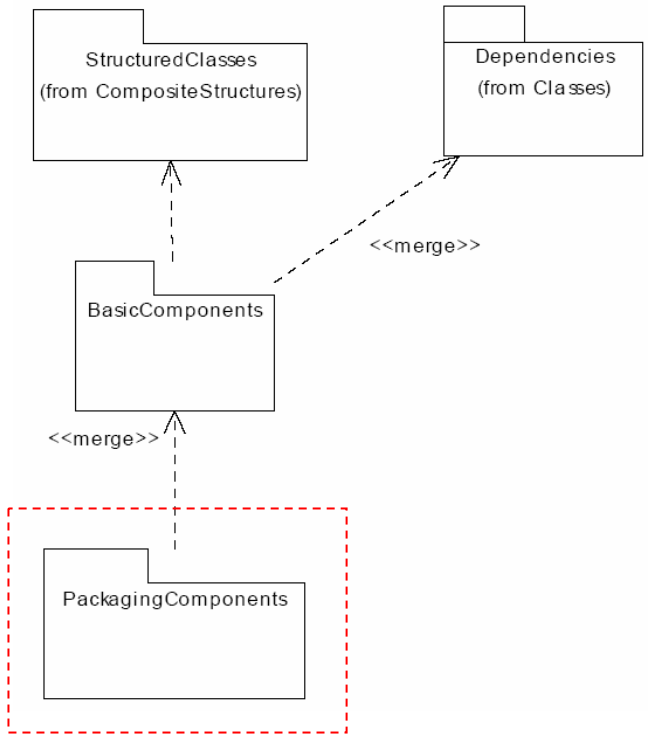
What is special for Components

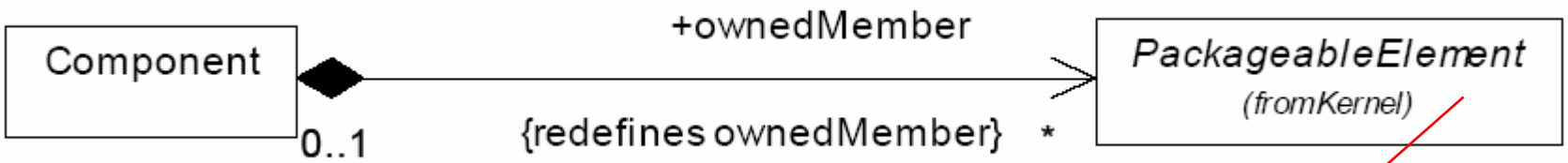
- Realization by a number of classes



... and

- may be kind of 'package', i.e. it may have model elements that you would not have for classes

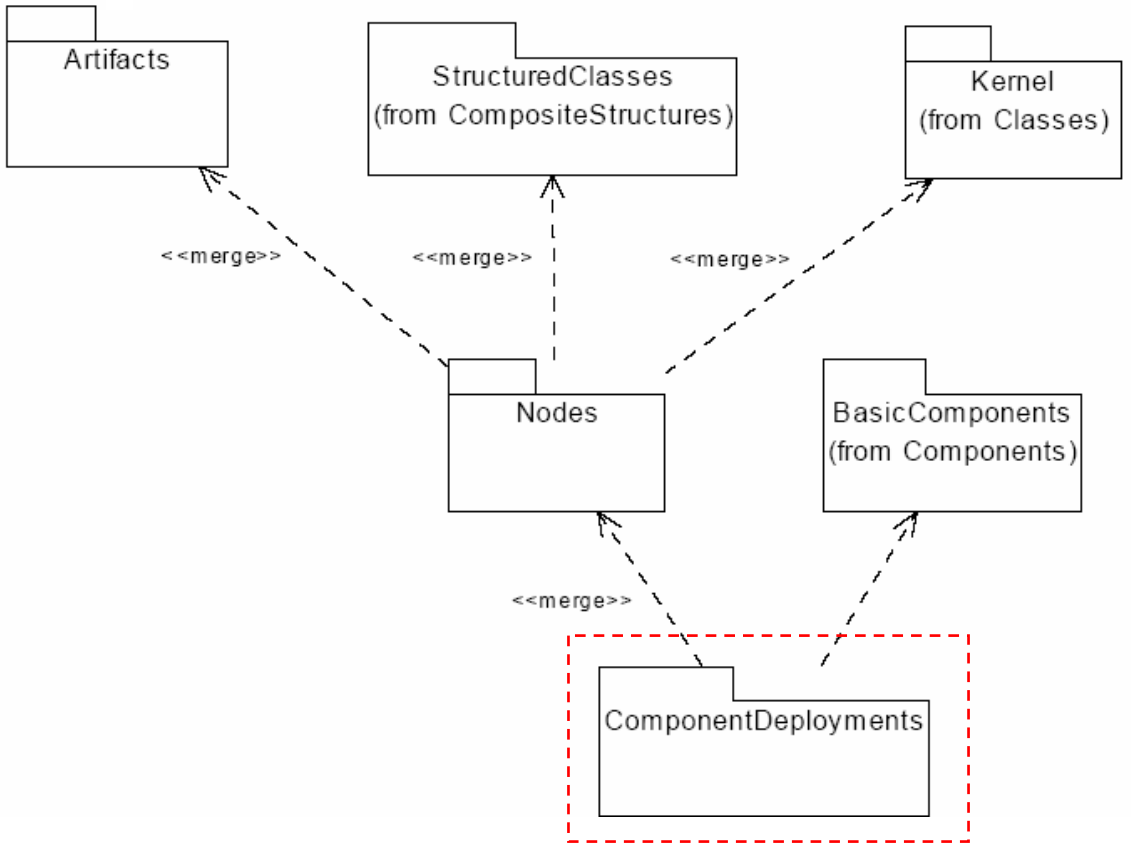




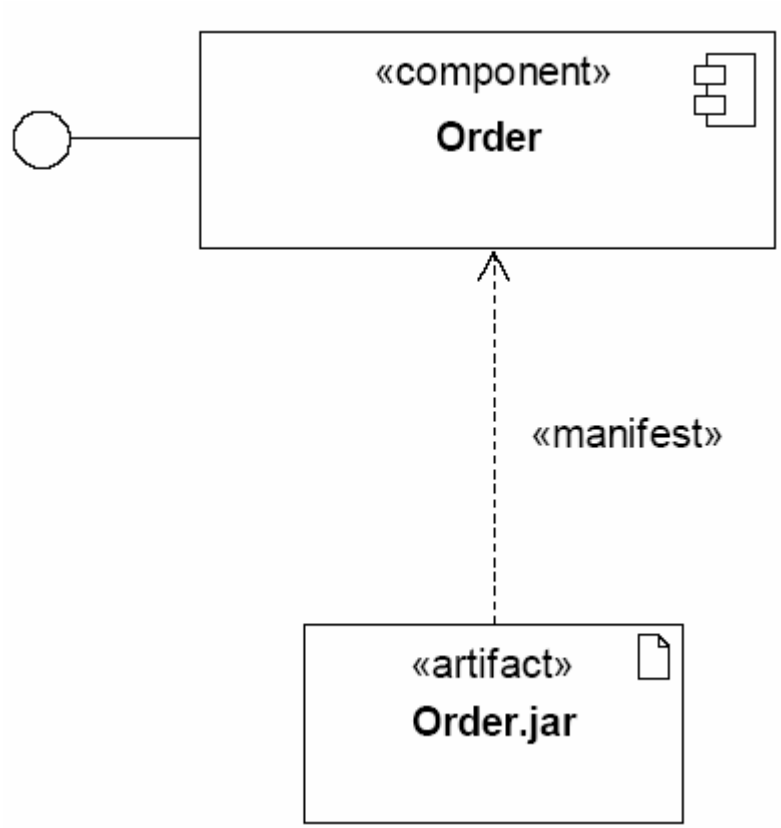
A component may have e.g. use cases, sequence diagrams, packages, dependencies, components

Deployment of components

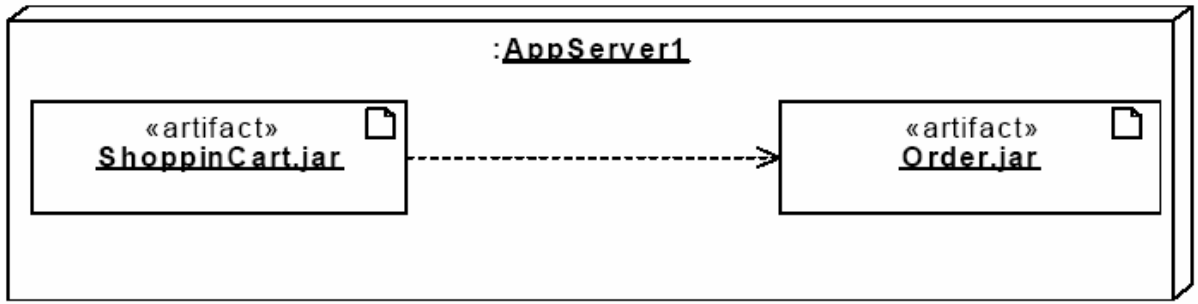
- Artifacts,
- Nodes,
- Network of Nodes,
- ...

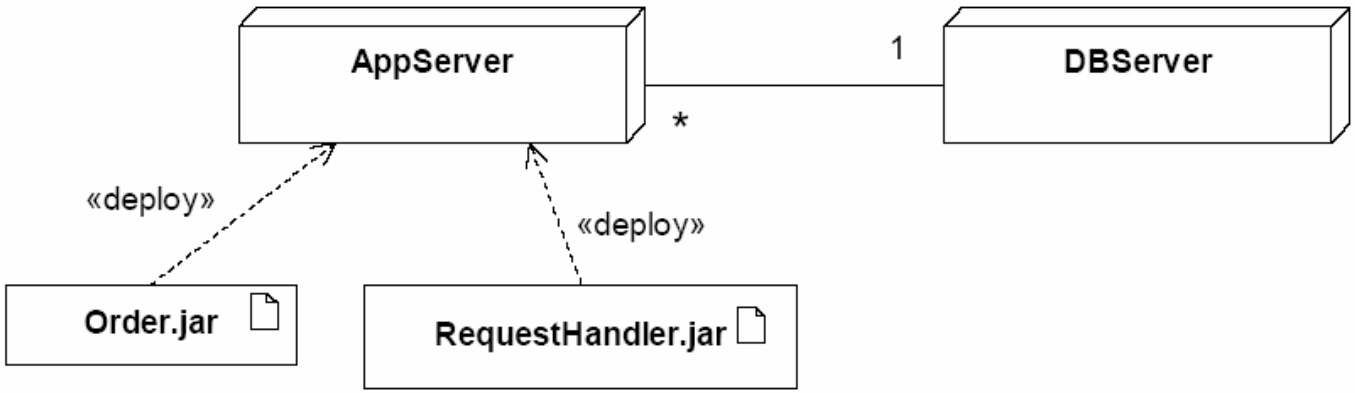


Artifacts



Nodes with deployed artifacts





Must be profiled for actual component models

Table 29 - Example Profile for Enterprise Java Beans

Stereotype	Base Class	Parent	Tags	Constraints
EJBEntityBean «EJBEntityBean»	Component	N/A	N/A	N/A
EJBSessionBean «EJBSessionBean»	Component	N/A	N/A	N/A
EJBMessage DrivenBean «EJBMessage DrivenBean»	Component	N/A	N/A	N/A
EJBHome «EJBHome»	Interface	N/A	N/A	N/A
EJBRemote «EJBRemote»	Interface	N/A	N/A	N/A

Must be profiled for actual component models

Table 31 - Example Profile for .NET Components

Stereotype	Base Class	Parent	Tags	Constraints
NetComponent «NetComponent»	Component	N/A	N/A	N/A
NETProperty «NETProperty»	Property	N/A	N/A	N/A
NETAssembly «NETAssembly»	Package	N/A	N/A	N/A
MSI «MSI»	Artifact	N/A	N/A	N/A
DLL «DLL»	Artifact	«file»	N/A	N/A
EXE «EXE»	Artifact	«file»	N/A	N/A

Summary

- Components = composite classes + deployment
- i.e. composite structures (parts, ports and connectors) not specific to components!
  - Some reasons why this is a god idea
    - Even analysis models may be structured
    - Models may be deployed (e.g. on real-time or embedded platforms) even without component technology
    - Example: System family modeling based upon plug-ins (but not using components)
  - In general: users of UML tend to
    - tailor UML to whatever they need, except MicroSoft: Domain Specific Languages
    - transform models to whatever platform they are using (the essence of MDA)

System Family Modeling (Product Line Modeling)

- The modeling of architecture is important
  - including the modeling of commonalities and variations
  - independent of whether components are used or not

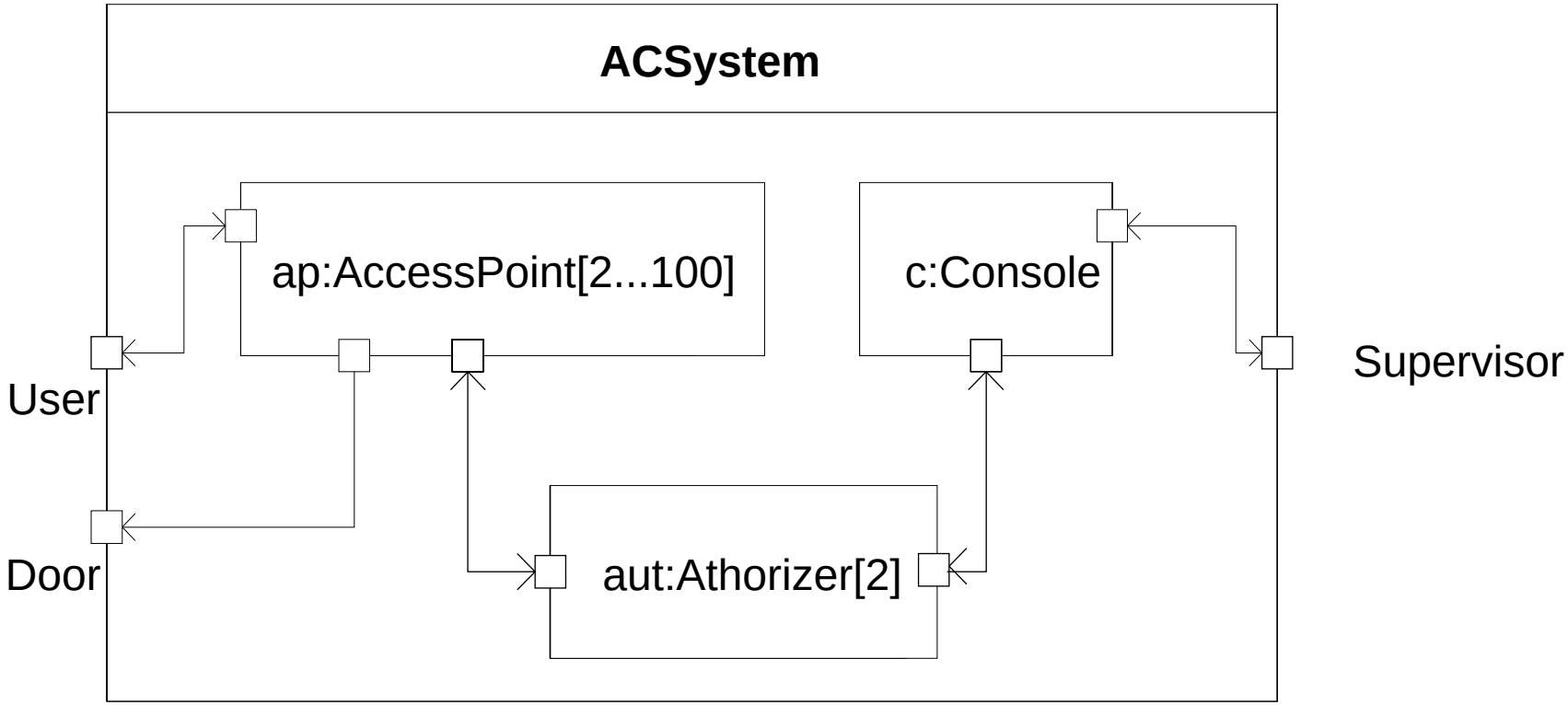
  - Approaches
    - Frameworks with hot-spots/plug-ins
    - Feature models: ’Stereotyping’ variation model elements
    - Templates

Eureka Σ! 2023 Programme

ITEA project ip02009

FAMILIES

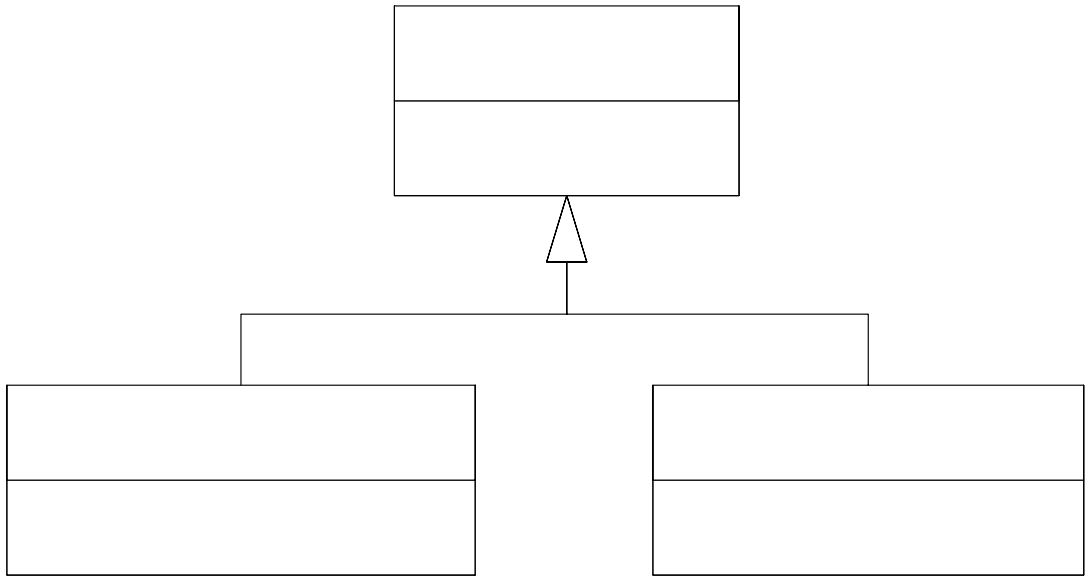
Example – Access Control System



From
Haugen, Ø., B. Møller-Pedersen, and T. Weigert, *Structural Modeling with UML 2.0*, in *UML for Real*, L. Lavagno, G. Martin, and B. Selic, Editors. 2003, Kluwer Academic Publishers, Boston.

kluwer
the language of science

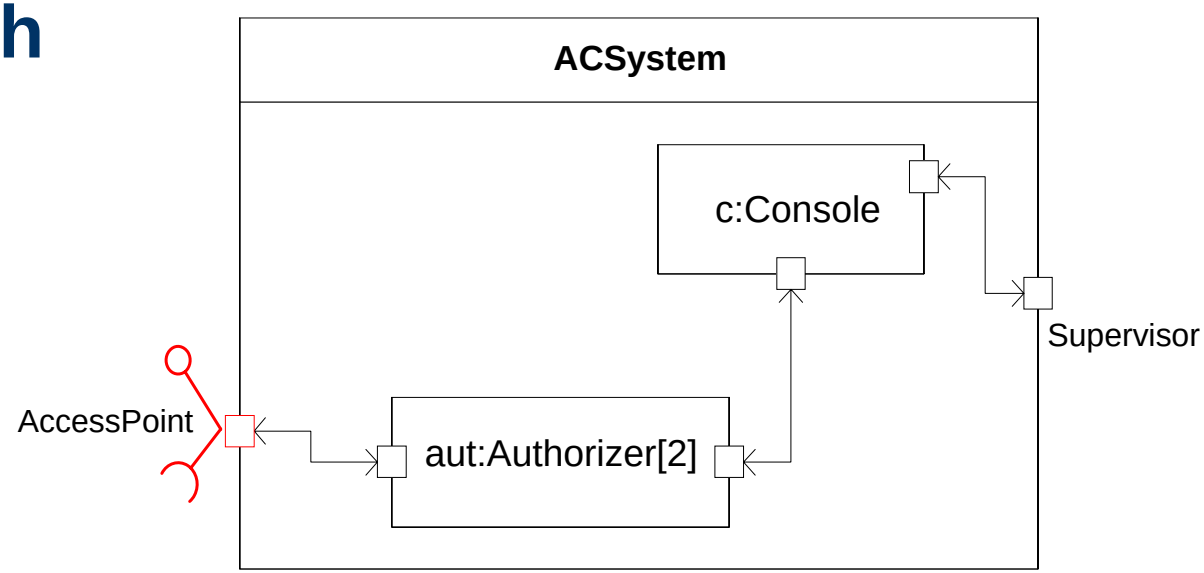
Variations

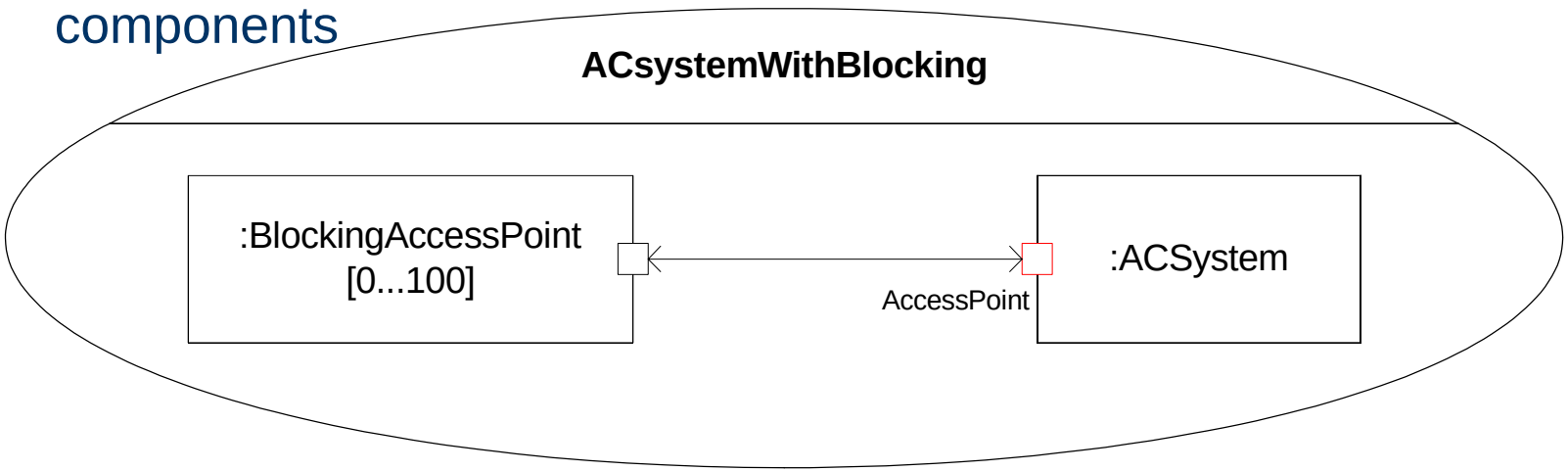


Acces

Plug-in approach

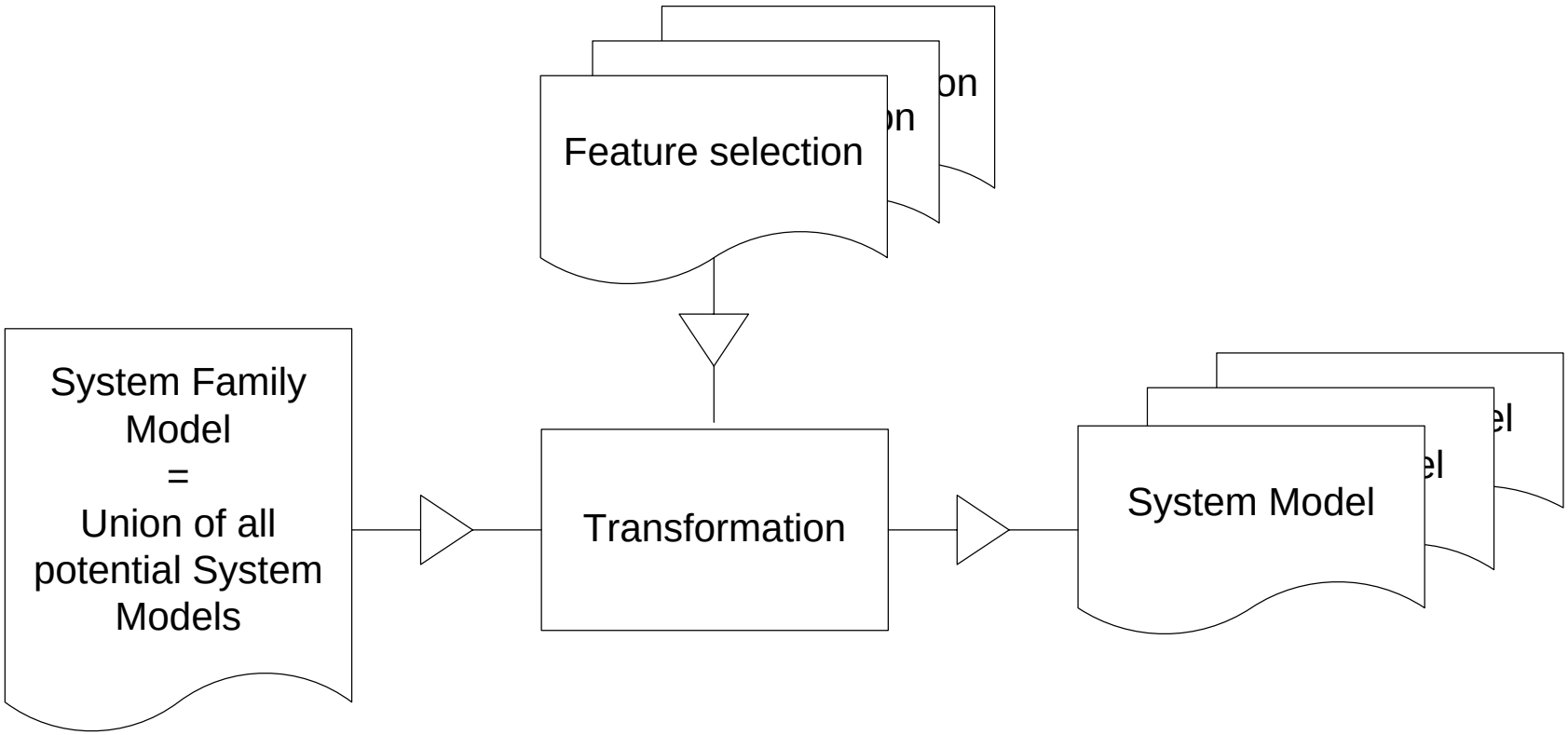
- Directly supported by ports and connectors with well-defined interfaces
- Important that this works with classes and not just for components





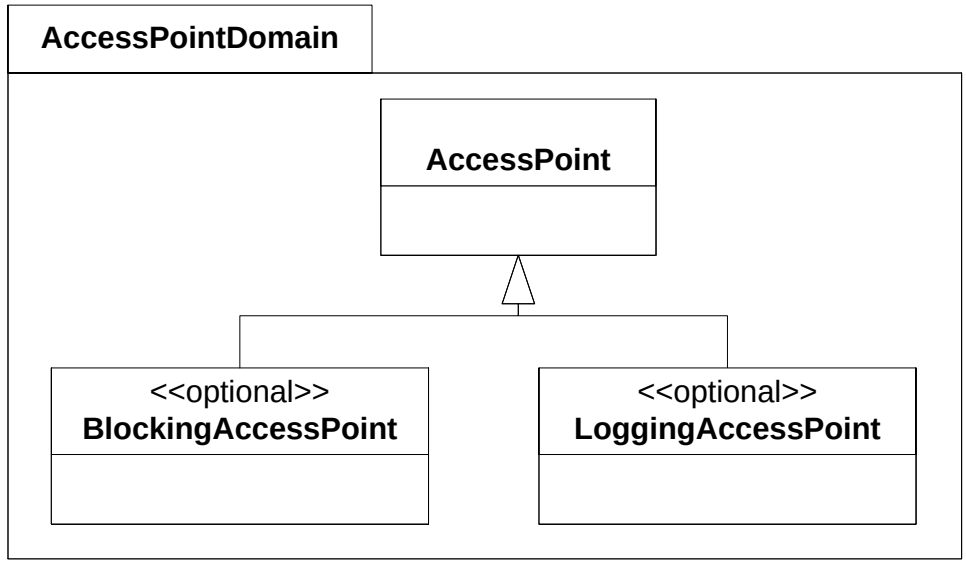
Feature models

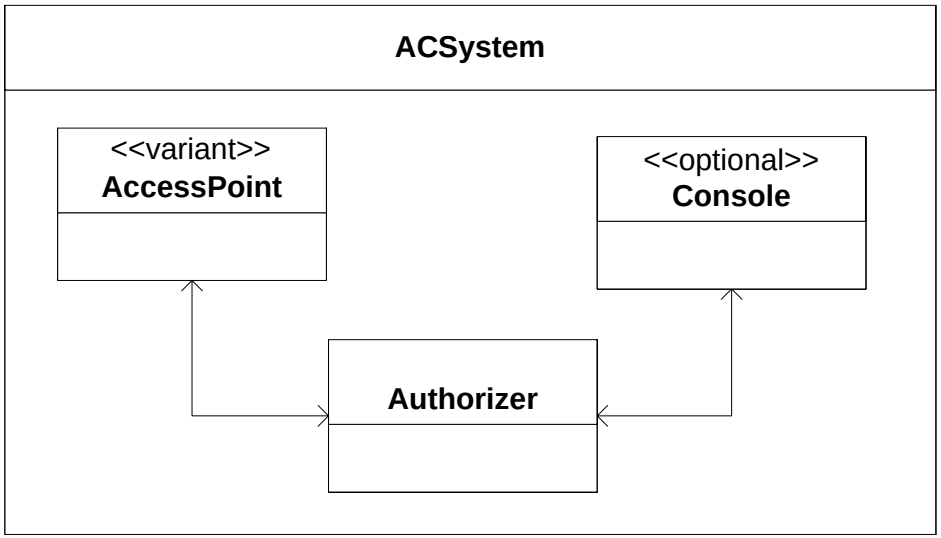
- One large system family model, with variations by stereotyping model elements
- If not modeling, this is called Generative Programming



Stereotyping classes

- Good news
  - UML2.0 allows more than one stereotype for each class
  - <<optional>> in addition to a real stereotype (e.g. <<embedded>>)

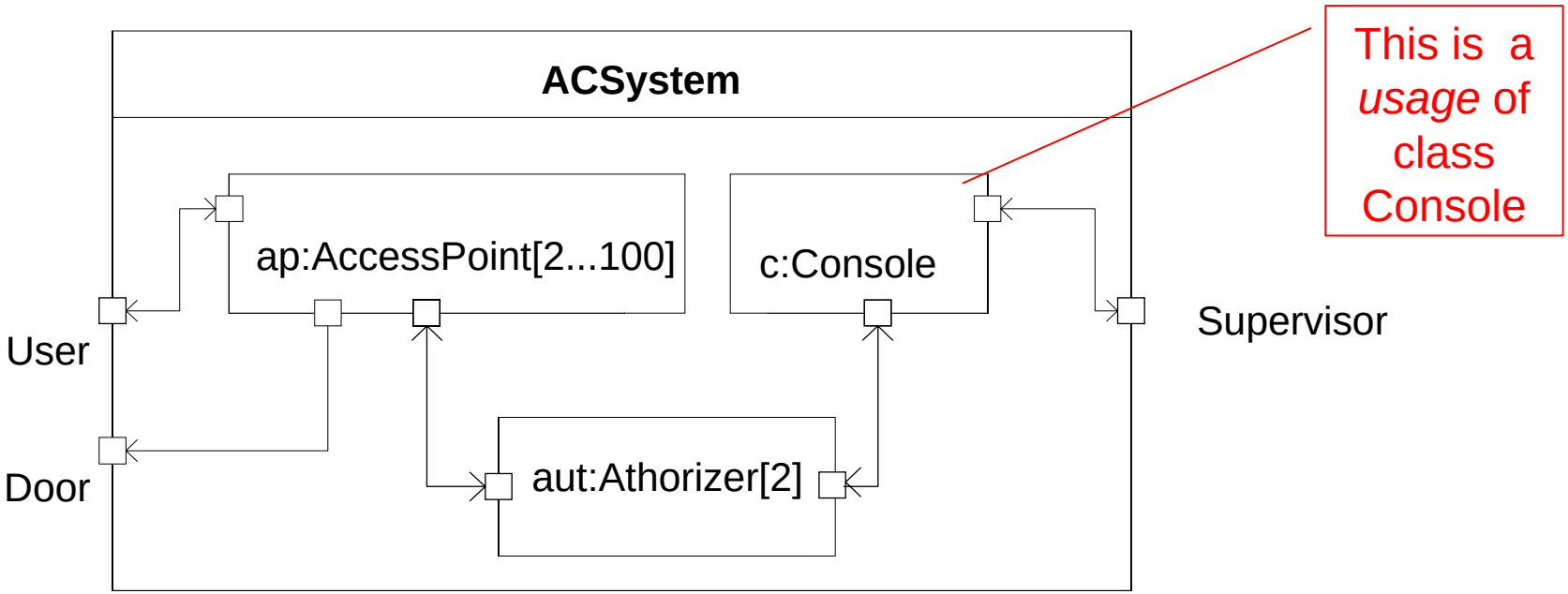




- Still does not help'
  - This tells which classes are **defined**, but it does *not* tell which classes from a given library are **used** in a given family or system

Even better news

- Usages of classes are highlighted in UML2.0

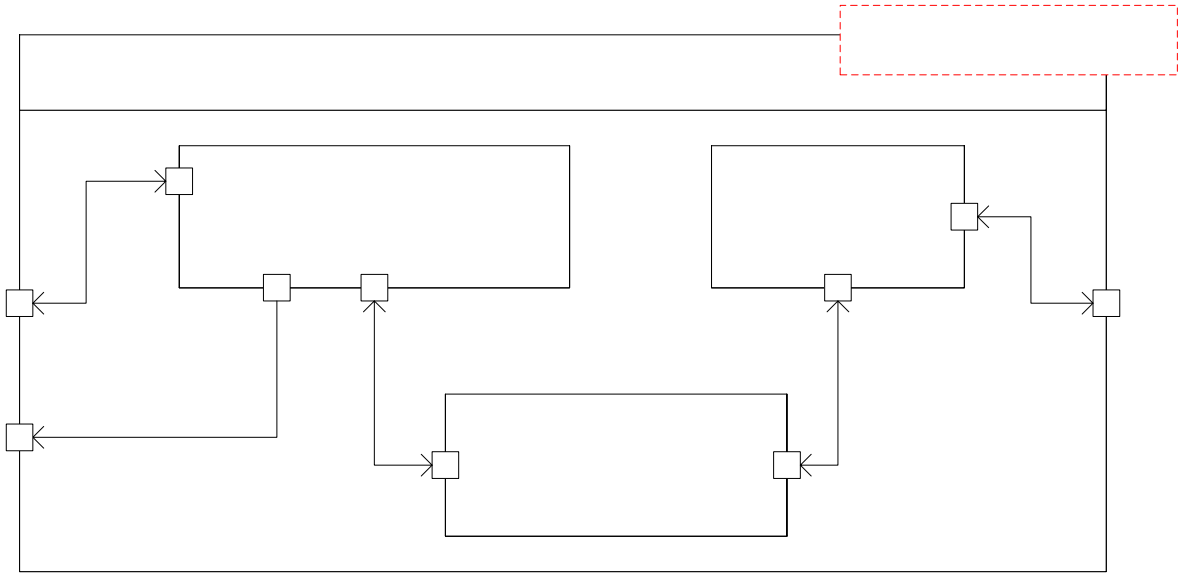


- ... and parts may of course be stereotyped

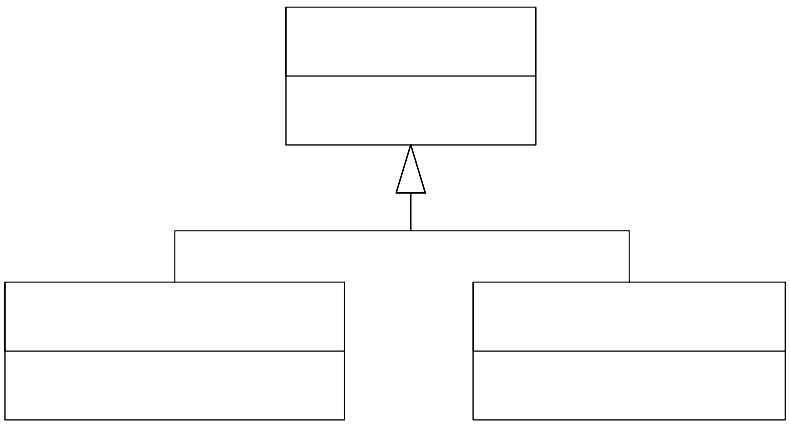
<<variant>>

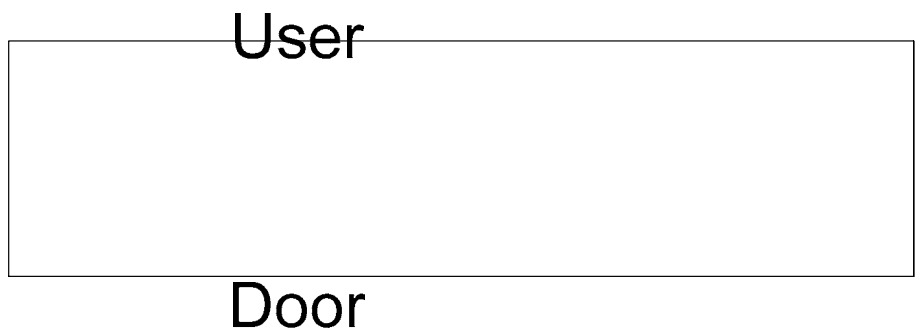
Templates

Variation by **constrained** template parameter: ACSystem knows that aType is atleast AccessPoint



ap: a





Yet another approach:

Product Lines

call for

Software Factories,

which in turn call for

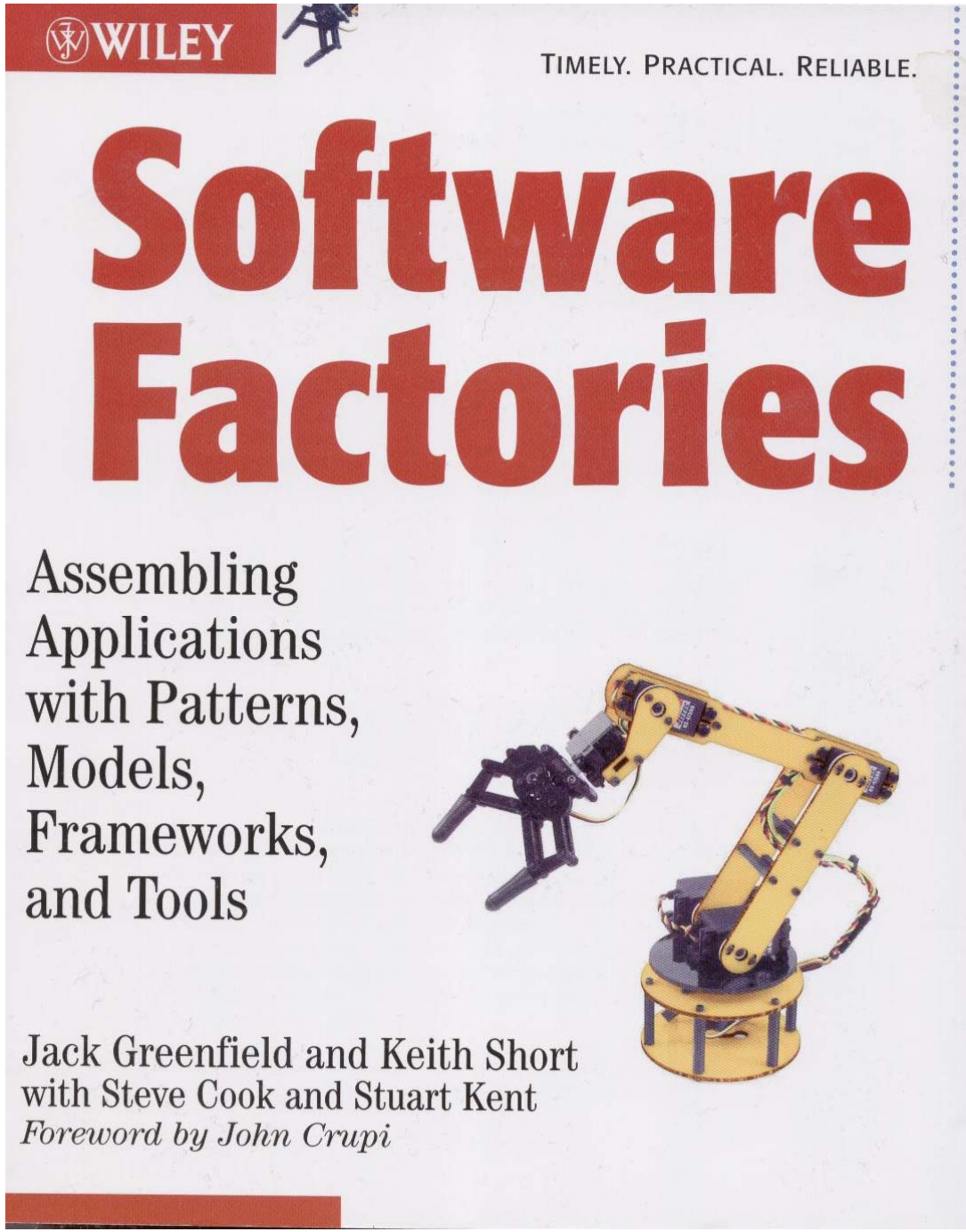
Domain Specific Languages (DSL):

Forget UML,

make one language for each domain,

for each product family

and/or for each platform

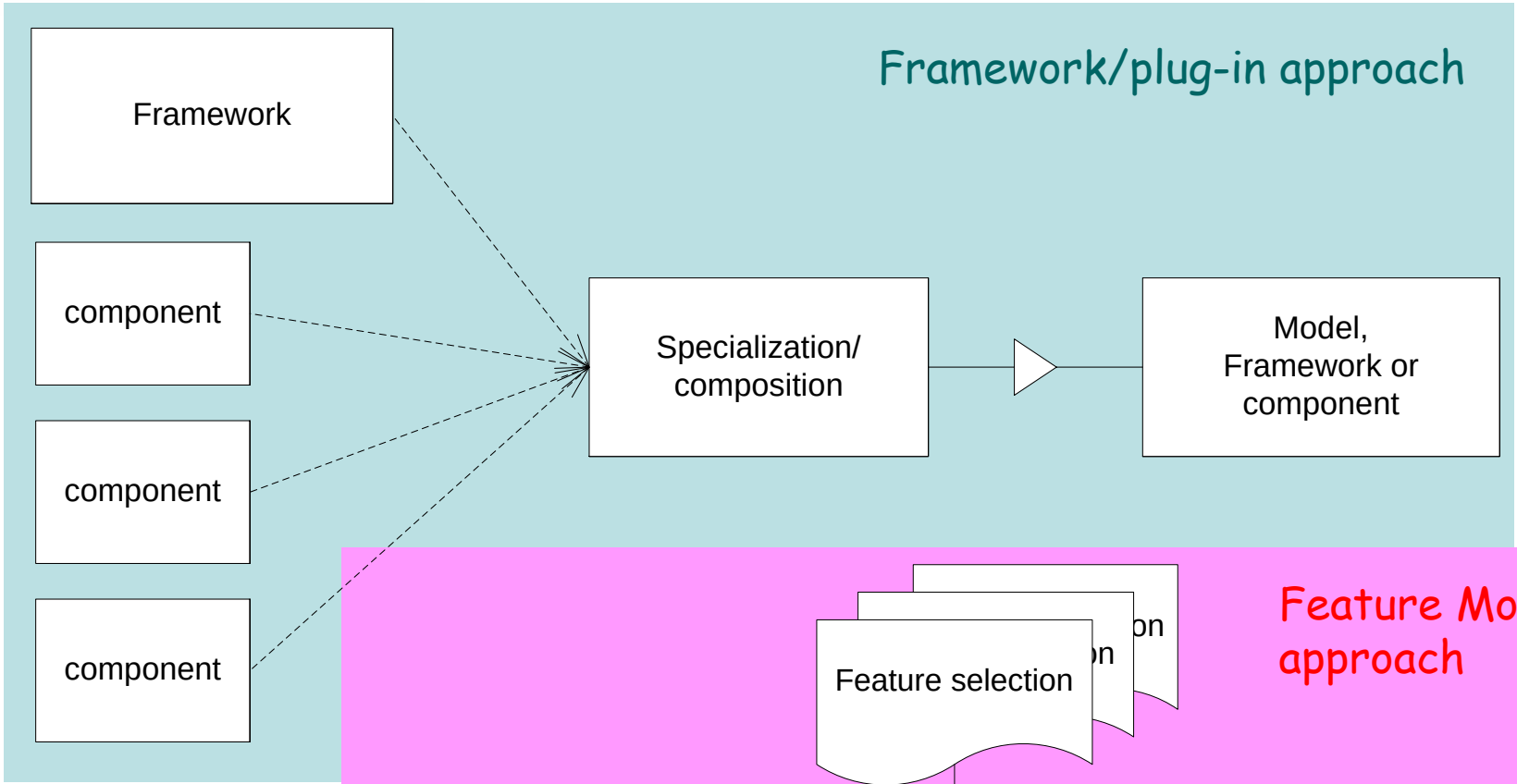


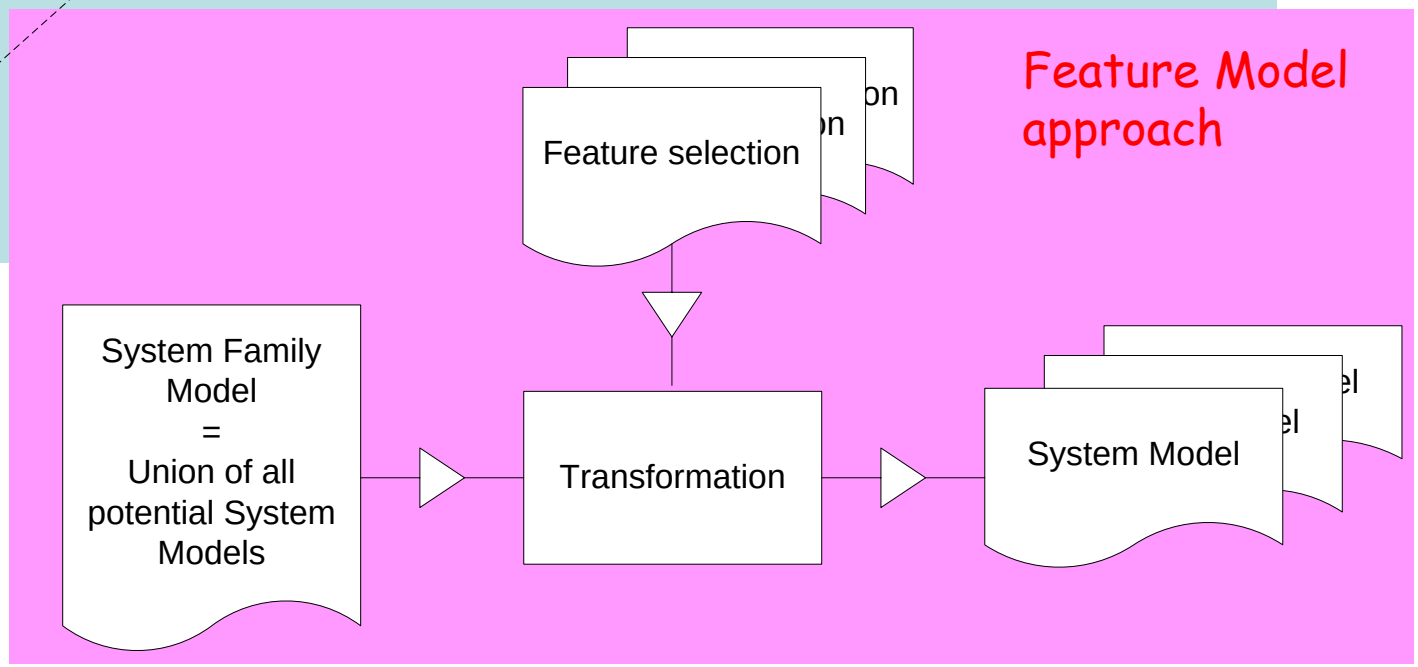
Recall this one?

- UML cannot be used because UML Components do not support .NET Component properties

Table 31 - Example Profile for .NET Components

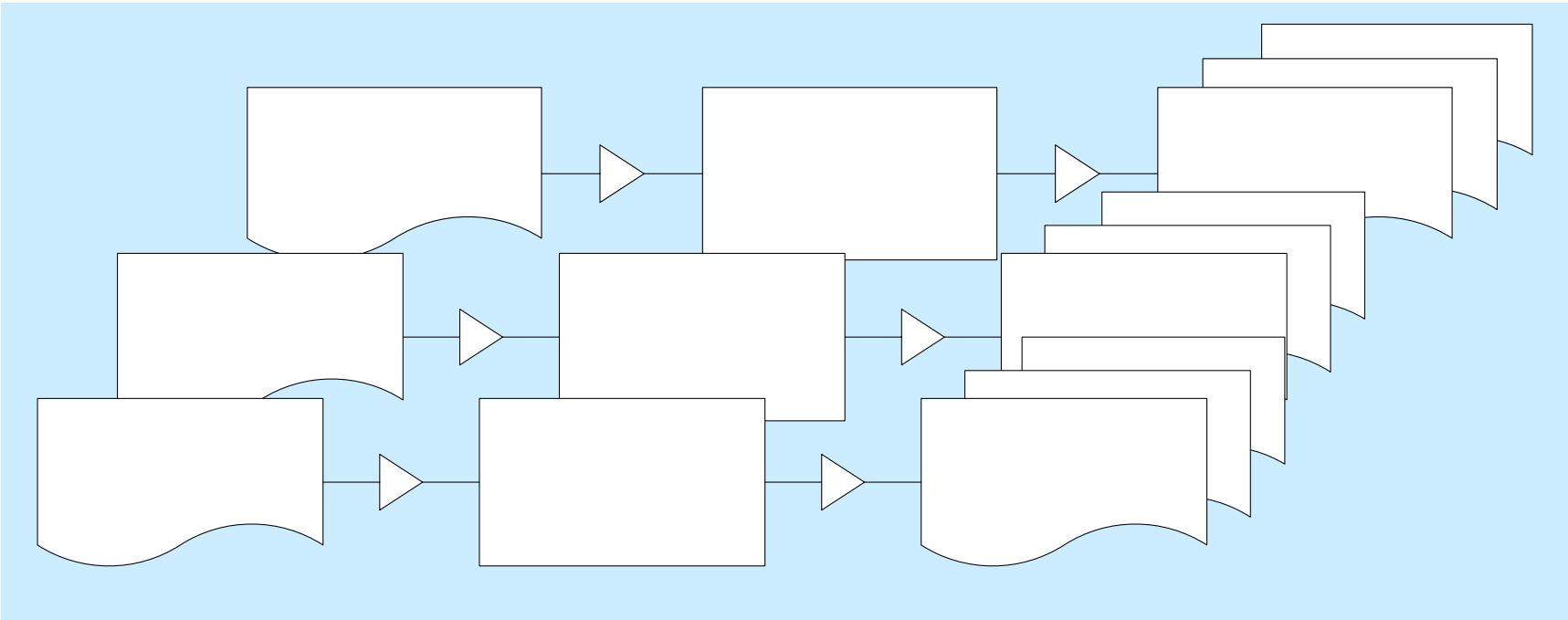
Stereotype	Base Class	Parent	Tags	Constraints
NetComponent «NetComponent»	Component	N/A	N/A	N/A
NETProperty «NETProperty»	Property	N/A	N/A	N/A
NETAssembly «NETAssembly»	Package	N/A	N/A	N/A
MSI «MSI»	Artifact	N/A	N/A	N/A
DLL «DLL»	Artifact	«file»	N/A	N/A
EXE «EXE»	Artifact	«file»	N/A	N/A


Framework/plug-in approach
Framework
component
component
component
Specialization/ composition
Model, Framework or component


Feature Model approach
Feature selection
System Family Model = Union of all potential System Models
Transformation
System Model

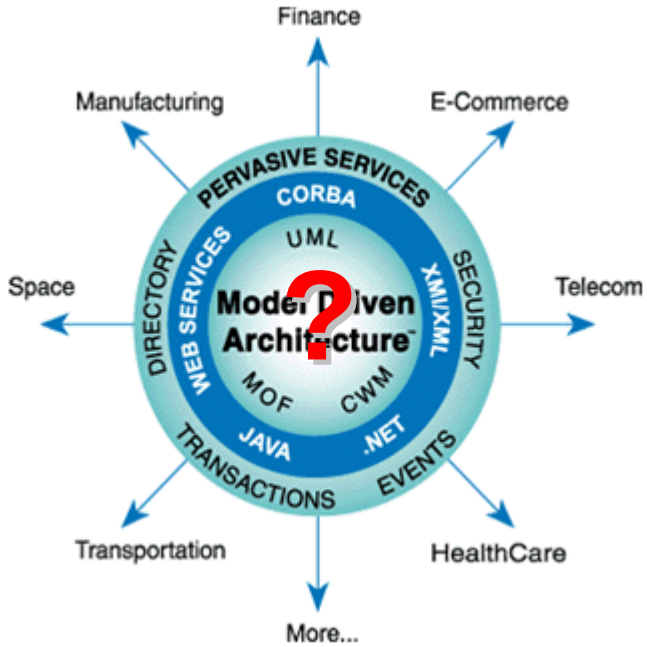
The Domain-Specific-Language Approach

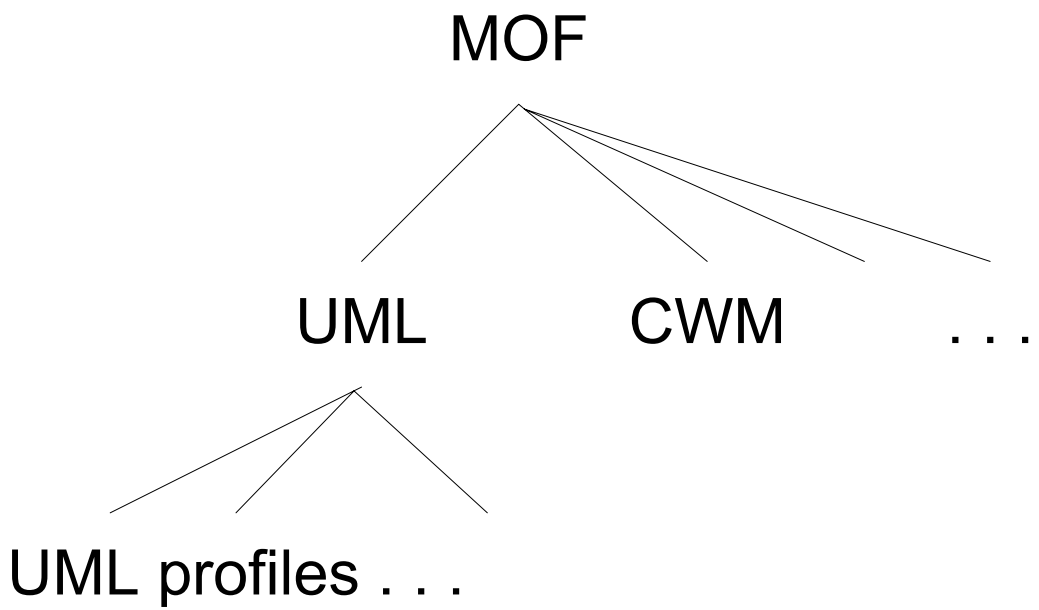
- no system family model
- new language for each family, many systems made in this language



What about MDA

- and Domain Specific Languages (DSL)?





SDL DSL

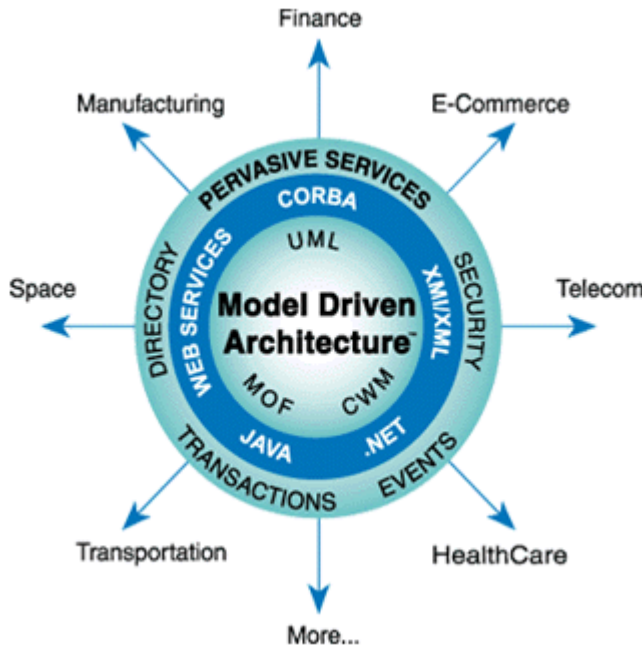
LSD LDS

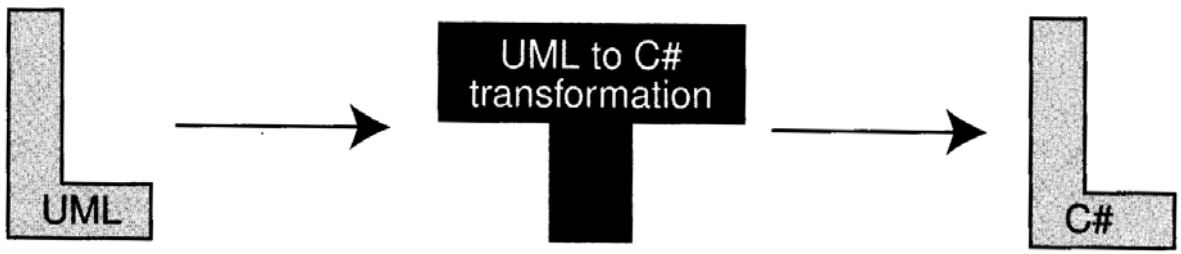
MDA –

- The dominant approach
  - ‘press-the-button’ approach
  - executable UML, code generation
  - programming at another level?
  - Requires profiling, closing the many semantic variation points of UML

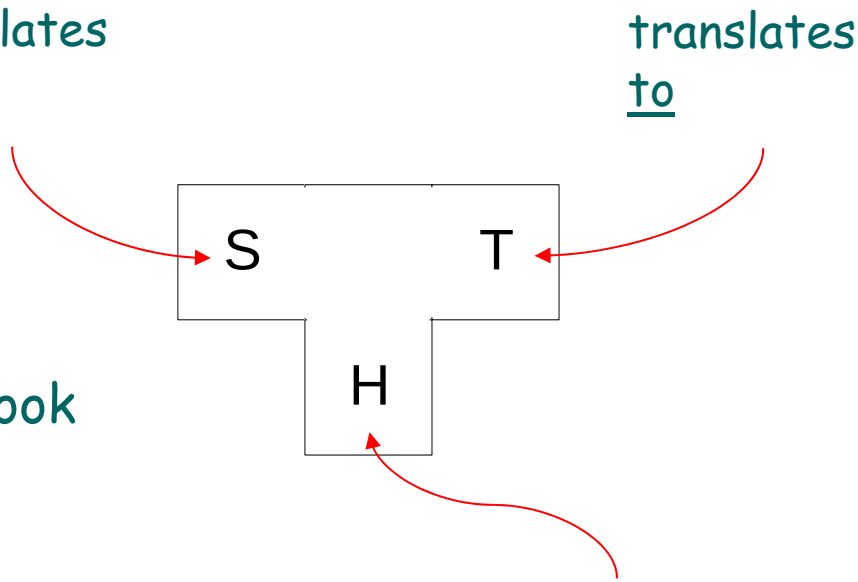
MDA: Model Driven Architecture

- From ”MDA explained”
  - MDA is about transformations
  - from PIMs (Platform Independent Models)
  - to PSMs (Platform Specific Models)

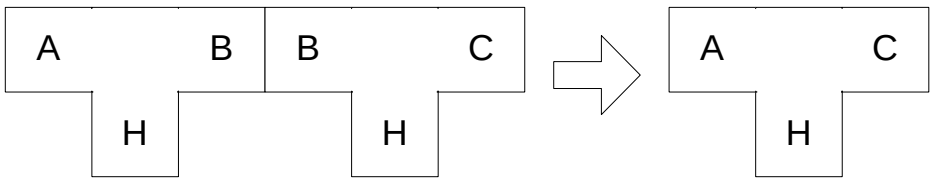


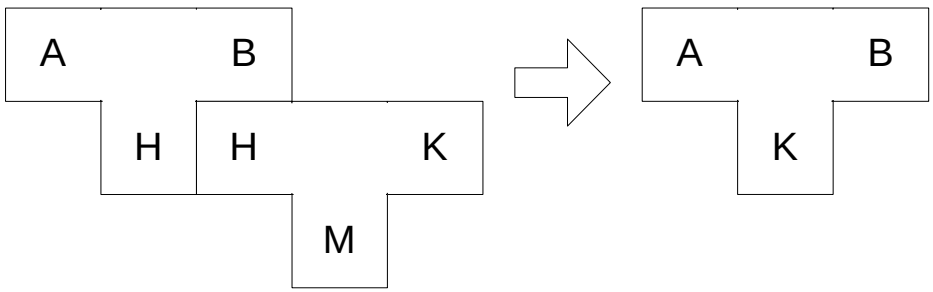


Looks familiar!?



From our compiler course textbook



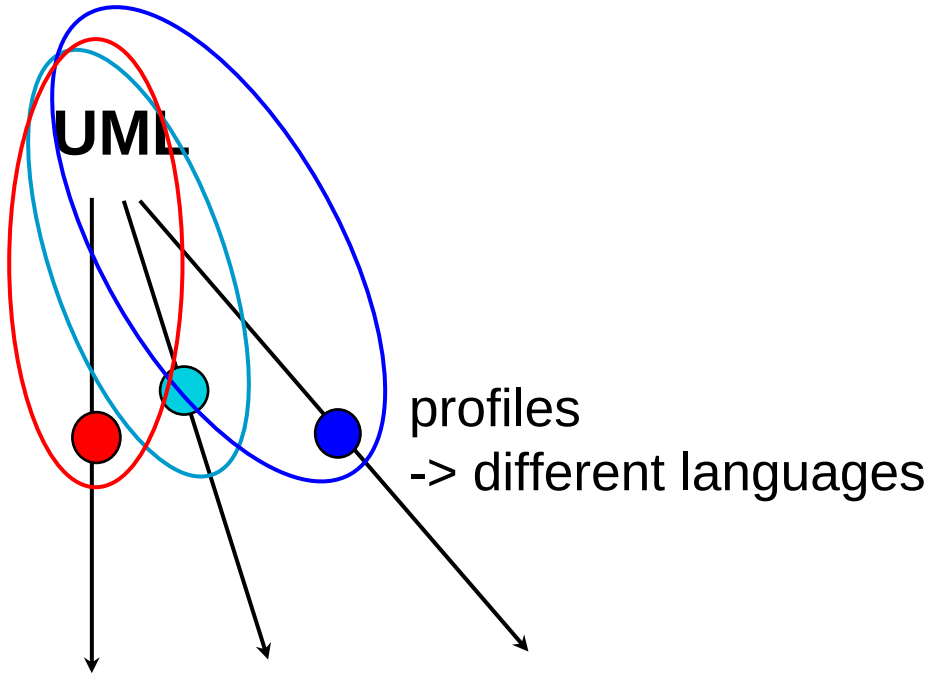


MDA -

- An alternative approach
  - the ‘multiple-models-are-better-than-one-model’ approach
  - Property models (e.g. interactions, collaborations, use cases)
  - Object models (class models, state machines)
  - Comparing/checking object models against property models

MDA by means of profiling?

- Domain, company, department, project standard?



Domain Specific Languages <> UML profiles

- Domain Specific Languages
  - Gives the exact right language concepts
  - Often defined from scratch, have to be maintanied
  - Multi-domain models will have multiple DSLs, commonality between languages not solved
- UML profiling
  - Exploits the common (but sparse) semantics of UML
  - A model an use several profiles, in principle
  - Profiles tend to tweak UML, and in fact make new metamodels
  - Models with stereotypes often unreadable
- Developers will anyhow require real tool support comparable to tools they have for programming languages