MDE and System Modeling

Topics covered

- ✧ MDE
- ✧ Models & Metamodels
- ✧ Types of models
- ✧ Transformations

System modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Existing and planned system models

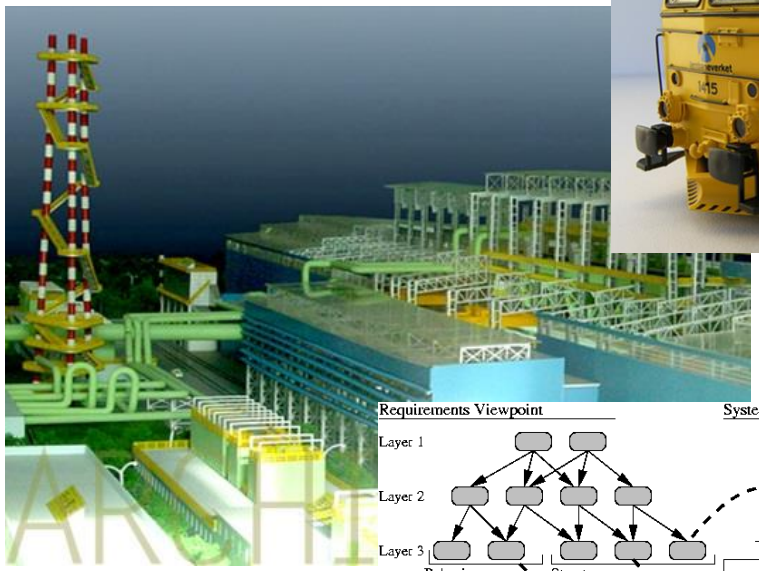
- Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

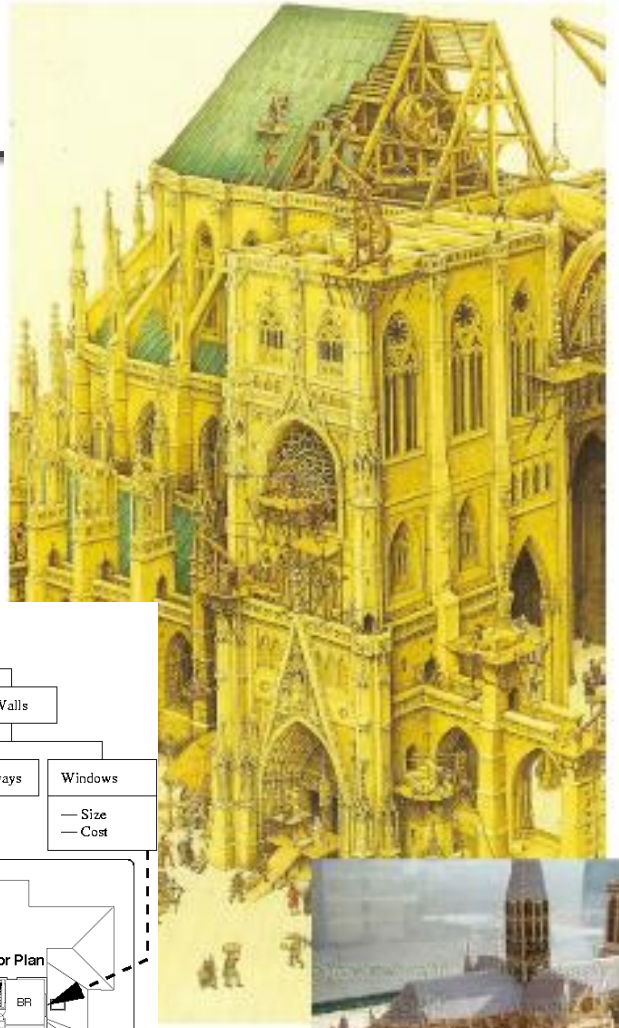
Model-driven engineering

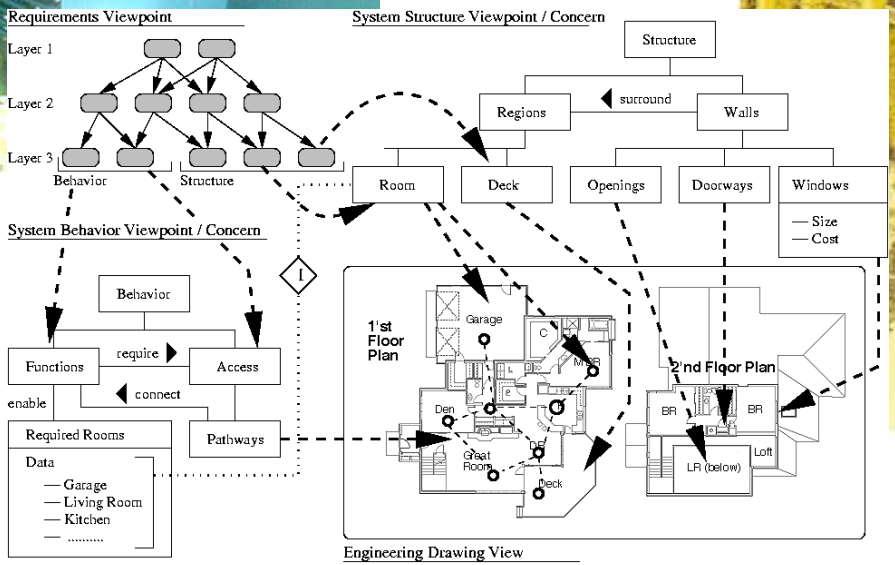
MODELS

http://www.flickr.com/photos/navarzo4/6450276881

http://www.archimod.com







http://www.isr.umd.edu/~austin/paladinRM/future-work.gif

Bran Selić, SDL Forum 2013 Keynote:
”Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?””

Why Engineers Build Models

- Engineering model: a selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns
- We don't see everything at once; what we do see is adjusted to human needs and understanding
- Reducing complexity to the human scale

abstraction

simplified representation

reasoning

http://commons.wikimedia.org

Bran Selić, SDL Forum 2013 Keynote:
"Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?"

Purpose of Engineering Models

- Descriptive models:
  - To help us understand (i.e., reason about) complex systems
  - To communicate understanding and design intent to others
  - To predict the interesting characteristics of systems
- Prescriptive models:
  - To specify what systems must do

Bran Selić, SDL Forum 2013 Keynote:
"Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?"

Modeling vs. Programming Languages

- The primary purpose and focus of programming languages is implementation:
  - The ultimate form of prescription
  - Implementation requires total precision and “full” detail
  - Prescription takes precedence over description
- To be useful for humans, a modeling language must support description as a first-order concern:
  - I.e., communication, prediction, and understanding
  - These generally require omission of “irrelevant” detail such as details of the underlying computing technology used to implement the software

Bran Selić, SDL Forum 2013 Keynote:
”Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?”

Desired Characteristics of Models

- Abstraction: emphasize important aspects and ignore the irrelevant ones
- Understandability: easy to understand by users
- Accuracy: correctly represent the modeled system for intended purpose
- Prediction: use them to answer questions about the modeled system to detect errors and omissions and determine the most important tradeoffs in complex designs before committing resources for their realization
- Low cost: cheaper to build and study

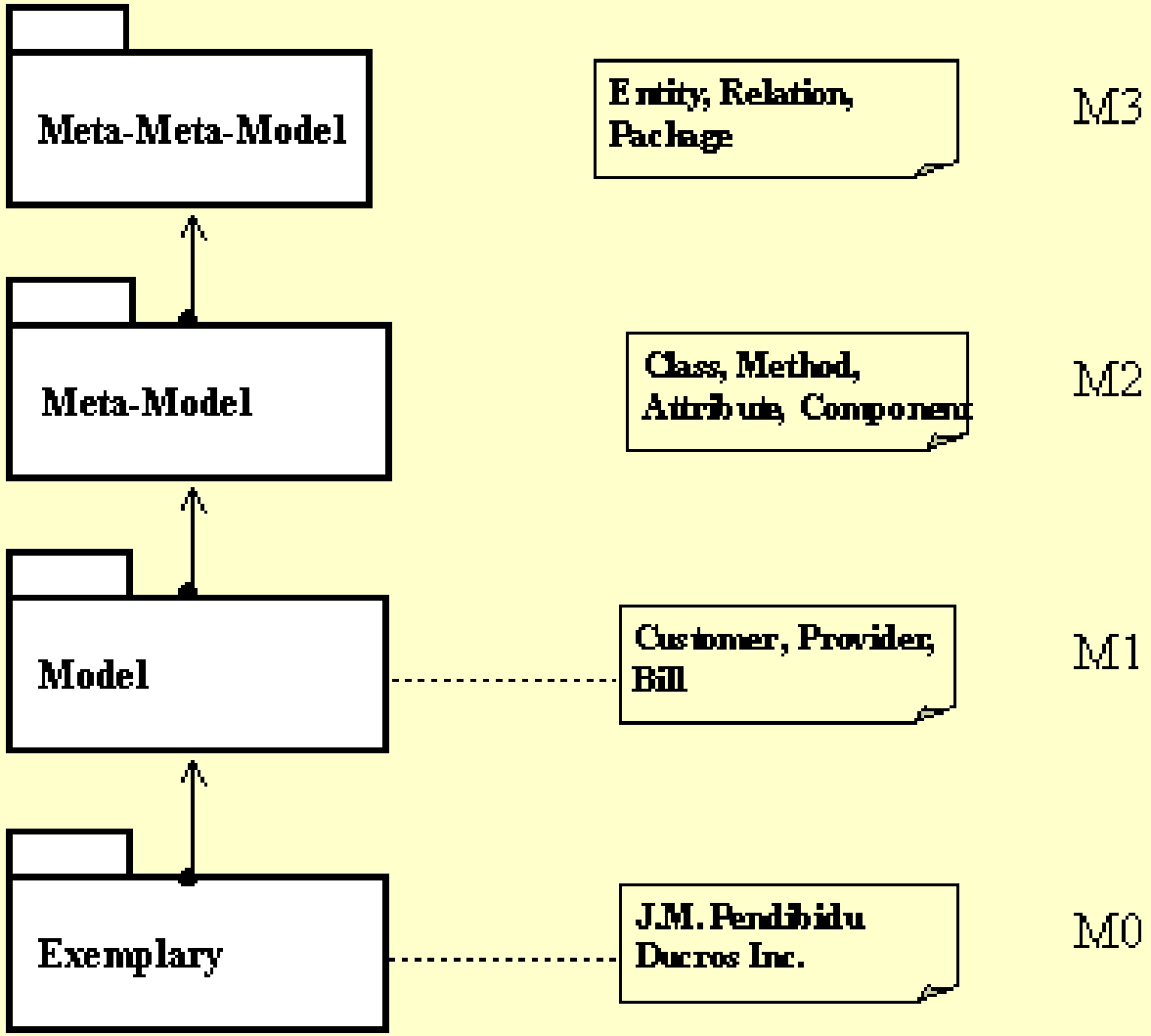
Usefulness of Models

- Understanding the problem (or reality)
- Communication among stakeholders
  - Customers, users, developers…
- Controlling complexity
  - Abstraction
  - Through analysis (formal)
  - Investigate and compare alternative solutions
  - Minimizing risks
- Developing (software) systems
  - Guide implementation
  - Facilitate evolution

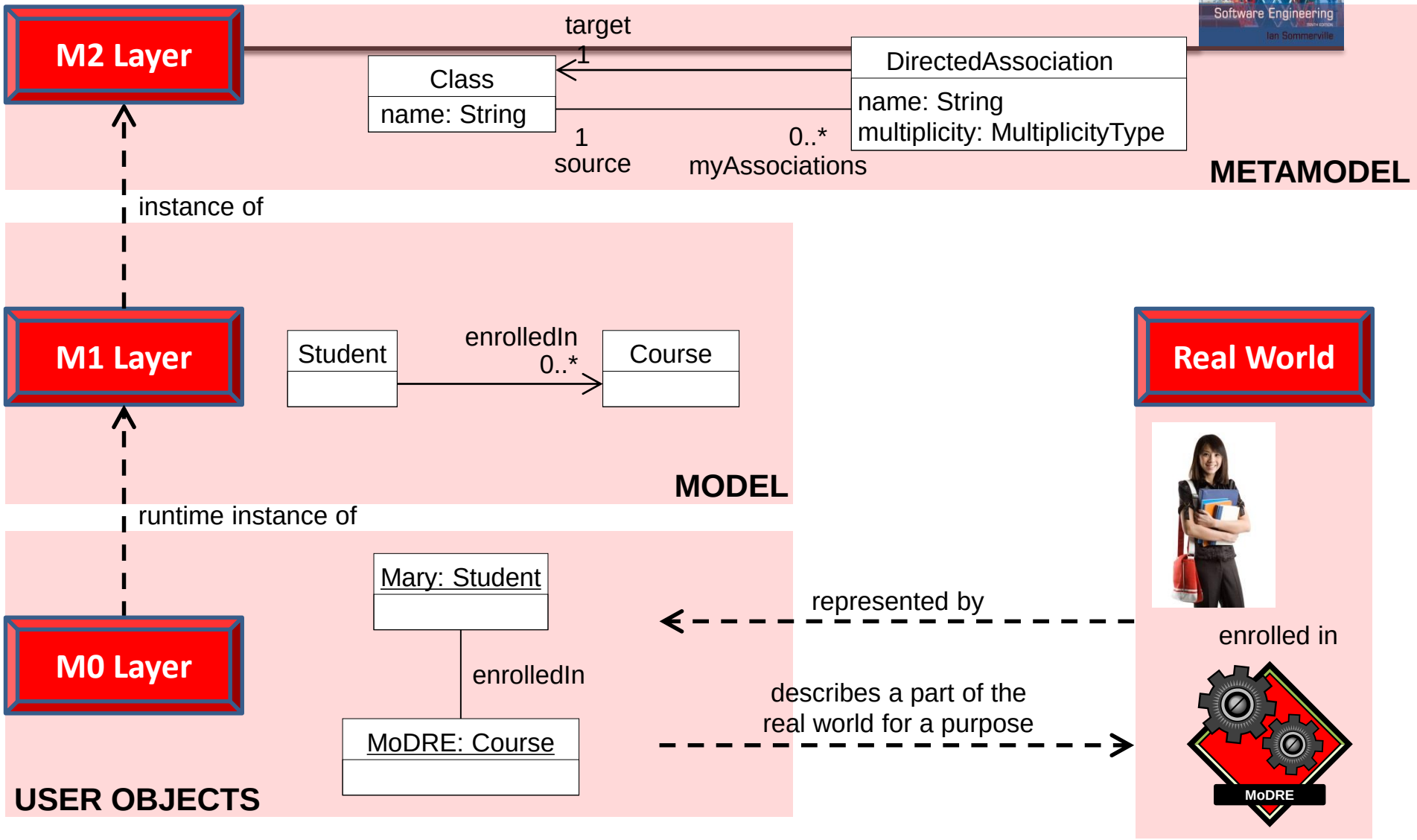
Model Driven Engineering

- With MDE models are no longer simple mediums for describing systems or only facilitating inter-team communication…
- MDE proposes the systematic use of models (specified through metamodels) as first-class software artifacts and their subsequent transformations throughout its life cycle.
- A software system is obtained through the definition of different models at different abstraction layers.
- MDE increases the level of abstraction and automates the development process
  - This provides faster and more reliable results.

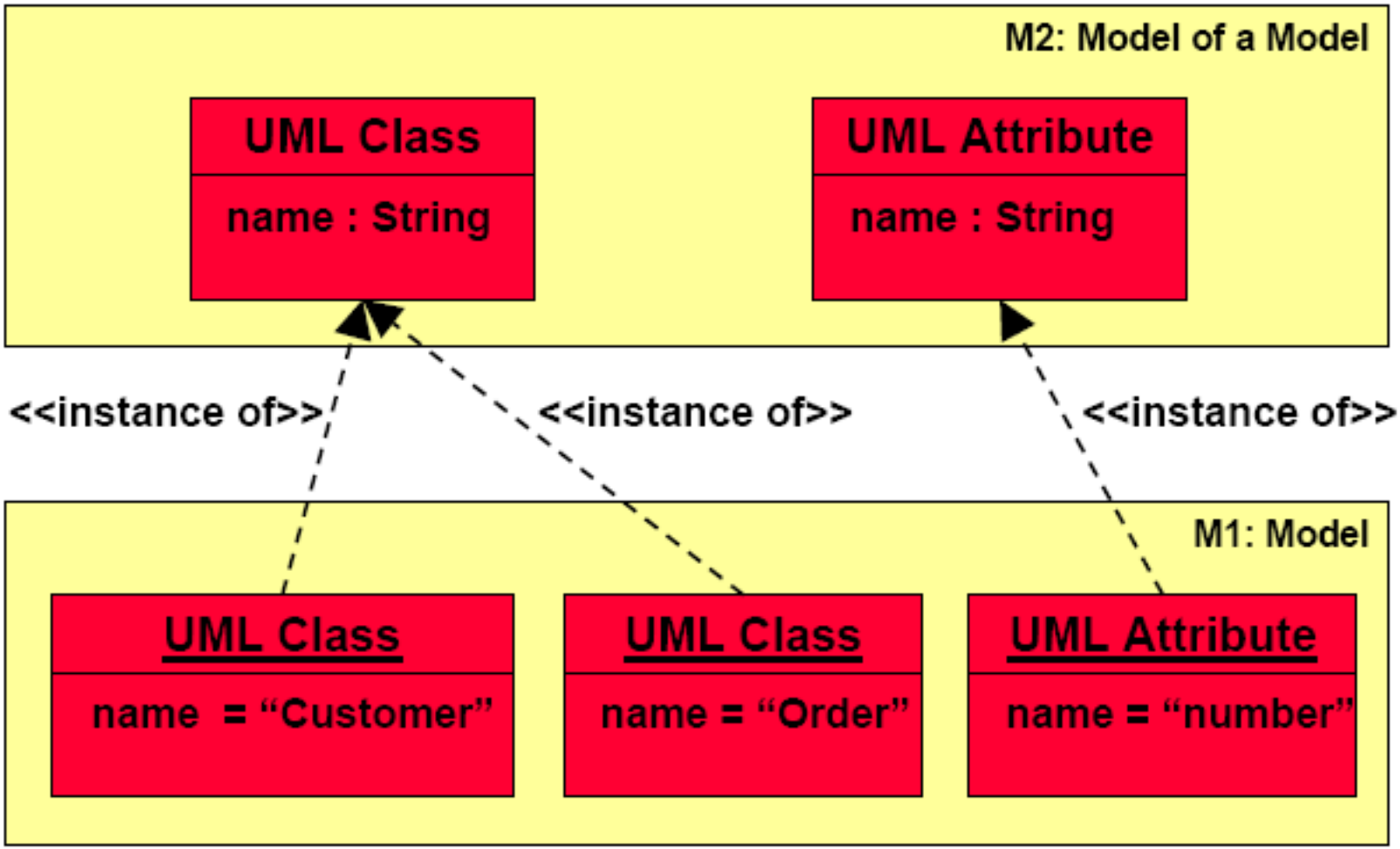
Four-Level Metamodel Hierarchy



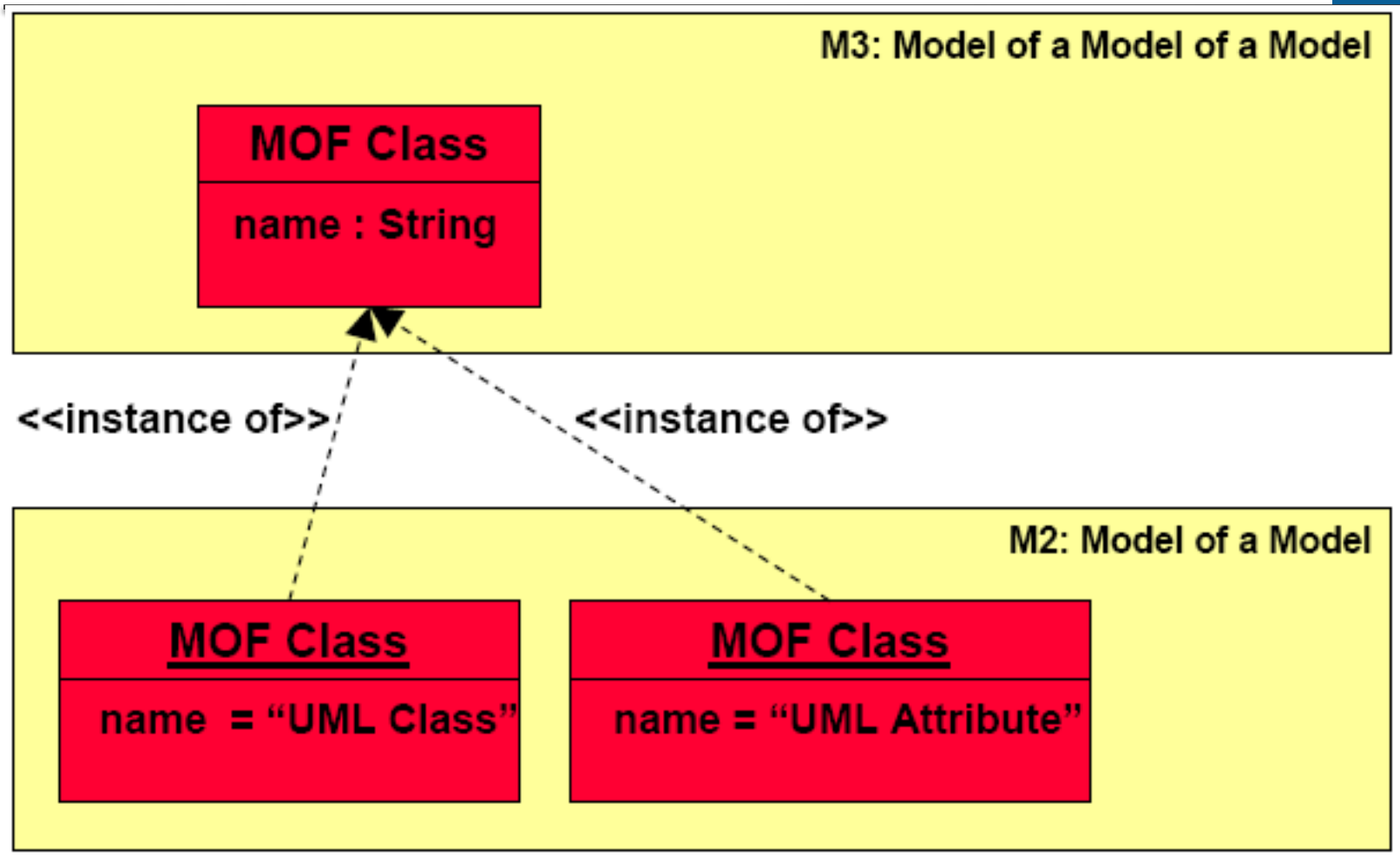
Layered Metamodel Hierarchy



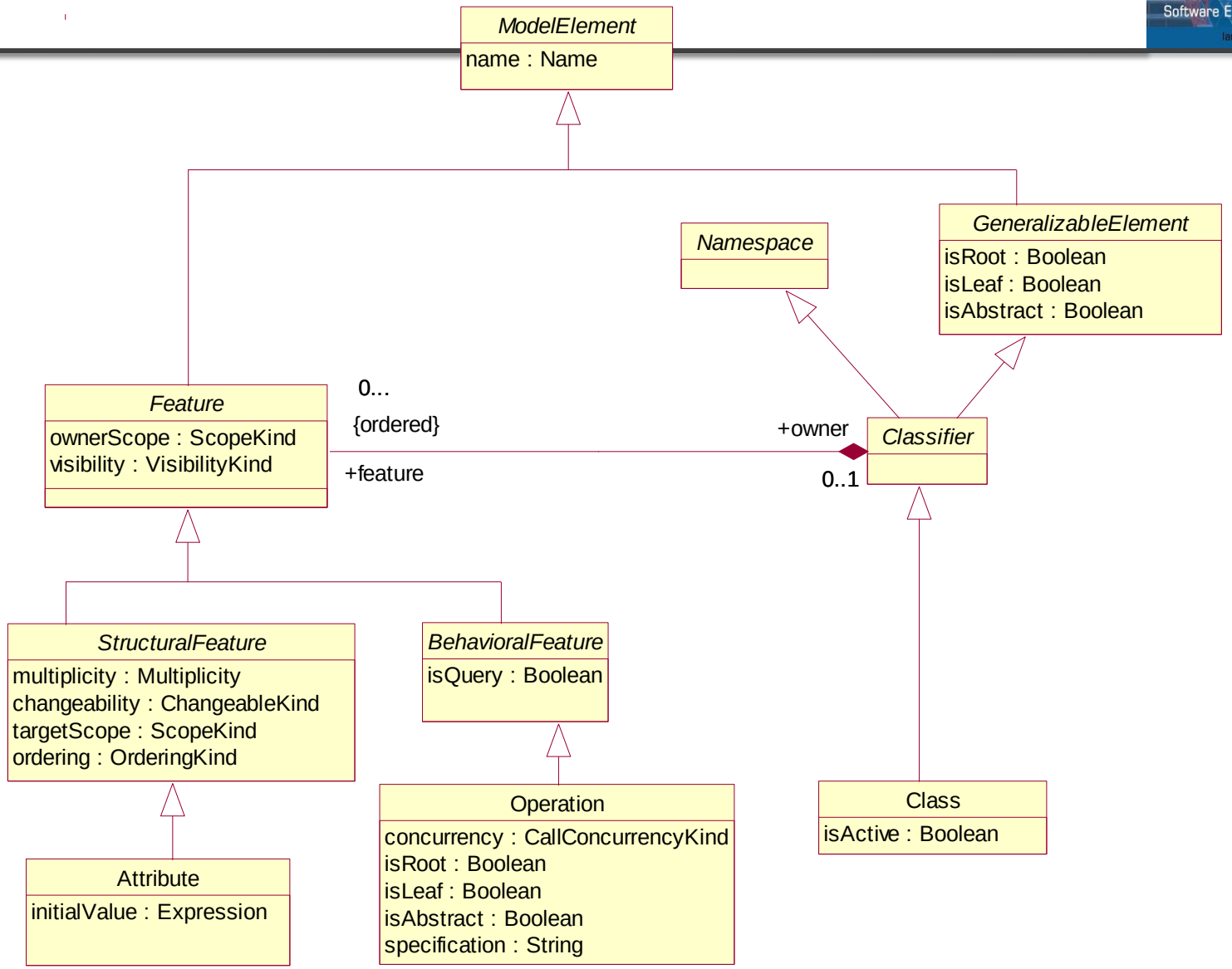
Model & metamodel (from Kleppe et MDA explained)



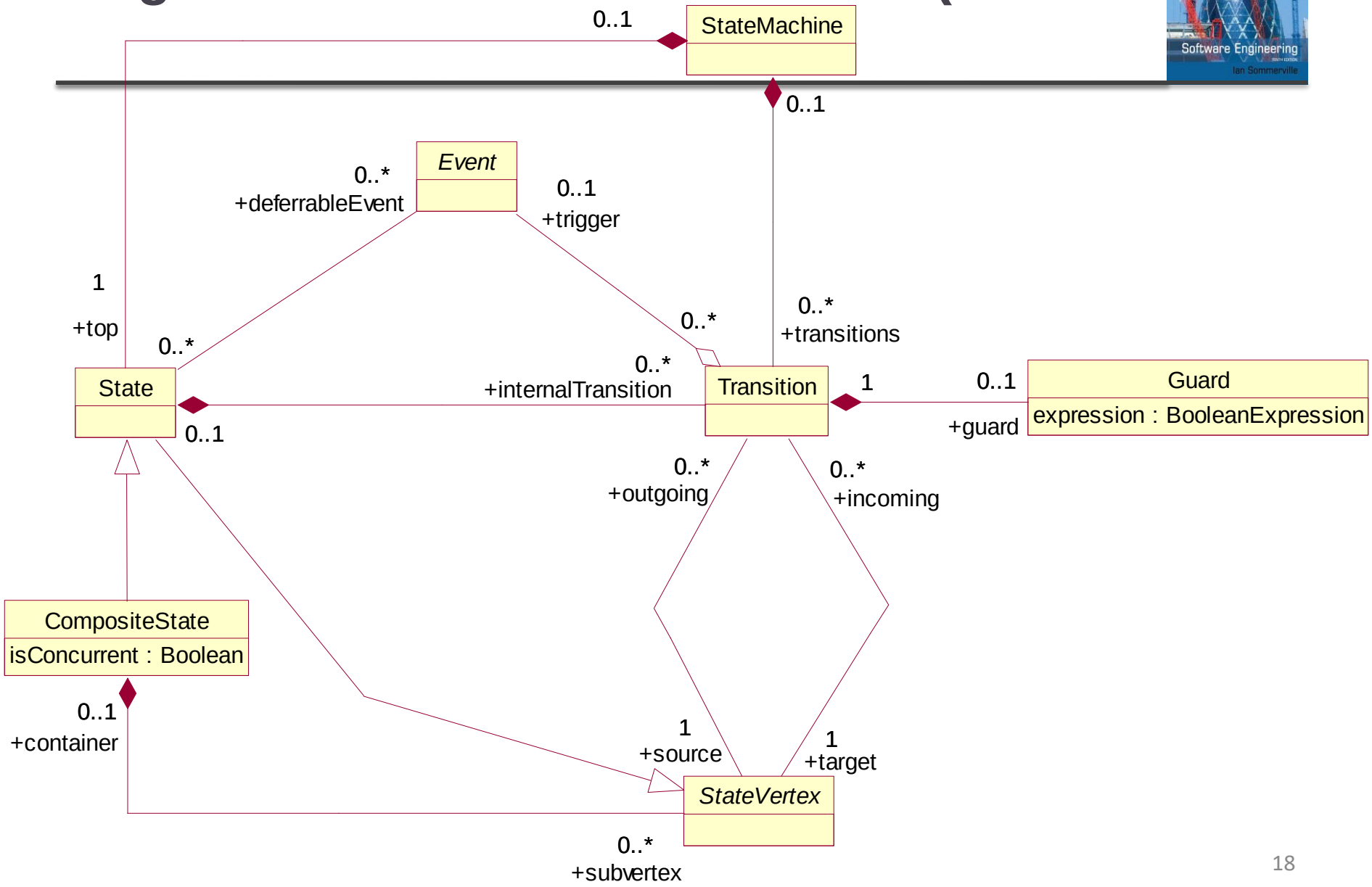
Metamodel & Meta-metamodel (from Kleppe et MDA explained)



Fragment of UML Metamodel (Classes)



Fragment of the UML Metamodel (Statecharts)

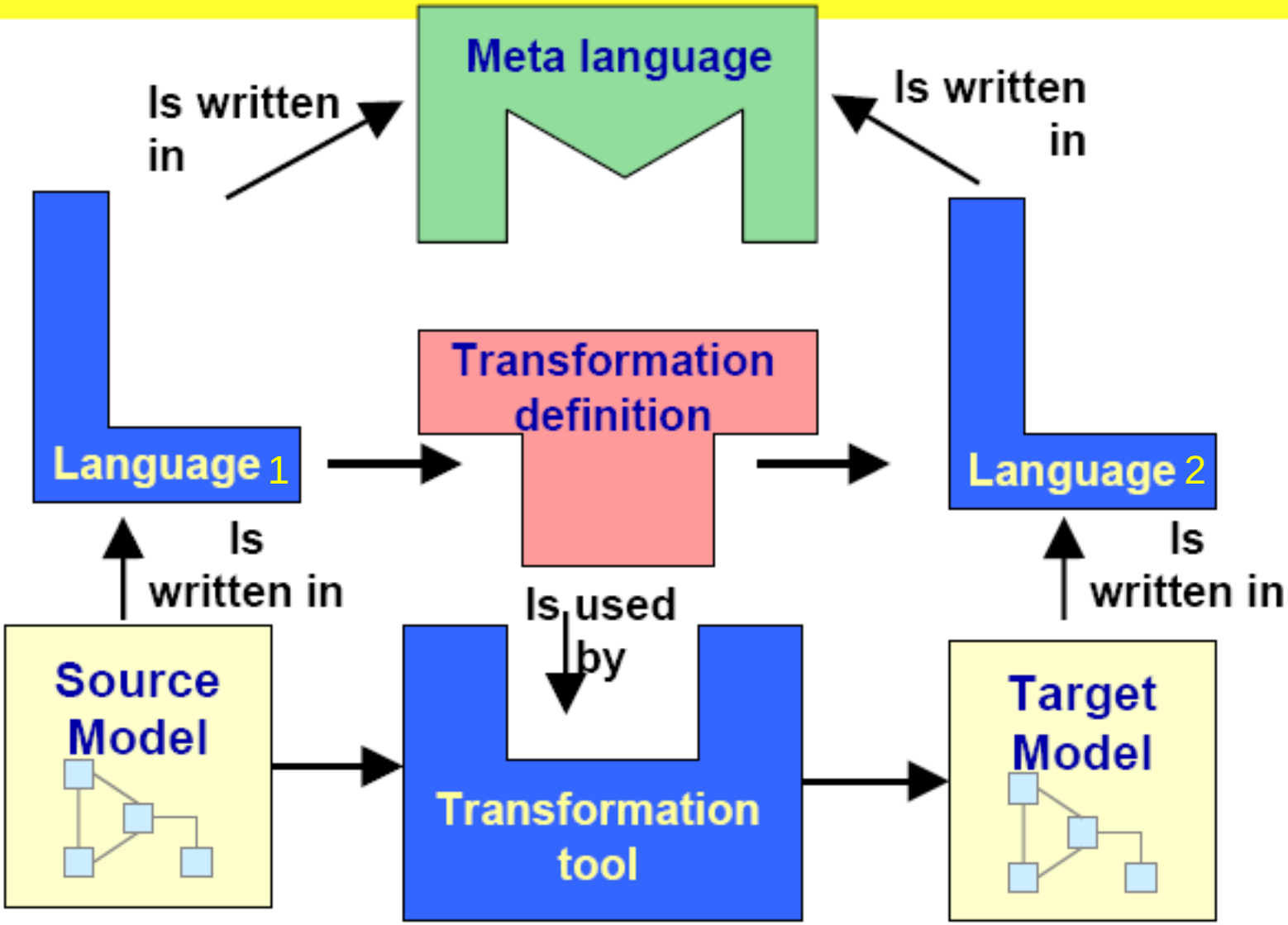


Model-driven engineering

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

MDA Framework (from Kleppe et MDA explained)

MDA Framework (from Kleppe et MDA explained)



MDE + RE!

- ✧ RE also benefits from MDE
- ✧ MDE may be used to:
  - ensure consistency between different kinds of requirements analysis models (e.g., goal, scenario or domain models)
  - to automatically build architectural models from requirements
  - to increase separation of concerns and their composability
  - to help evaluating models' quality attributes.

Usage of model-driven engineering

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- Pros
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- Cons
  - Models for abstraction and not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

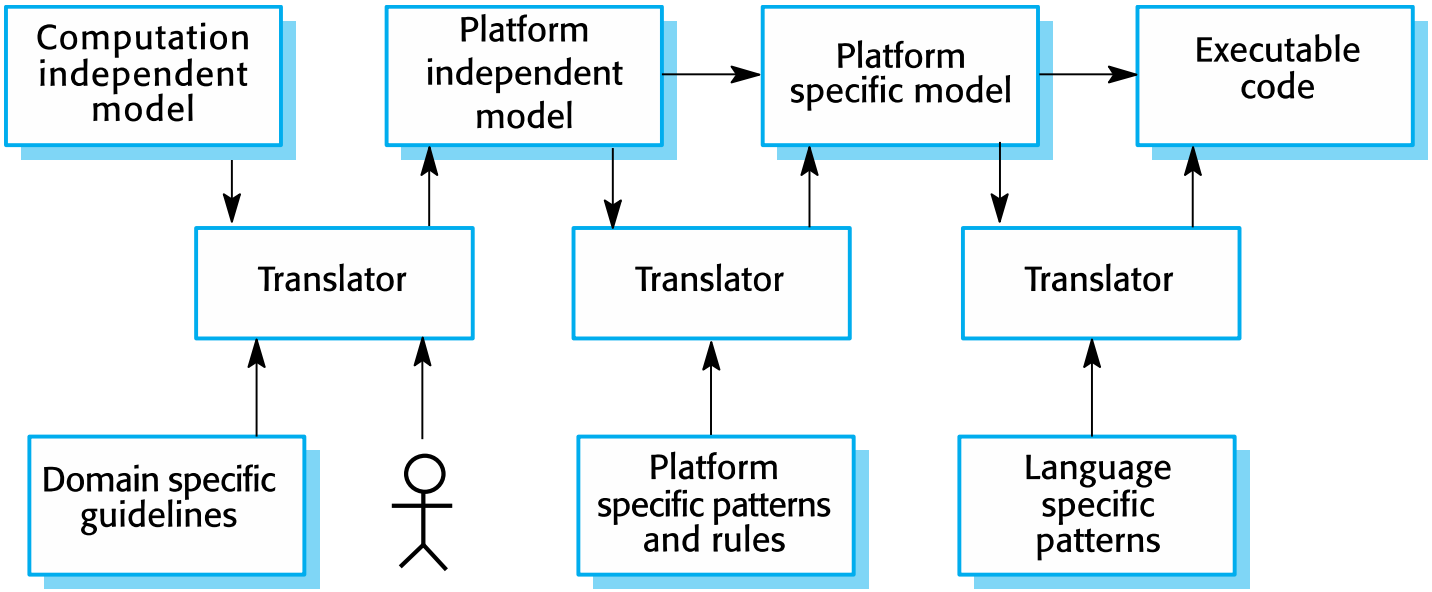
Model driven architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering
- MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

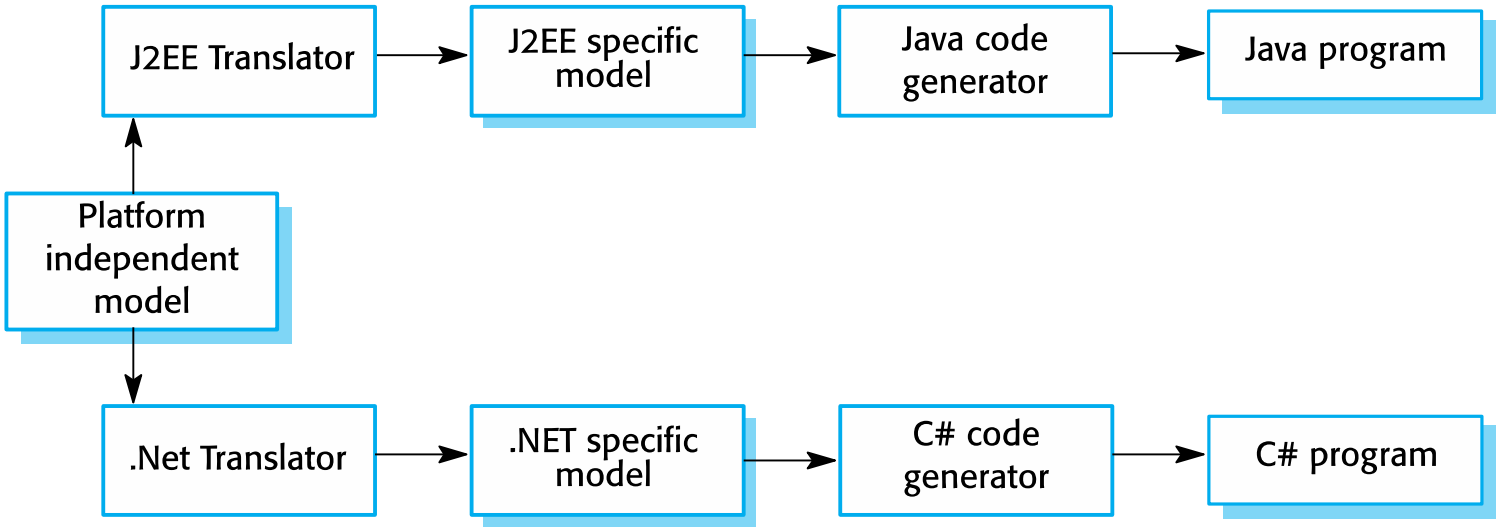
Types of model

- A computation independent model (CIM)
  - This models the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- Platform specific models (PSM)
  - These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

MDA transformations



Multiple platform-specific models



Model Transformations: the case of ATL

✧ A language to describe families and other for persons

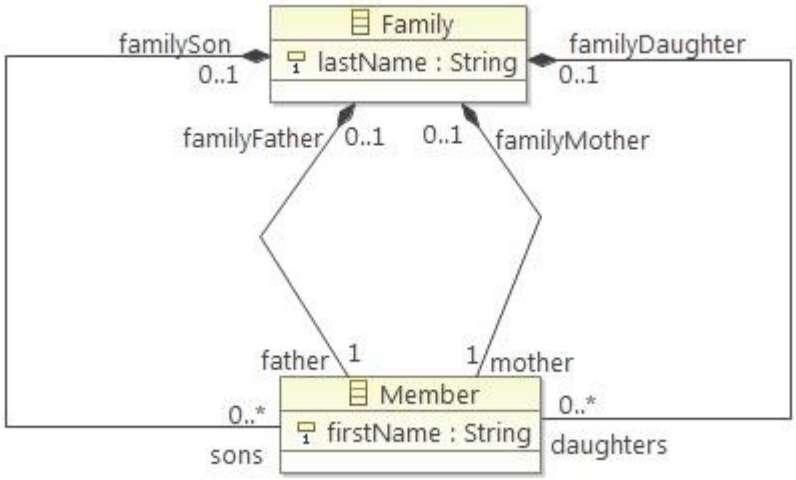
Transforming this …

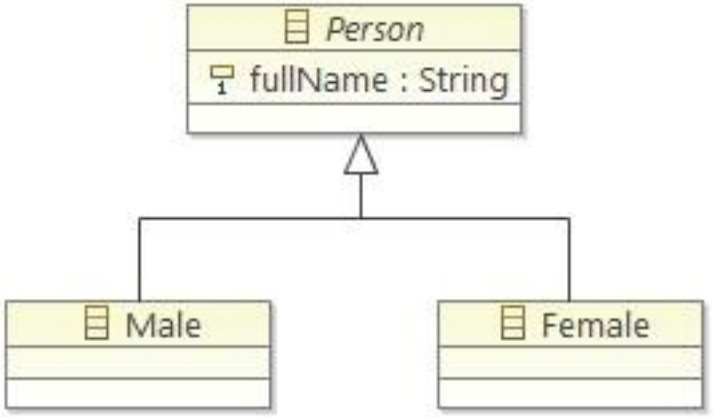
…
Family March
Father: Jim
Mother: Cindy
Son: Brandon
Daughter: Brenda
… other Families

… into this.

…
Mr. Jim March
Mrs. Cindy March
Mr. Brandon March
Mrs. Brenda March
… other Persons

Metamodels






```
helper context Families!Member def: isFemale(): Boolean =
        if not self.familyMother.oclIsUndefined() then
                true
        else
                if not self.familyDaughter.oclIsUndefined() then
                        true
                else
                        false
                endif
        endif
```

Regras de transformação

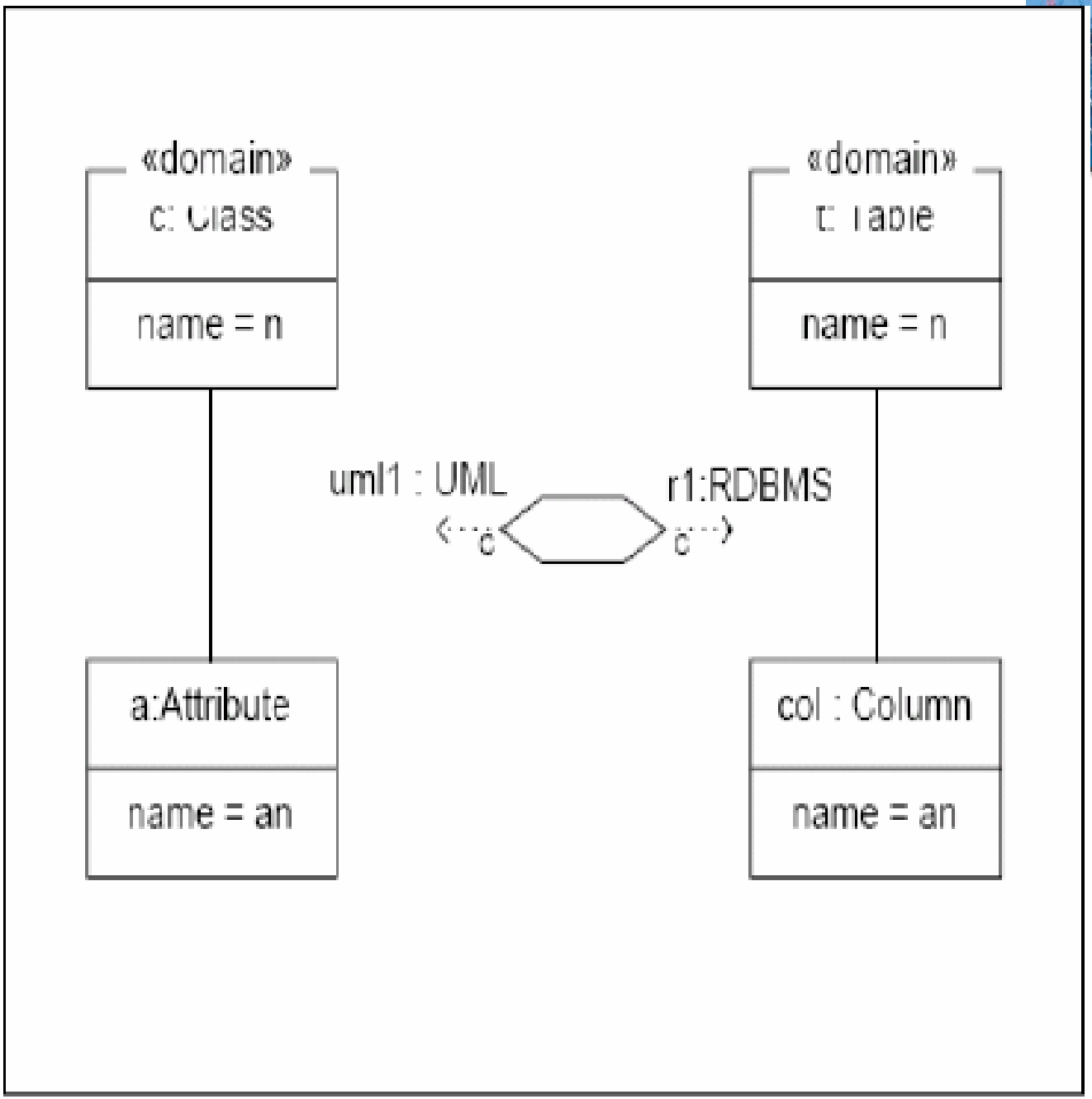
- Member to Male

```
rule Member2Male {
        from
                s : Families!Member (not s.isFemale())
        to
                t : Persons!Male (
                        fullName <- s.firstName + ' ' + s.familyName
                )
```

- Member to Female

```
rule Member2Female {
        from
                s : Families!Member (s.isFemale())
        to
                t : Persons!Female (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}
```

QVT

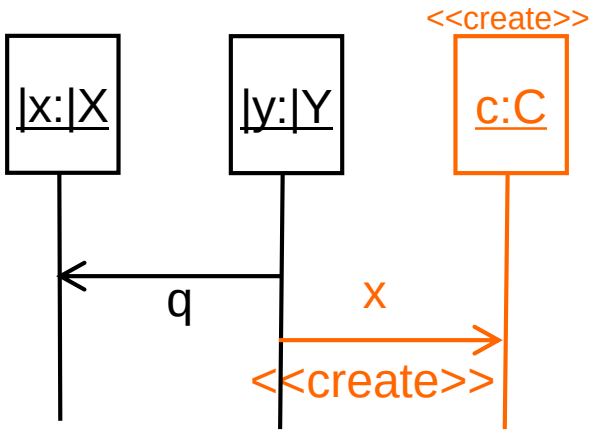

«domain»
c: Class
name = n
«domain»
t: Table
name = n
uml1 : UML
r1:RDBMS
c
c
a:Attribute
name = an
col : Column
name = an

Model Compositions as Transformations

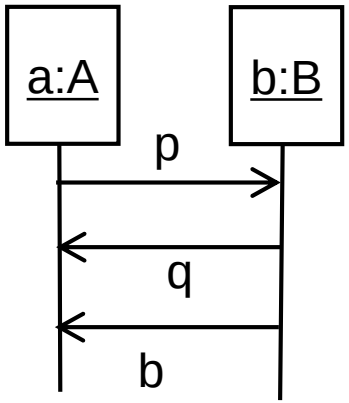
Using Graph Transformations

Simple Example (1)

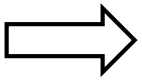
aspect scenario

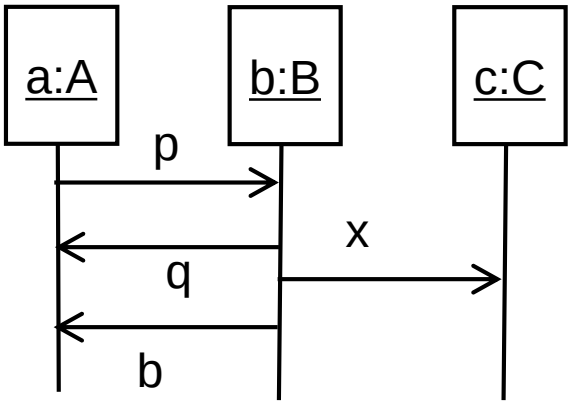


1. Match unstereotyped elements against base
2. Modify base according to stereotyped elements



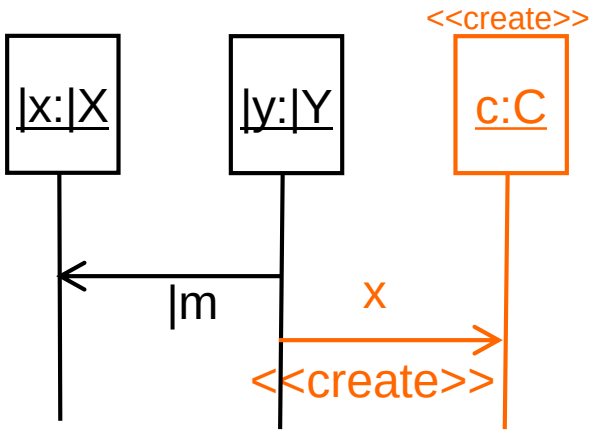
example application

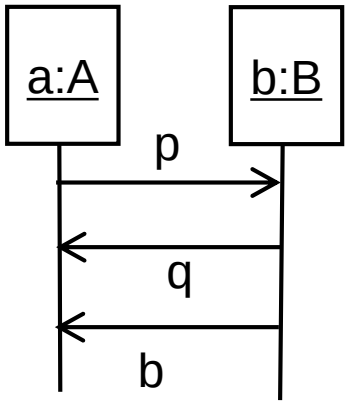




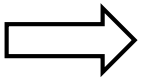
Simple Example (2)

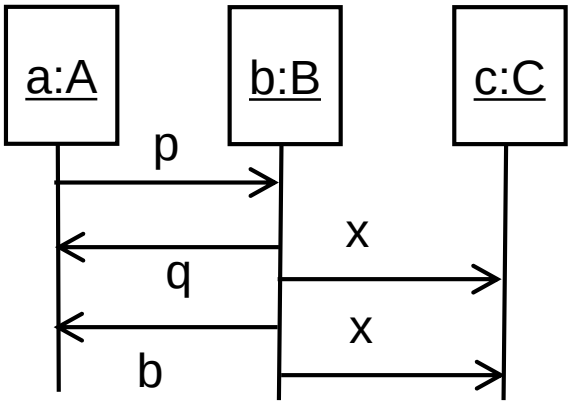
aspect scenario





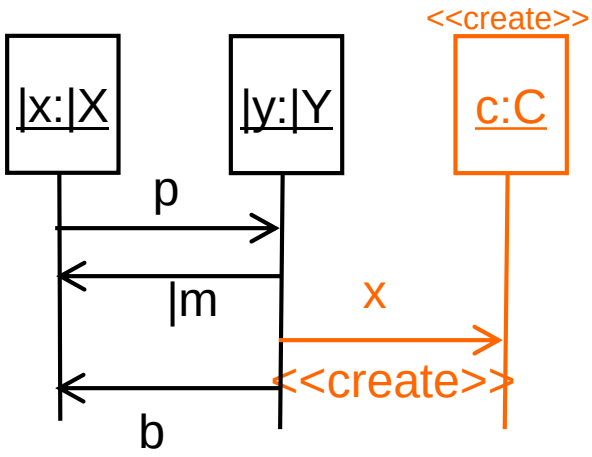
example application

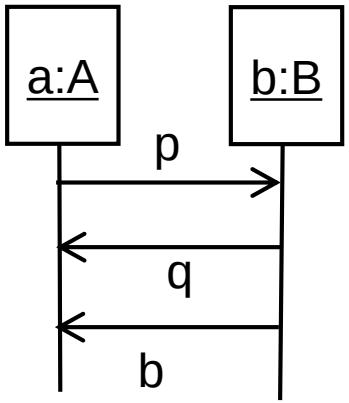




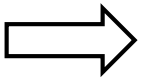
Simple Example (3)

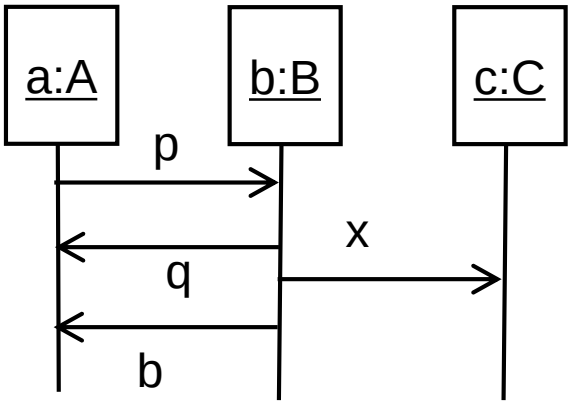
aspect scenario





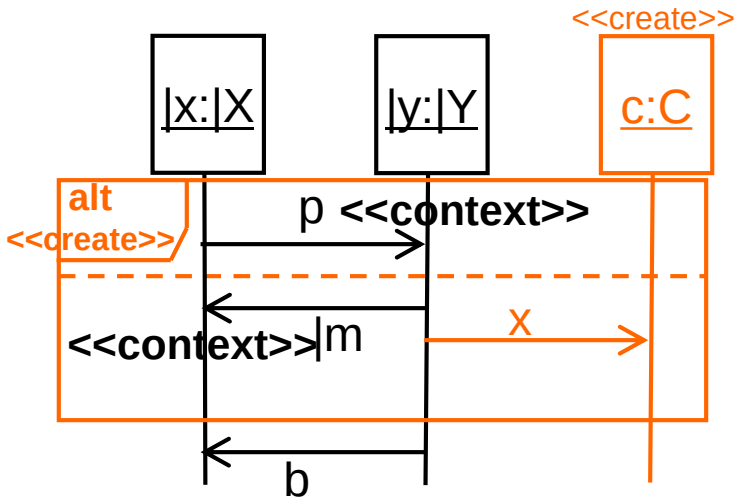
example application

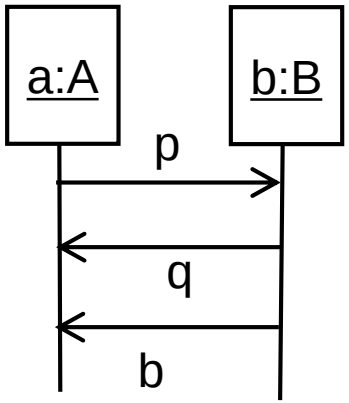




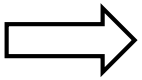
Simple Example (4)

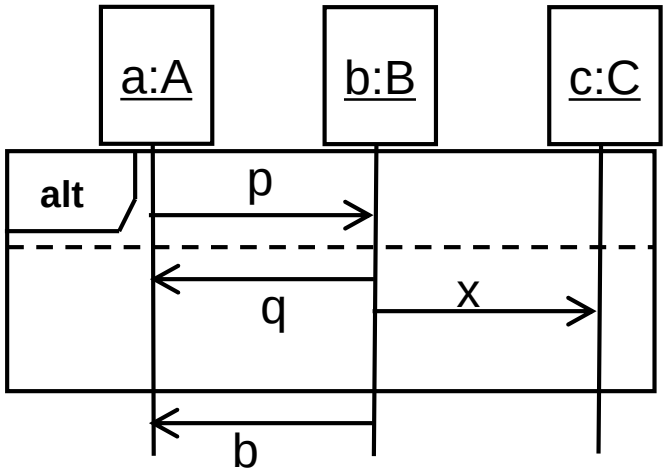
aspect scenario





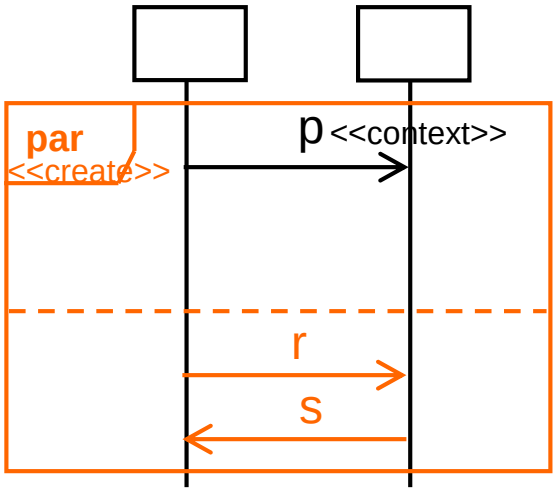
example application

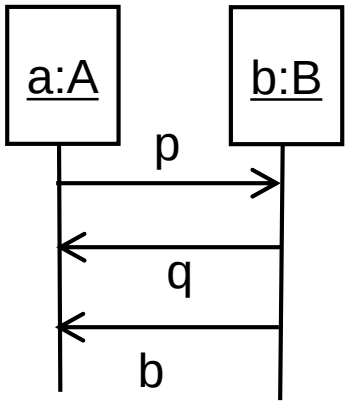




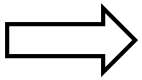
More complex example (1)

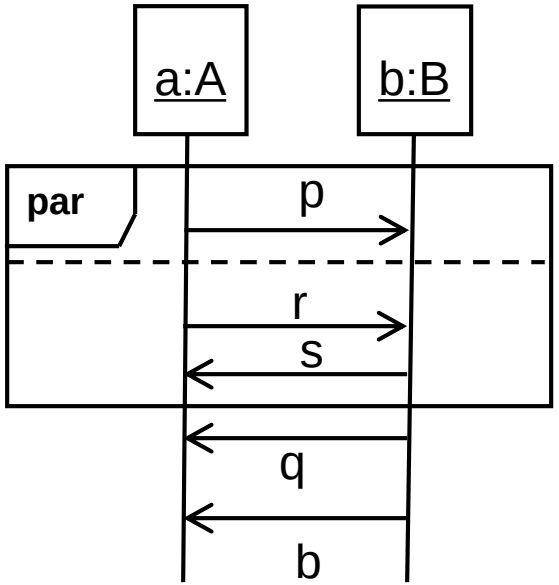
aspect scenario





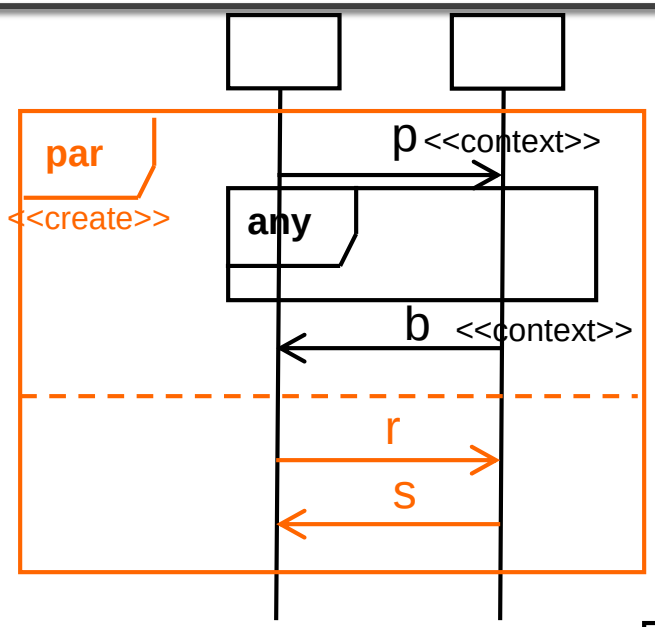
example application

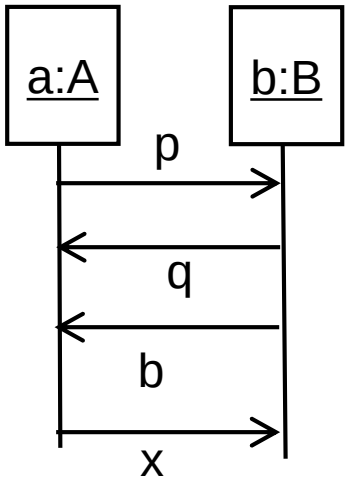




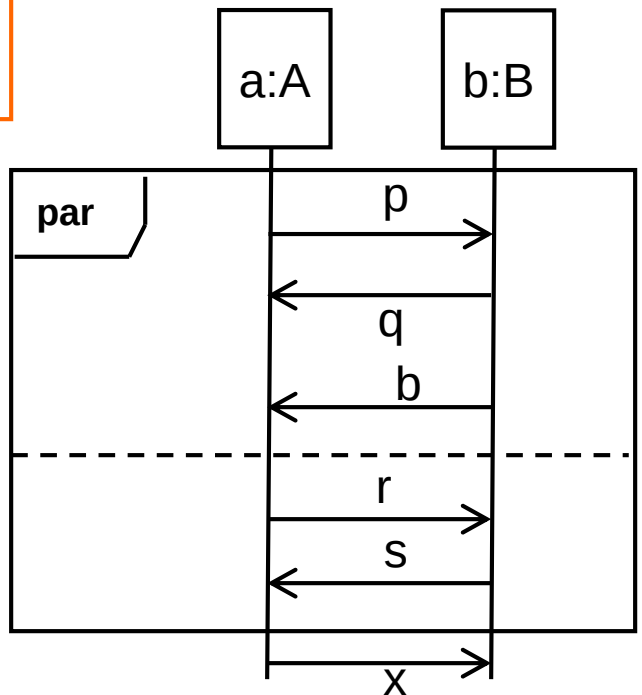
More complex example (2)

aspect scenario

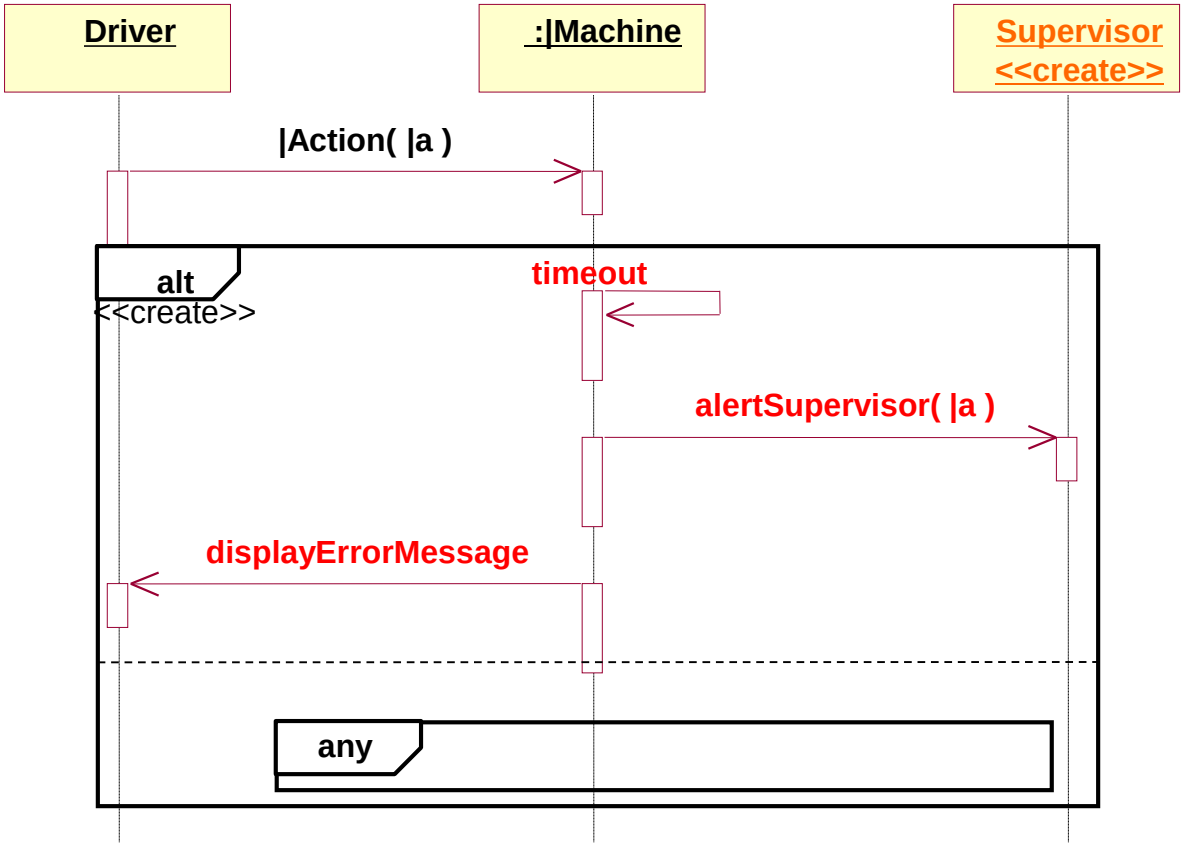




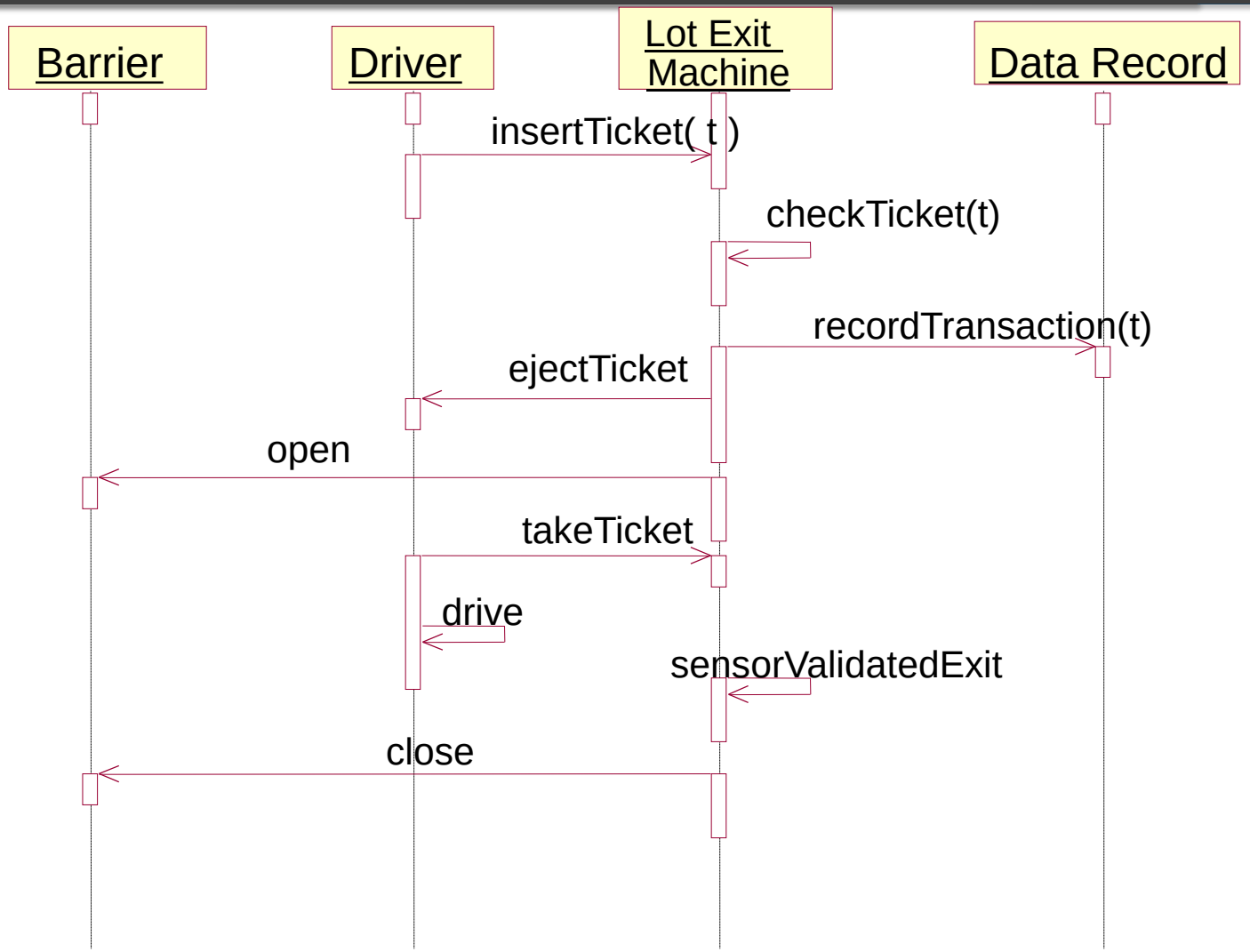
example application



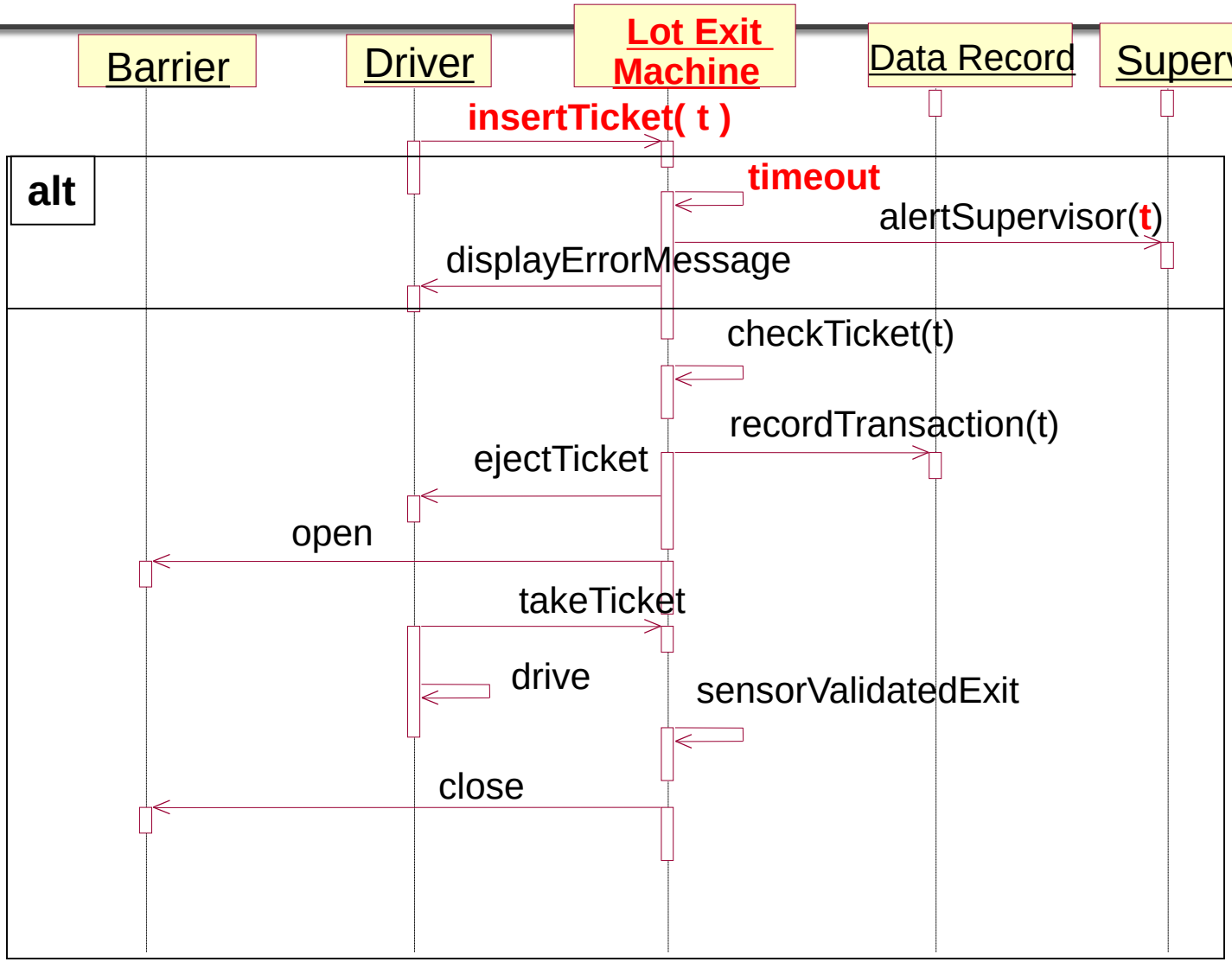
Scenario:
“Machine is broken” (A1)



Car parking example: “Exiting with a paid ticket” (UF1-I1)



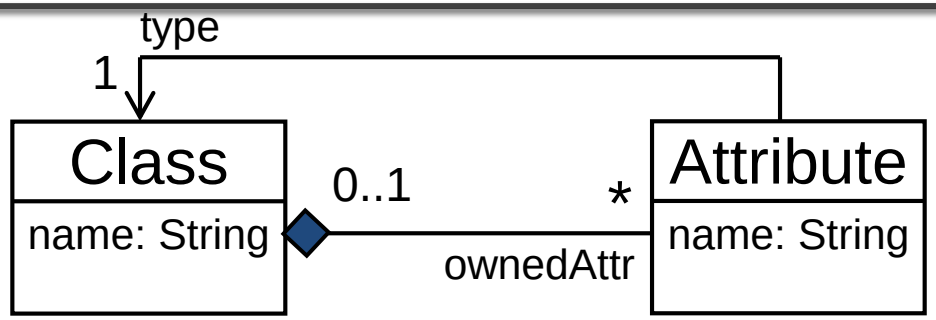
Composed Interaction



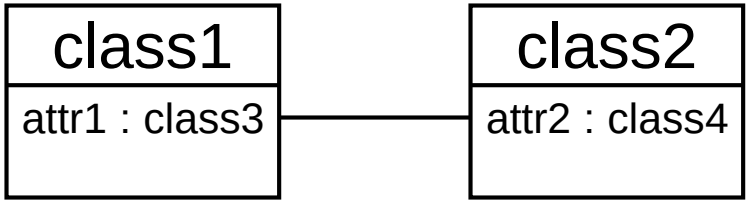
red = former roles

Graph Transformation

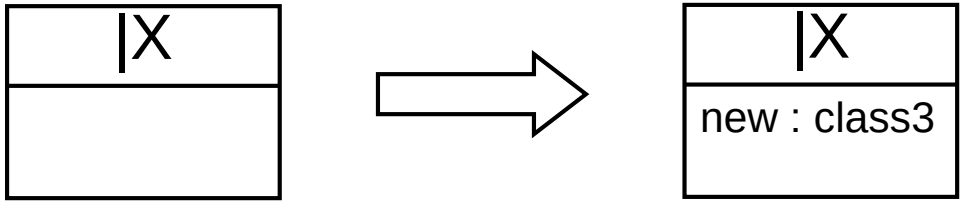
(a) Type Graph



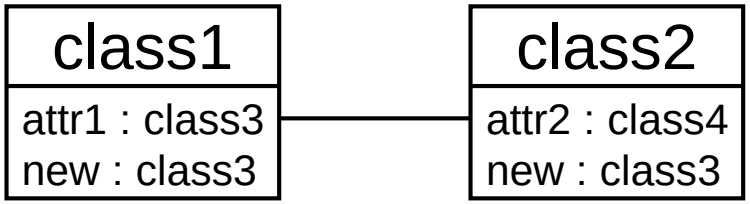
(b) UML Model



(c) Graph Rule



(d) Transformed UML Model



Agile methods and MDA

- The developers of MDA claim that it is intended to support an iterative approach to development and so can be used within agile methods.
- The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto, so few agile developers feel comfortable with model-driven engineering.
- If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required.

Adoption of MDA

- A range of factors has limited the adoption of MDE/MDA
- Specialized tool support is required to convert models from one level to another
- There is tool availability but organizations may require tool adaptation and customisation to their environment
- For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business

MDE tools

- http://www.mdetools.com/
- FUJABA (UML to Java)
- AToM3 (graph transformations)
- Merlin (Eclipse plugin)
- JTM (bidirectional transformation language)
- Medini QVT
- GReAT (Graph rewriting and Transformation)
- Viatra (transformation language)
- MOLA (research)
- ...

Adoption of MDA

- Models are a good way of facilitating discussions about a software design. However, the abstractions that are useful for discussions may not be the right abstractions for implementation.
- For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

Adoption of MDA

- For software products and information systems, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.
- The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.

Models and System Perspectives

System perspectives

- An **external perspe**ctive, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

Use of graphical models: completeness and correctness levels

- As a means of facilitating discussion about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.

Context models

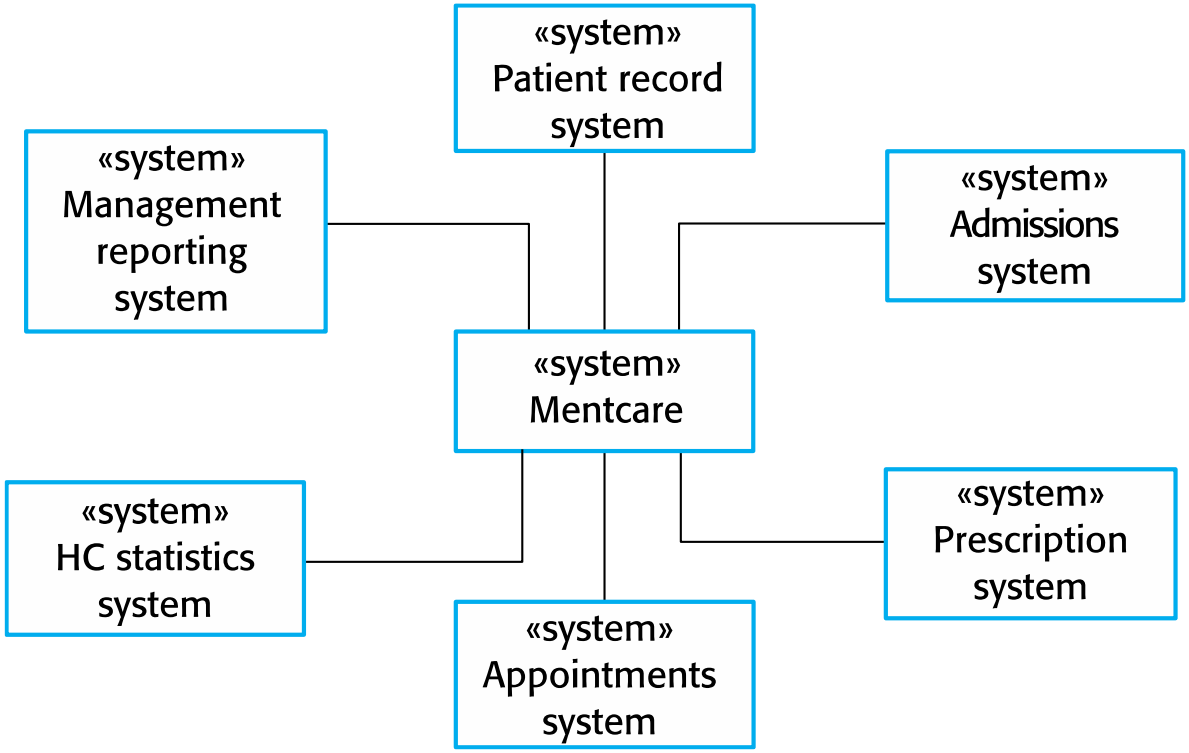
Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

System boundaries

- System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

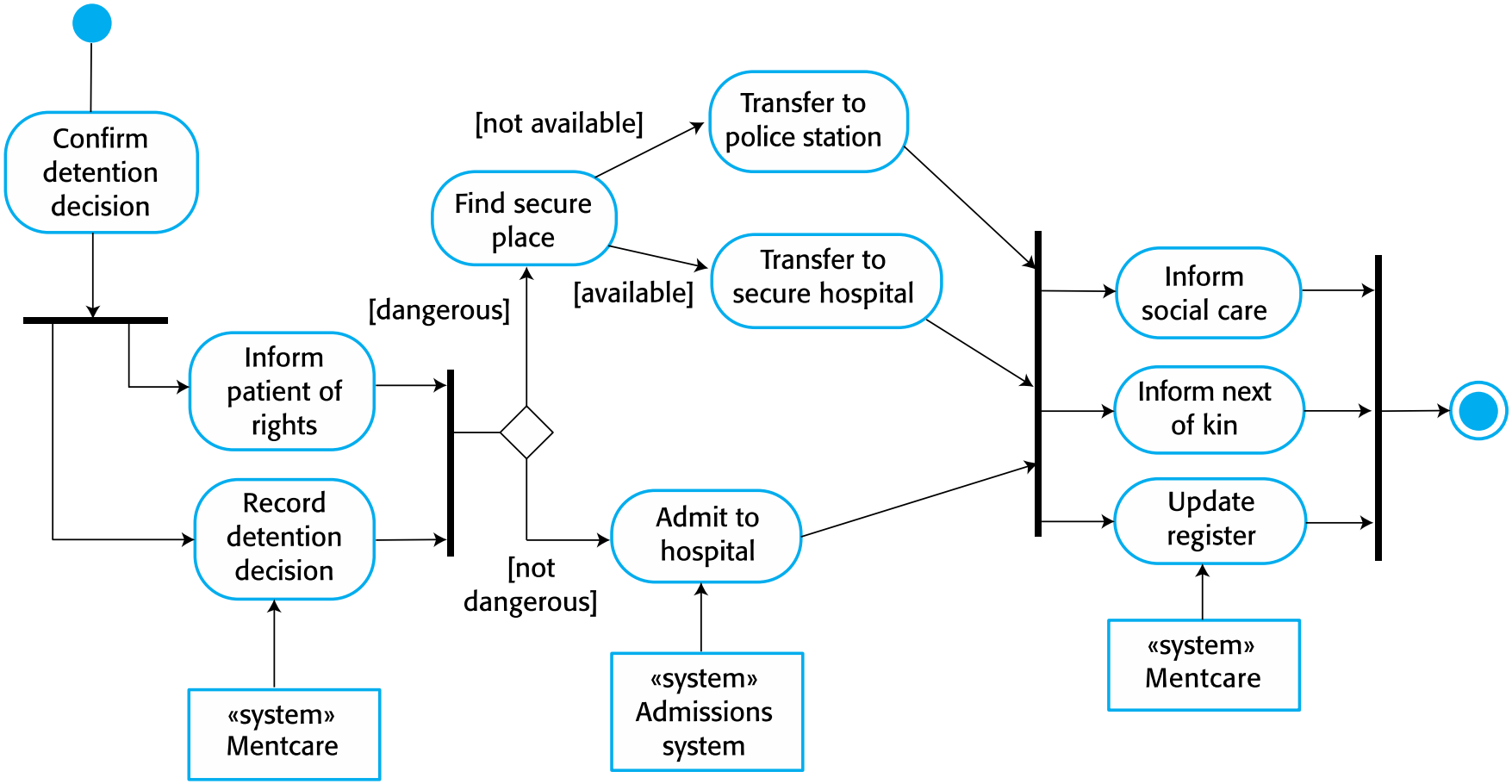
The context of the Mentcare system



Process perspective

- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
- Process models reveal how the system being developed is used in broader business processes.
- UML activity diagrams may be used to define business process models.

Process model of involuntary detention



Interaction models

Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- **Use case diagrams and sequence diagrams** may be used for interaction modeling.

Use case modeling

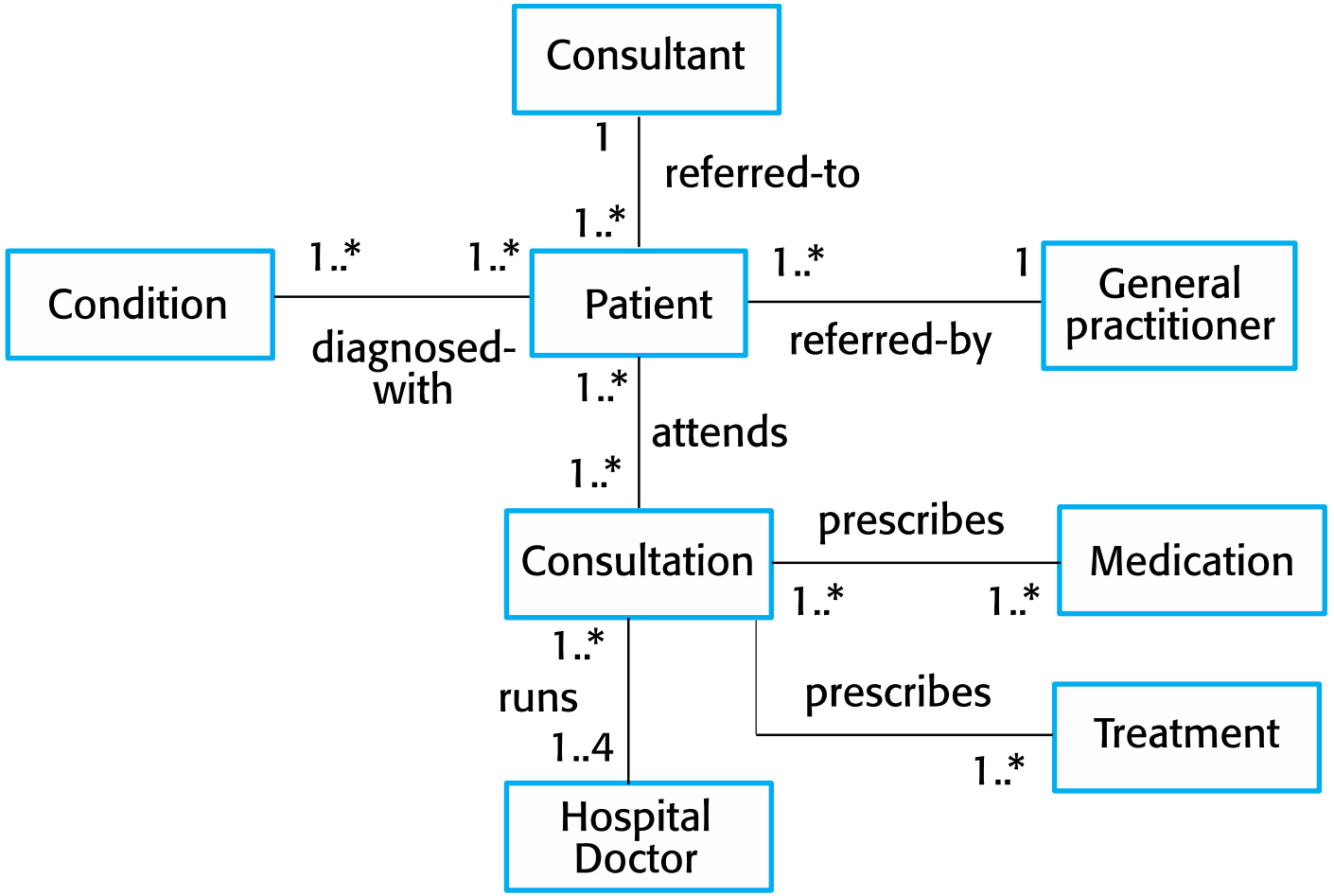
- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people or other systems.
- Represented diagramatically to provide an overview of the use case and in a more detailed textual form.

Structural models

Structural models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

Classes and associations in the MHC-PMS



Behavioral models

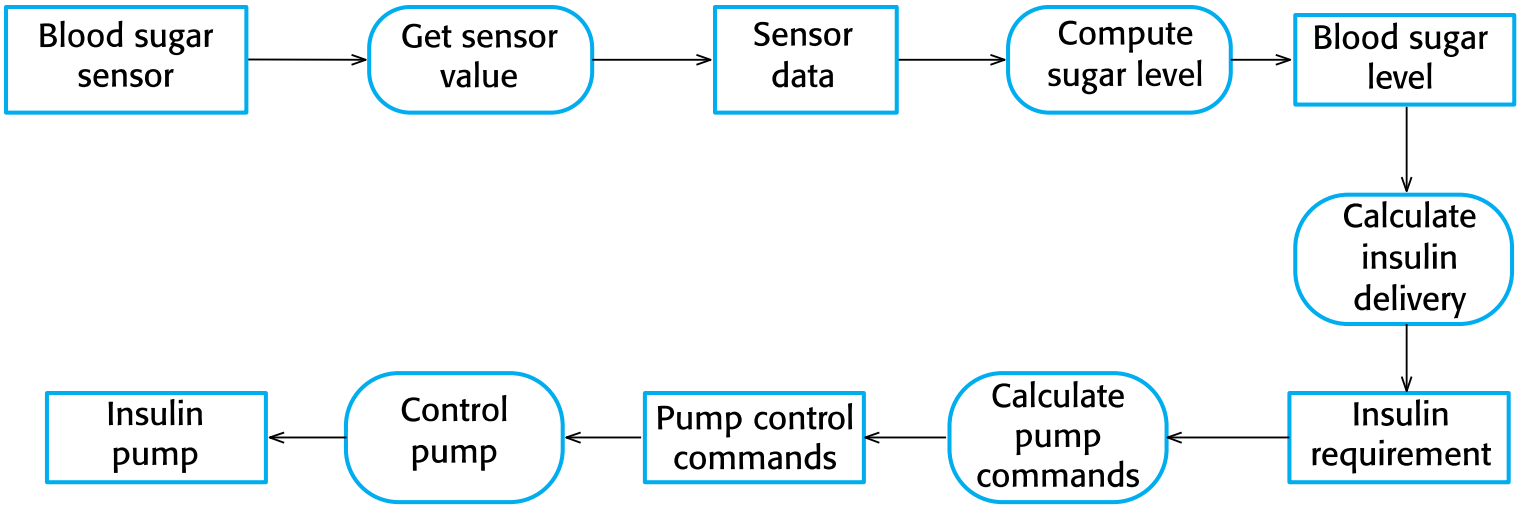
Behavioral models

- Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- You can think of these stimuli as being of two types:
  - Data Some data arrives that has to be processed by the system.
  - Events Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Data-driven modeling

- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
- They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

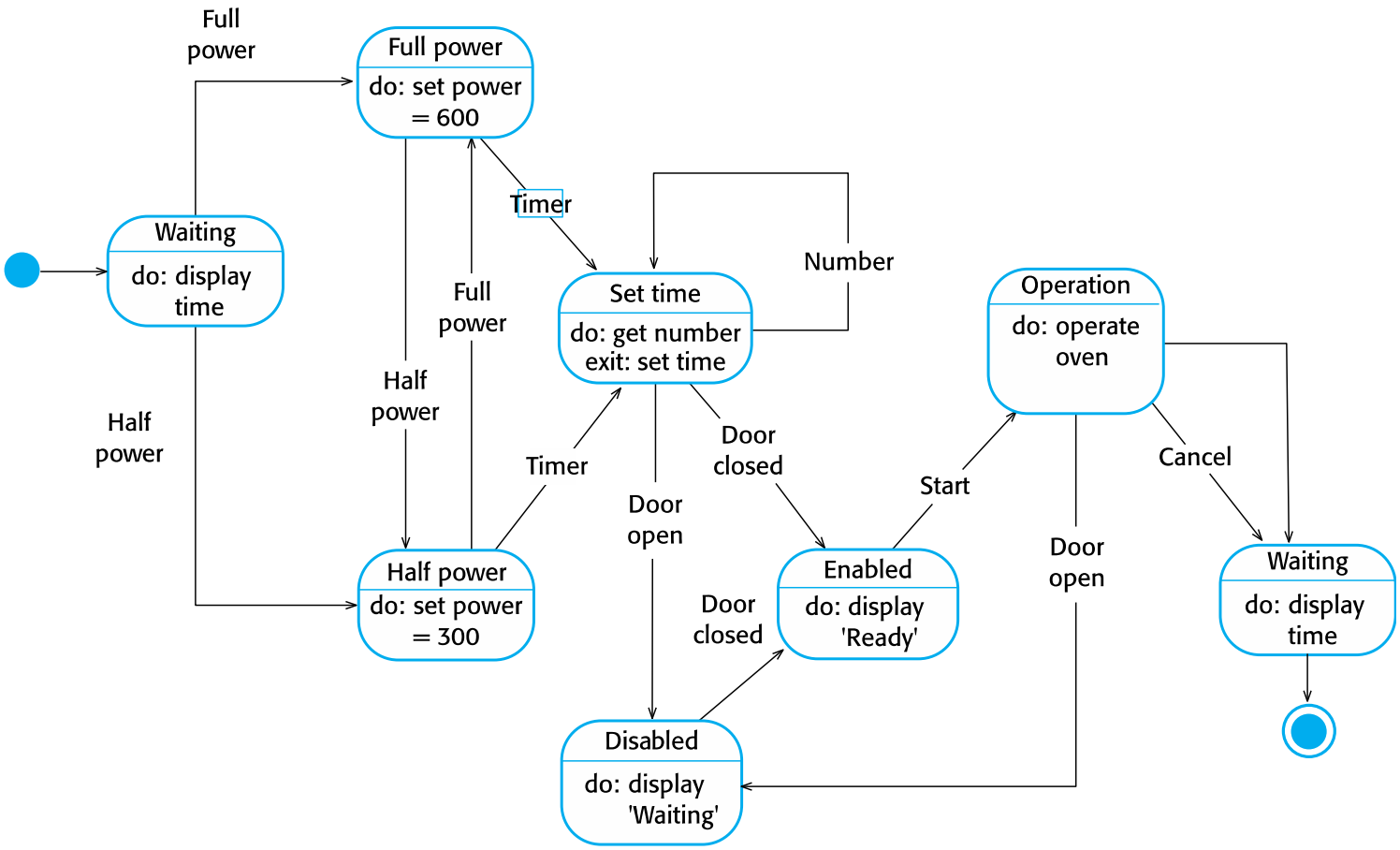
An activity model of the insulin pump's operation



Event-driven modeling

- Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

State diagram of a microwave oven



Microwave oven operation