# Software Engineering – Lecture 4
# System Modelling

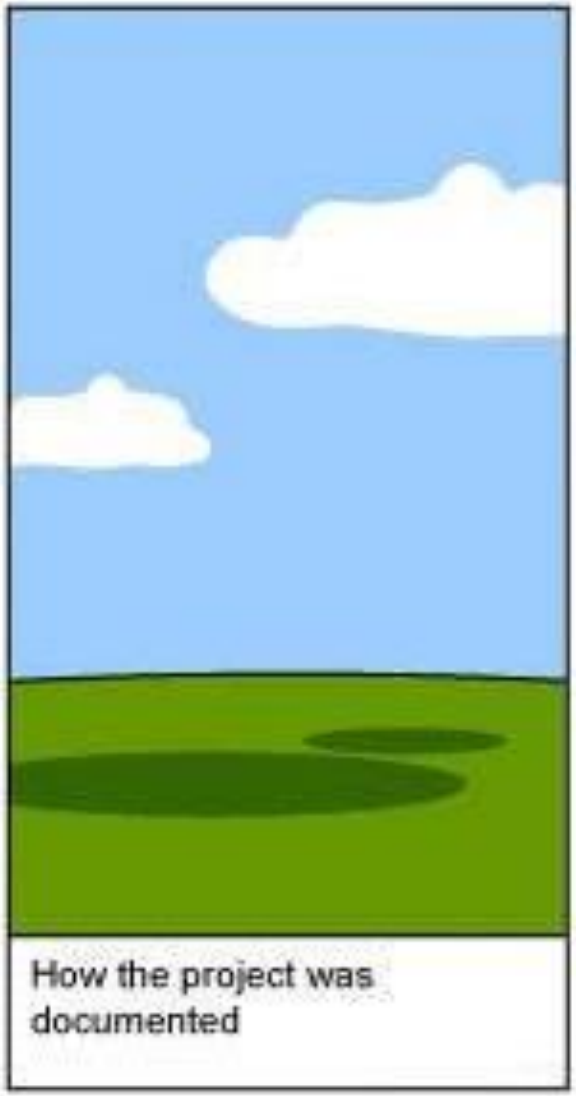

How the customer explained it

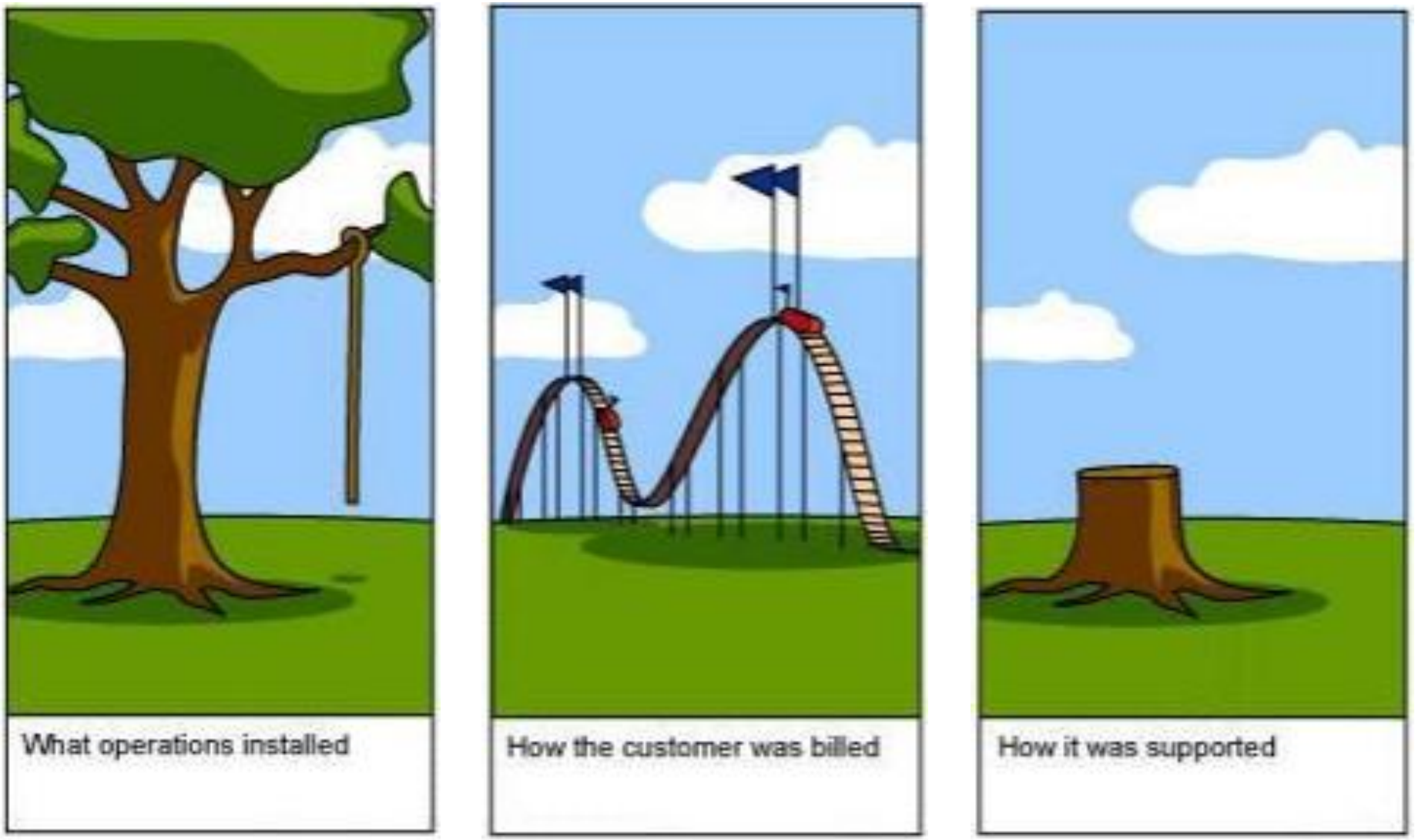

How the Project Leader understood it


How the Analyst designed it

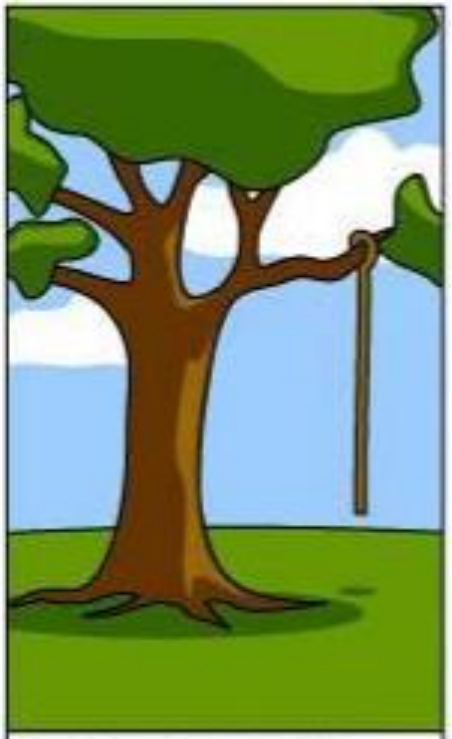

How the Programmer wrote it


How the Business Consultant described it


How the project was documented


What operations installed
How the customer was billed
How it was supported

How the customer explained it



What operations installed

What the customer really needed

# System models

System modelling is the process of developing *abstract models* of a system.

System models express system ***requirements*** in a more ***technical way.***

***System specifications*** are documented using a ***set of system models.***

Important bridge between the analysis and design processes.

# System modelling

System modelling helps the analyst to understand the system.

Models are used to communicate with customers.

Different models present the system from different perspectives:

- *External* perspective - showing the system's context / environment;
- *Interaction* perspective - showing the interactions between a system and its environment or between the components of a system.
- *Behavioral* perspective - showing the behaviour of the system;
- *Structural* perspective - showing the system architecture or data architecture.

# System modelling

Models of the existing system - used to

- Clarify what the existing system does
- Basis for discussing its strengths and weaknesses
- Lead to requirements for the new system

Models of the new system - used to

- Explain the proposed requirements to other system stakeholders
- Discuss design proposals
- Document the system for implementation.

In a model-driven engineering process – used to

- Generate a complete or partial system implementation from the system model.

# Model types

- A system model is an *abstraction of a system*.

Abstraction deliberately simplifies and leaves out details.

- Model types are based on different approaches to abstraction:

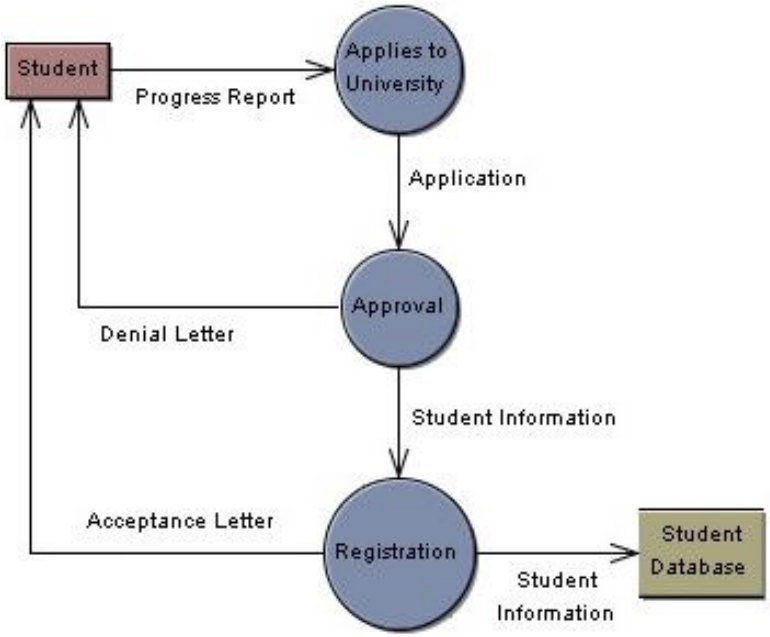
Ex.

*Data-flow* model concentrates on *flow of data* and *functional transformations* on that data; it leaves out details of the data structure.

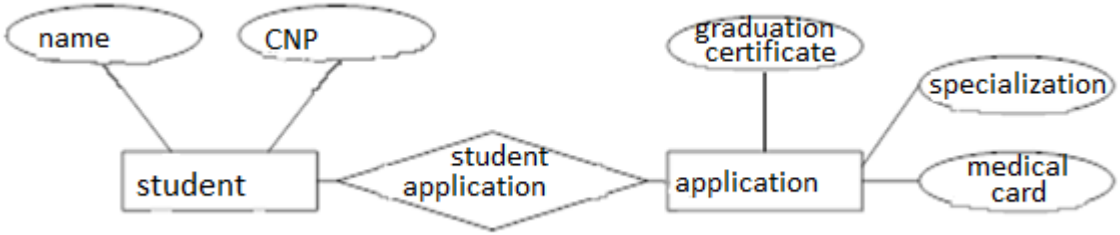
*Entity-relationship-attribute* model documents system *data structure* rather than its functionality.

# Model types - example

*Data-flow* model concentrates on *flow of data* and *functional transformations* on that data, leaves out details of the data structure.




*Entity-relationship-attribute* model documents system *data structure* rather than its functionality.



# UML modelling

UML – largely used graphical notation for system modelling

- ***Activity*** *diagrams* - the activities involved in a process or in data processing.
- ***Use case*** *diagrams* – the functions exposed by the system and their connections to the actors in its context.
- ***Sequence*** *diagrams* - interactions between actors and the system and between system components.
- ***Class*** *diagrams* - object classes in the system and their relationships.
- ***State machine*** *diagrams* - system behavior triggered by internal and external events.

# Use of graphical models

- 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.

# Topics covered

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

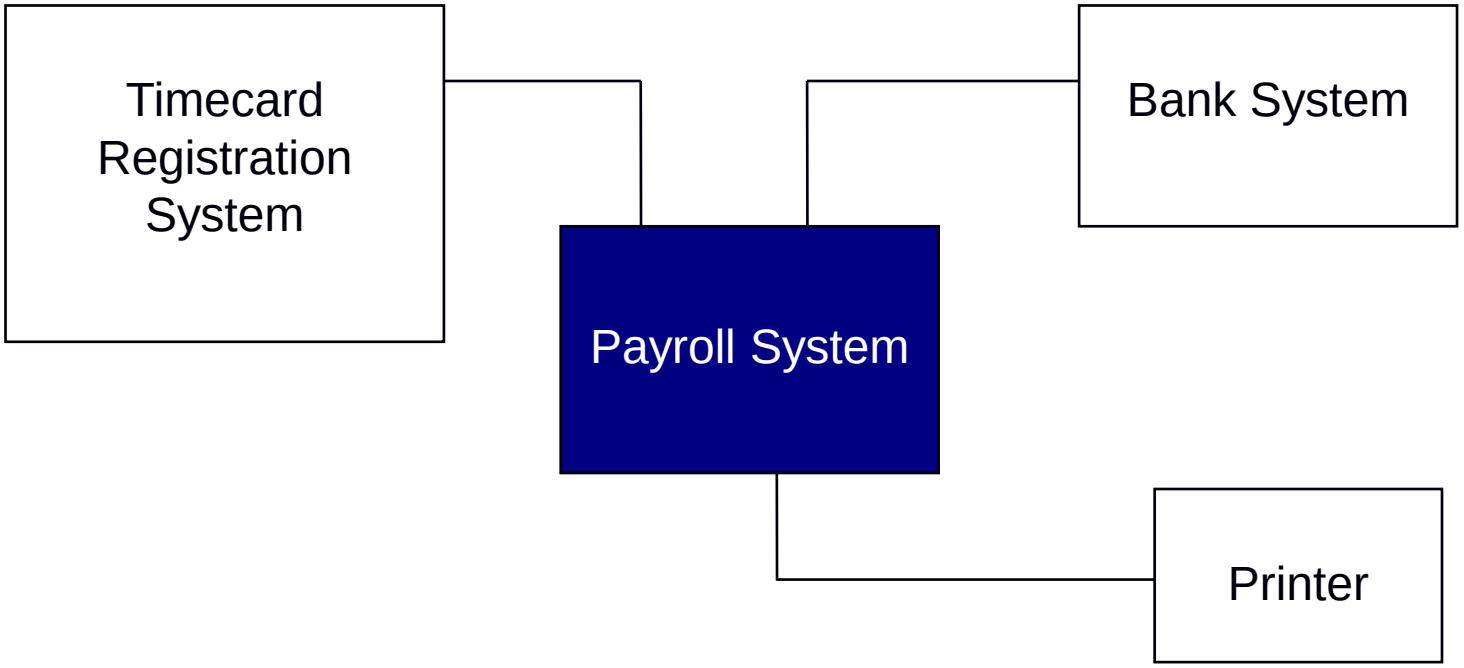
# Context models

Context models are used to illustrate the ***operational context of a system*** - they show what lies outside the *system boundaries*.

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

# Example: The context of Payroll System

*High-level architectural models* show the system and its relationship with other systems.




High-level architectural model; expressed as a block diagram.

Relationships are not defined => need to be supplemented by other models.

# Process perspective

- *Context* models simply show the *other systems* in the environment, not how the system being developed is used in that environment.

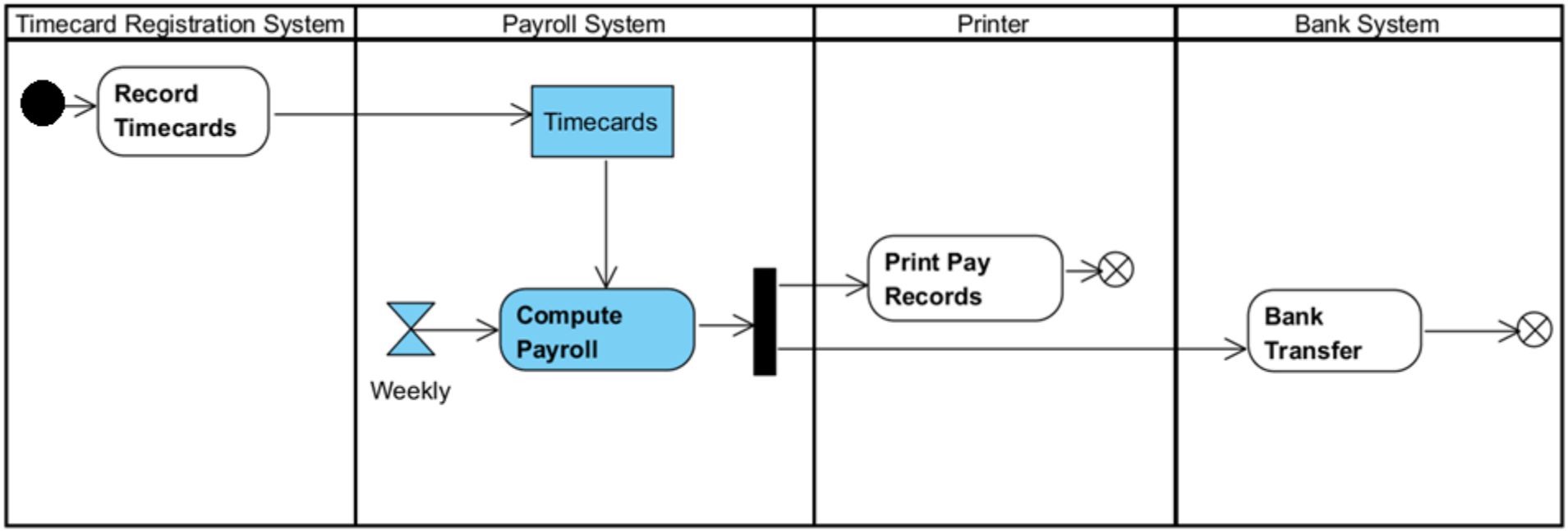
- ***Process*** models reveal how the system being developed is used *in broader business processes*.

Notation:

- UML activity diagram
- BPMN  (Business Process Model and Notation)

# Process perspective

## Example: Payroll system in its operational context



# Topics covered

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# Interaction models

- Modelling *user interaction*

Helps identify user *requirements*.

- Modelling *system-to-system interaction*

Highlights the *communication* problems that may arise.

- Modelling *component interaction*

Helps understand if a proposed system structure is likely to deliver the required system performance and dependability.

Used UML diagrams:

*Sequence diagrams* - model interactions

- between actors and the system
- between system components (external agents may also be included).

# Use case modelling

- Use cases - support requirements elicitation.

- Each use case represents a *discrete task* that generally *involves external interaction* with an actor.

OBS. Included or extending use cases may not interact directly with actors.

- Actors in a use case may be people or other systems.

Representation:

- UC diagram - an overview of the use cases
- detailed (textual) form for each use case

*https://whyusecases.ivarjacobson.com/?_ga=2.149213937.693533130.1615812350-1690294733.1615812350*

# Sequence diagrams

- Sequence diagrams - model the interactions between the actors and the objects within a system.

  Sequence diagram - the sequence of interactions that take place during a particular use case or use case instance (scenario).

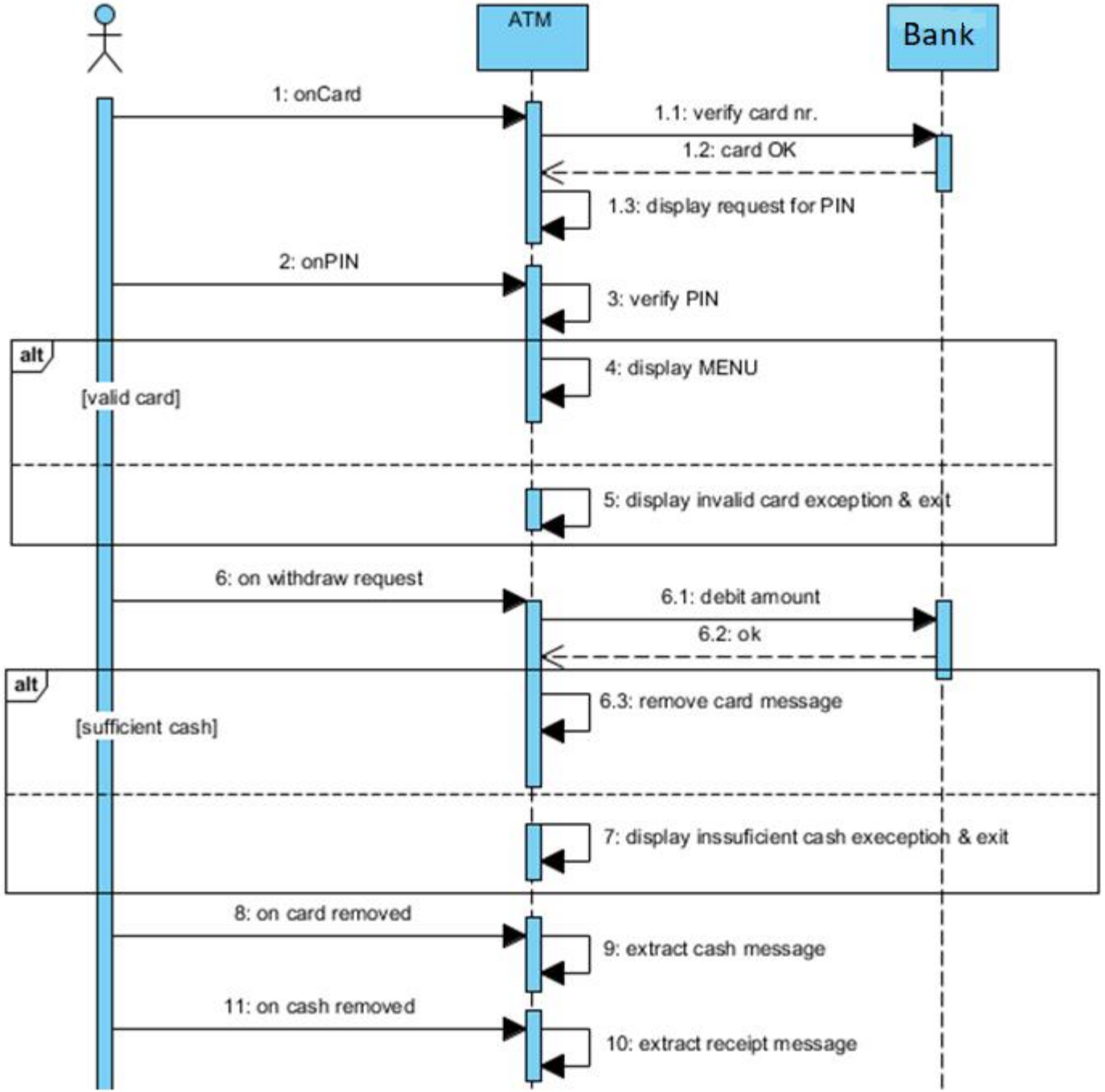
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.

- Interactions between objects are indicated by annotated arrows.

Example:

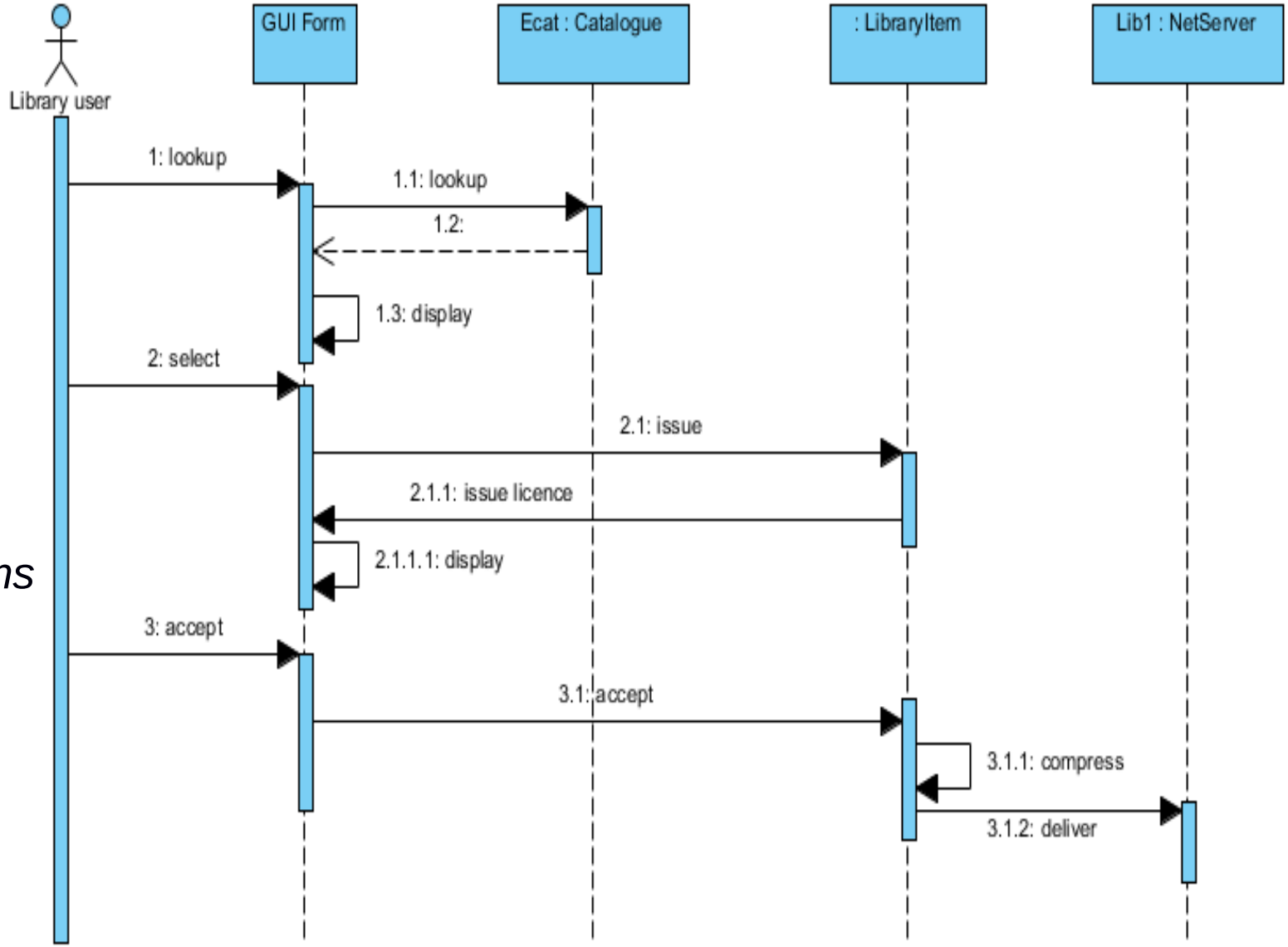
System level sequence diagram :

*ATM withdrawal*



Example:

Sequence diagram for the use case *Issue of electronic items*



# Topics covered

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# Structural models

- Structural models of software - the organization of a system in terms of the *elements* that make up that system and of their *relationships*.

- Structural models perspectives:
  - *static* - show the structure of the system design
  - *dynamic* - show the organization of the system at runtime.

- Structural models of a system are created when discussing and designing the *system architecture*.

Used UML diagrams:

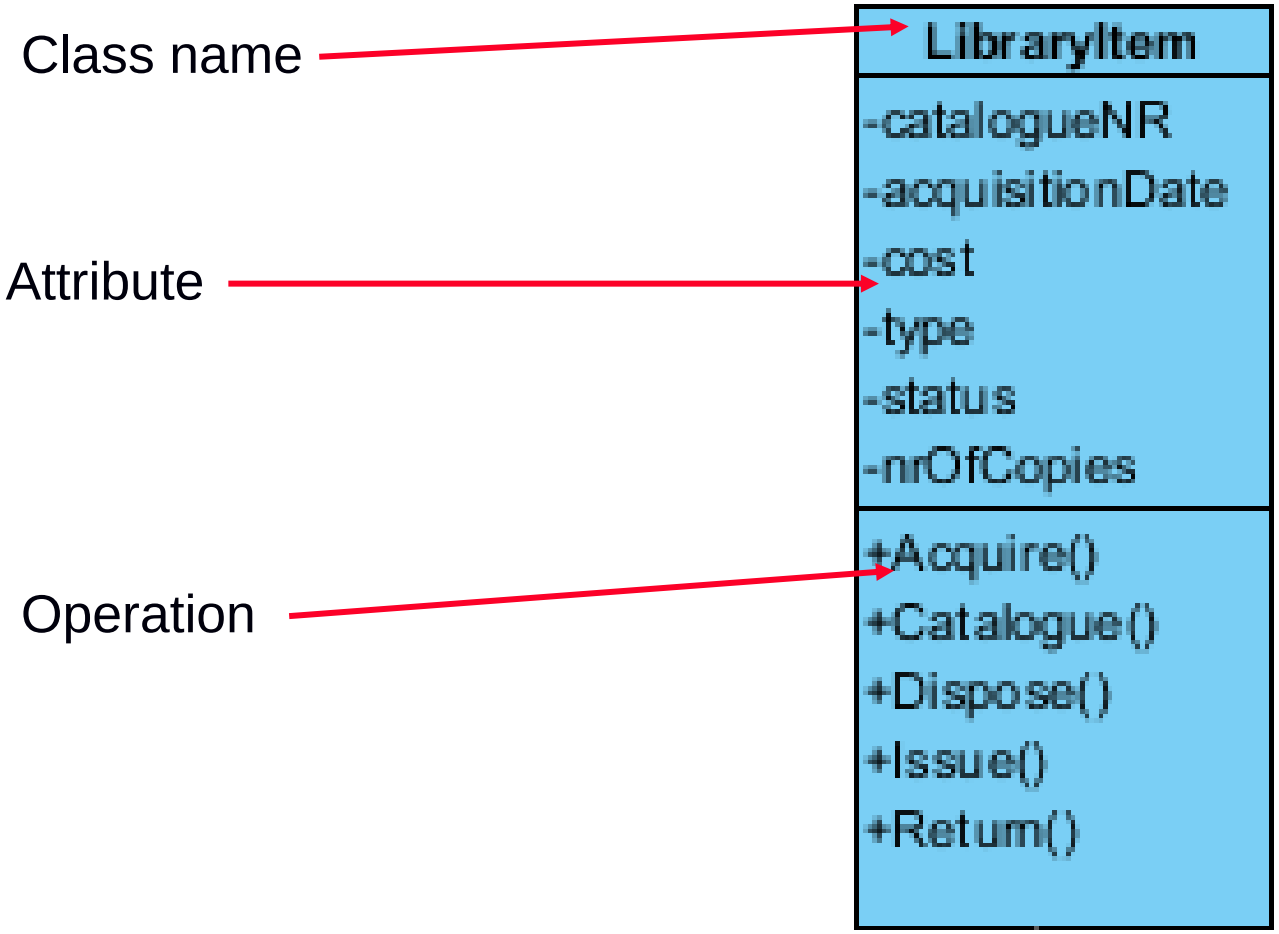
- Class diagram – static perspective (***units of code***)
- Component diagram – dynamic perspective (***units of execution***)

# Class diagram

In OO development approaches:

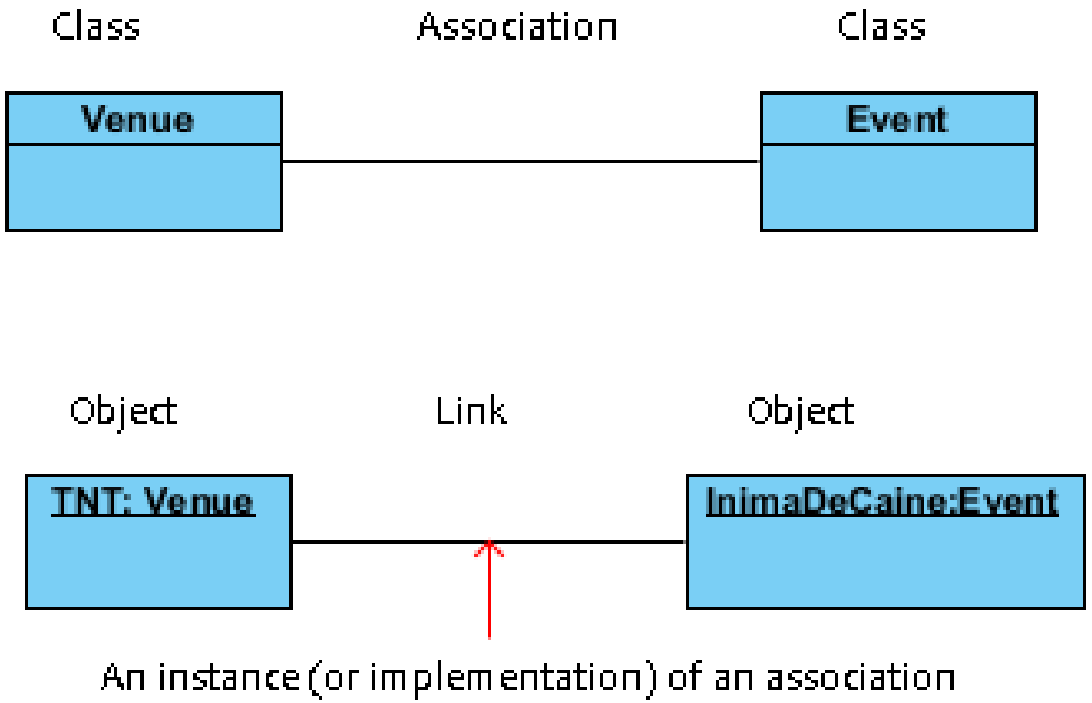
- *Software system* is structured as a collection of *interacting objects*.
- Object class = *abstraction* over a set of objects with common *attributes* and the *services* (operations) provided by each object in the class.
- Class diagrams describe the system in terms of object *classes* and their *relations*.

# UML notation



# Class relation type: Association

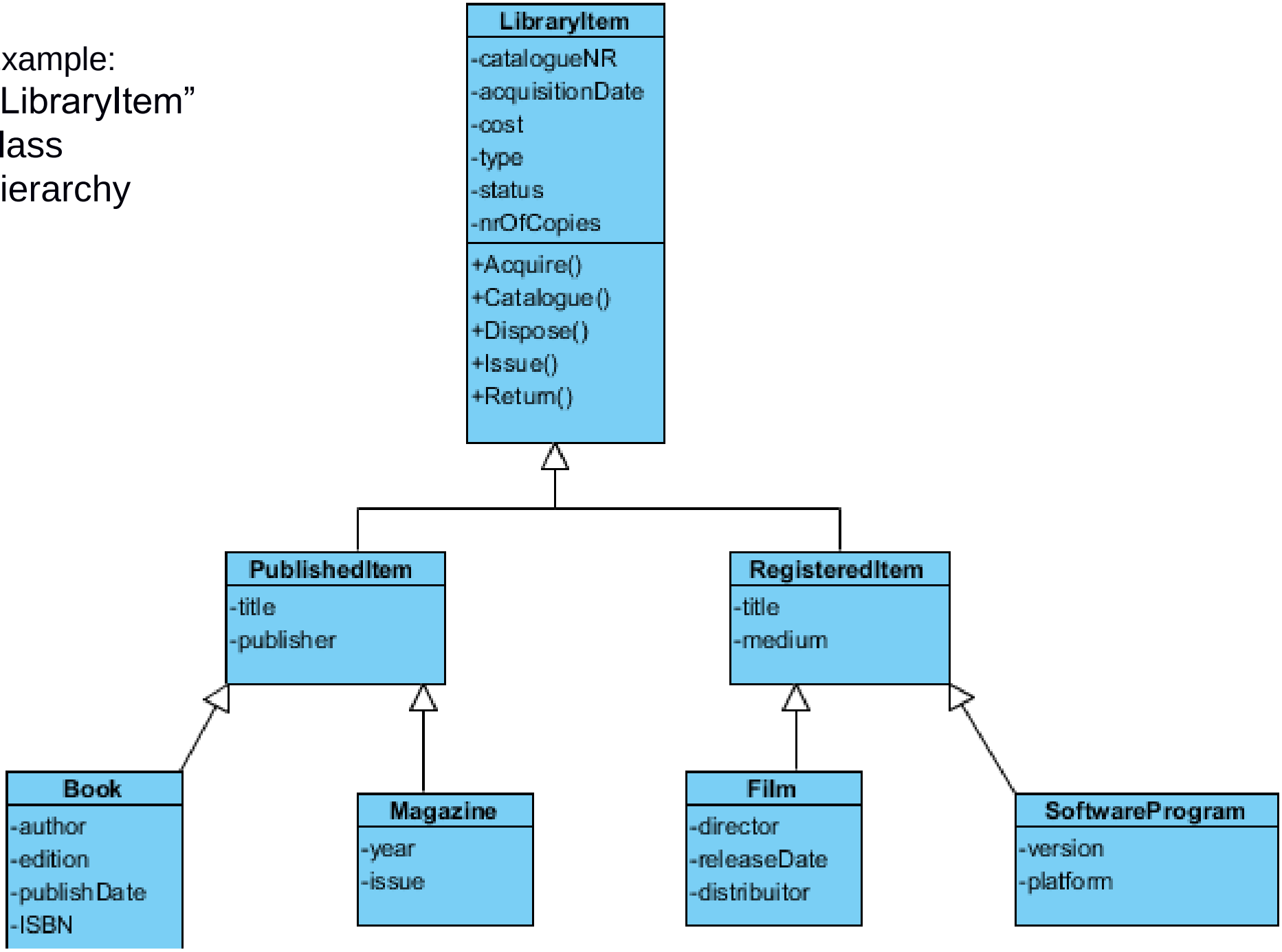
General relation meaning that the objects instantiated from associated classes ale linked.



# Class relation type: Generalization

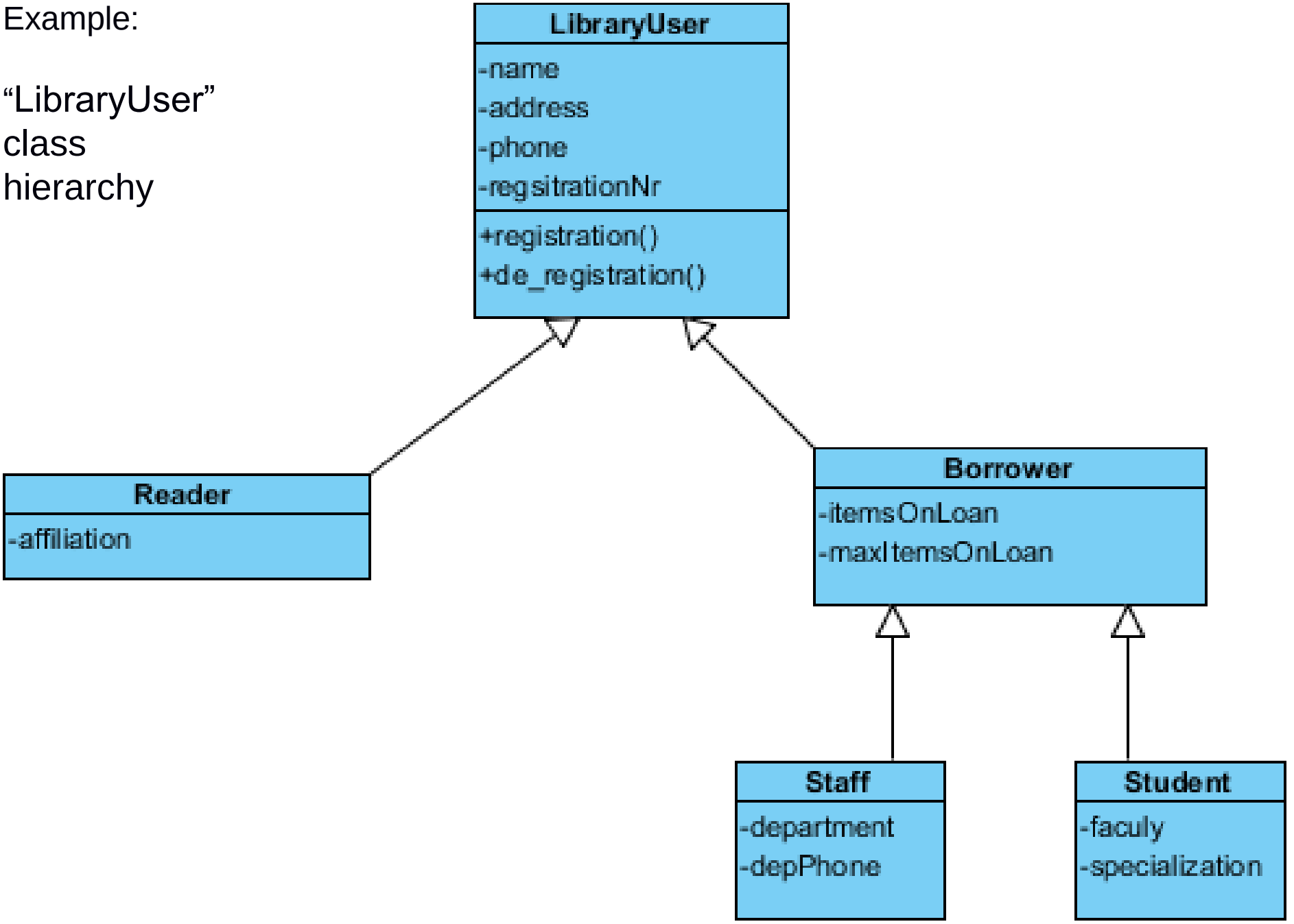
- Generalization - technique to manage complexity by *abstracting common characteristics* of more classes.
- Generalization is similar to the "*is-a*" relationship in semantic data models.
- In object-oriented languages (ex. Java) generalization is implemented using the class inheritance mechanisms built into the language.
- In a generalization, the attributes and operations associated with the *superclass* are also associated with all its *subclasses*.
- A subclass inherits all the attributes and operations from its superclasses. Subclasses can add more specific attributes and operations.

Example:
"LibraryItem" class hierarchy



Example:

"LibraryUser" class hierarchy



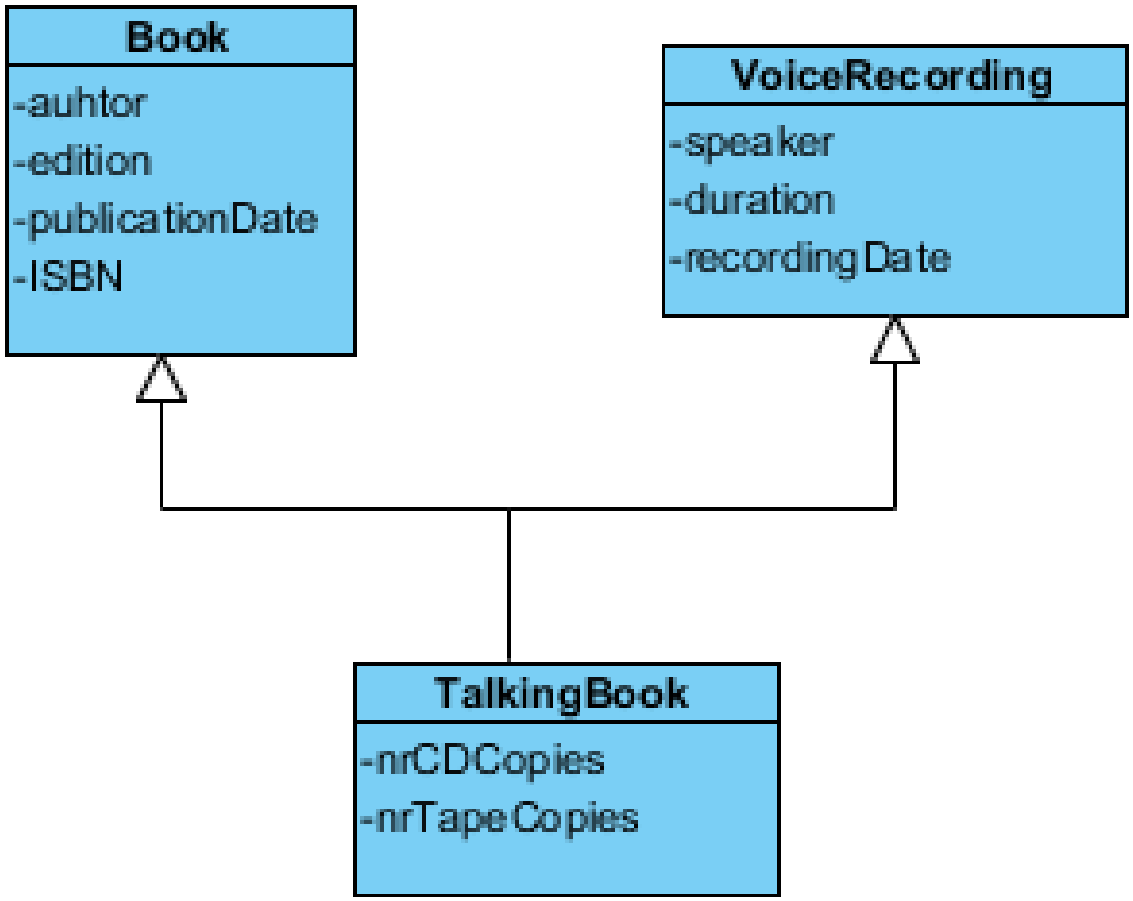
# Multiple inheritance

- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to *inherit from several super-classes*.

Obs.

- This can lead to *semantic conflicts* where attributes/services with the *same name* in different super-classes have *different semantics*.
- Multiple inheritance makes class *hierarchy reorganisation more complex*.

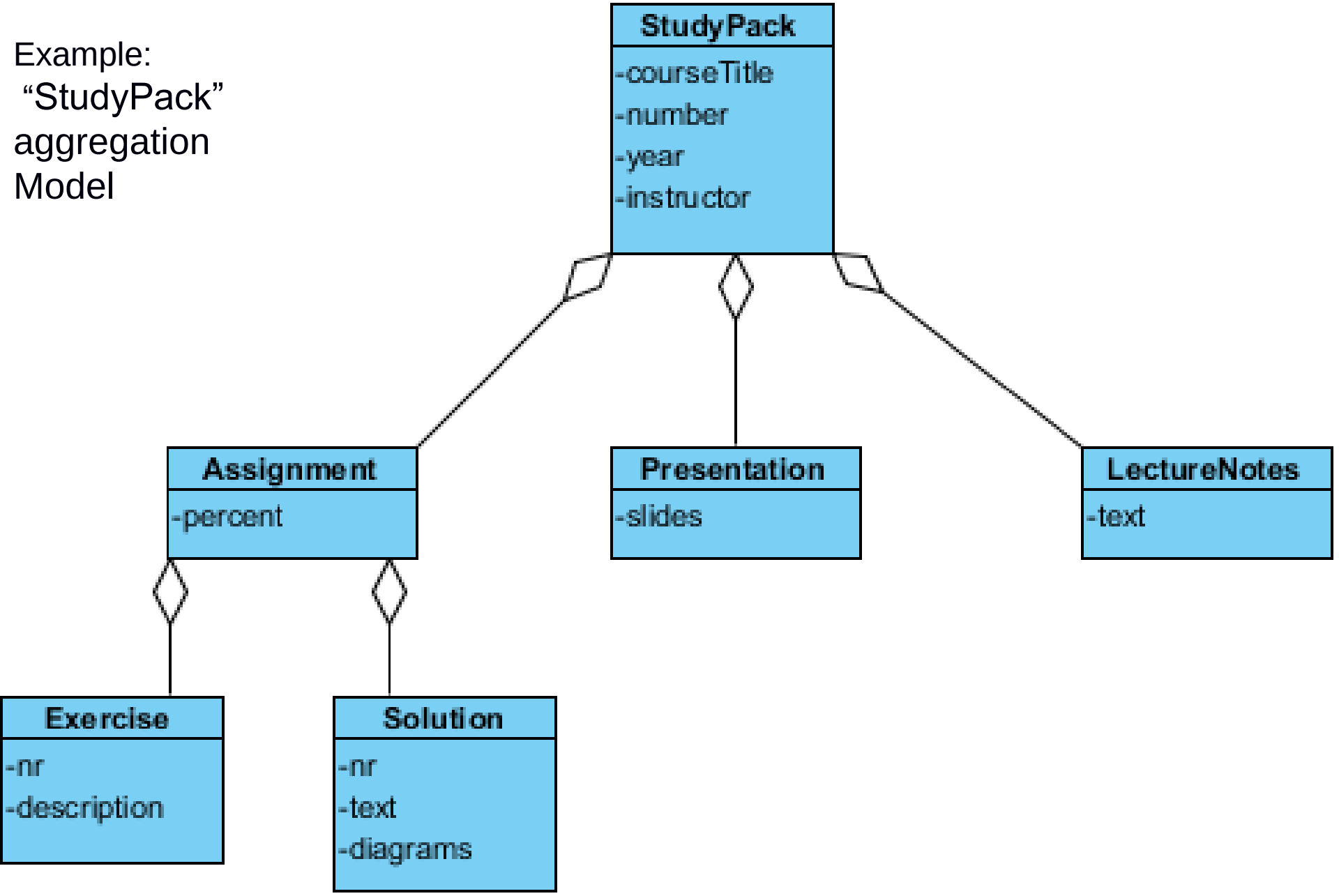
# Multiple inheritance: Example



# Class relation type: Aggregation

- An aggregation model shows how classes that are collections are *composed* of other classes.
- Aggregation models are similar to the "*part-of*" relationship in semantic data models.
- The object which represents the collection keeps a data structure of type `Collection` containing the references to the objects that form the aggregate.
- The element in the collection may keep a reference to the object representing the collection.

Example:
"StudyPack" aggregation Model



# Formative evaluation

1. In the following class diagram, which are the attributes of class Staff and which are the operations of class Student ?

2. In the following class diagram, which are the attributes of class Assignment ?

https://forms.gle/x1h9Y9Kjb2SAHpU46

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# Behavioral models

Model the *runtime behavior* of the system.

Types of behavioural modelling :

- *Data-driven modelling :* models show the sequence of actions involved in processing input data and generating an associated output;
- *Event-driven modelling :* models show how the system responds to events.

Obs. These models show different perspectives, so both of them are required to describe the system's behaviour.

# Data-driven modelling

Show how data is processed as it moves through the system.

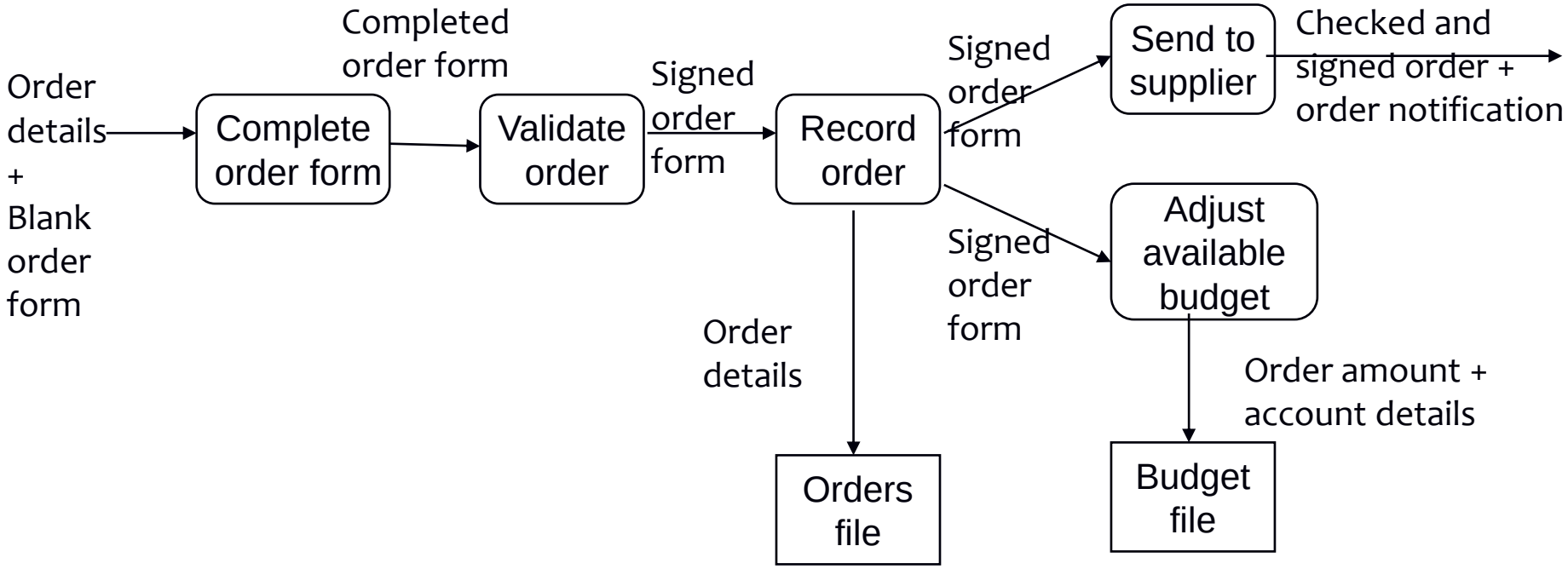
Types of diagrams:

- Data flow diagram (DFDs)
- UML activity diagram

Utility

- Tracking and documenting *how the data is associated with a process* is helpful to develop an overall understanding of the system.
- Data flow diagrams may also be used in showing the *data exchange* between a system and other systems in its *context*.

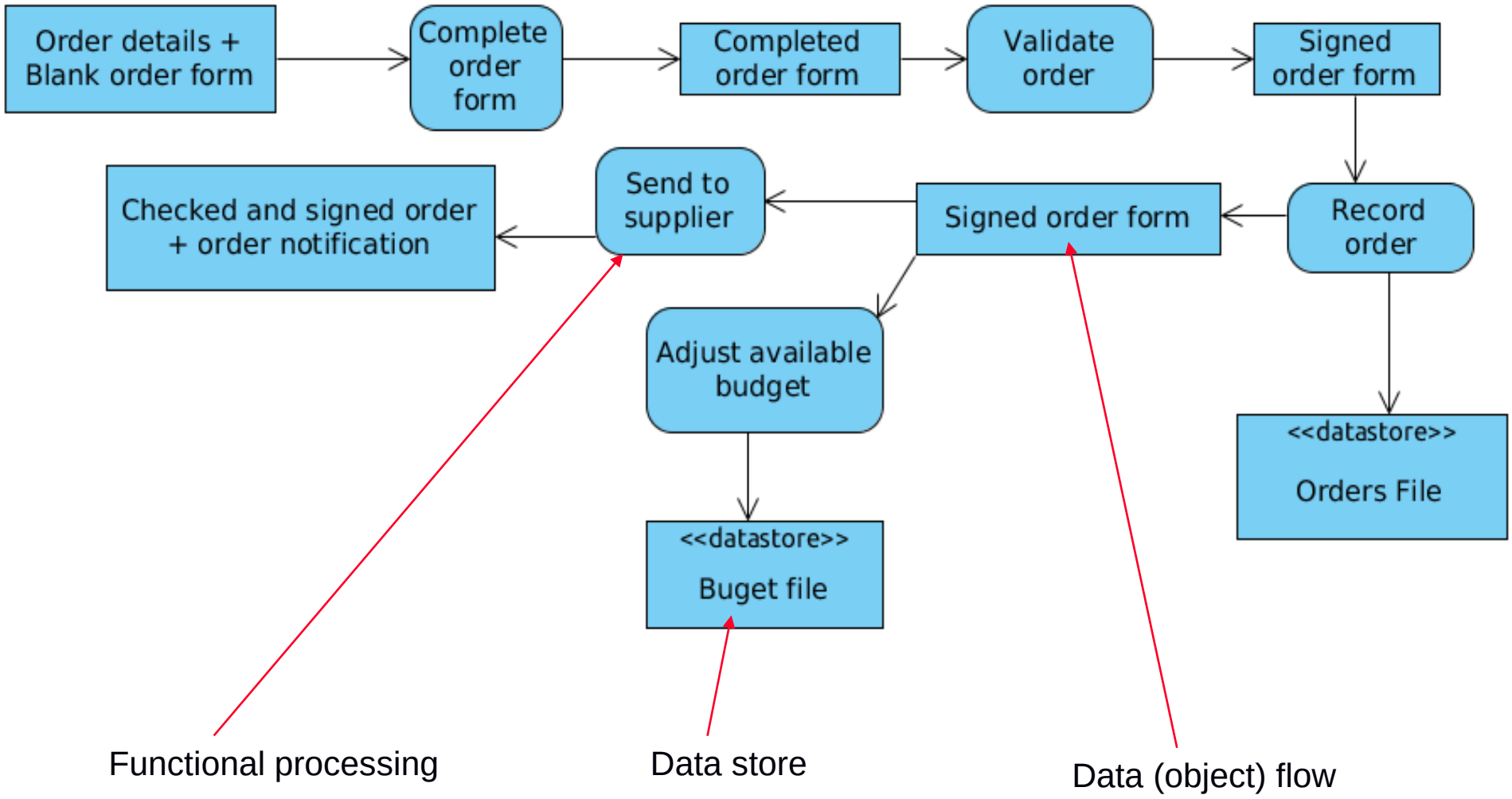
# Example DFD: Order processing DFD




Notation:

- Round rectangle – functional processing
- Rectangle – data store
- Labelled arrow – data flow.

# Example Activity Diagram: Order processing activity diagram



# Event-driven modelling

Model the behaviour of the system in response to external and internal events.

- Show the *system's responses to stimuli* (so are often used for modelling real-time systems).

Modelling elements:

- system ***states*** as *nodes*
- ***events*** as *edges* between these nodes.

When an event occurs, the system moves from one state to another = makes a ***state transition***.

# State machine diagram (SMD)

Describes a behavior or a behavioral feature.

Modelling elements:

- States
- Transitions connecting states
- Triggers for the transitions
- Activities performed in the execution of transitions
- Activities performed throughout the duration of each state

Obs. May express behavior on different levels of abstraction.

# State machine diagram (SMD)

State – describes the condition of an object expressed in terms of:

- object *attributes*
- *behavior* the object is engaged in.

Object state = cumulative result of object behaviour, defined by the values of its properties (attributes and relations).

Inside a state an object may:

- execute an activity
- wait for an event
- fulfill a condition.

# State machine diagram (SMD)

Formal: a state models a period of time in which an *invariant* condition holds true.

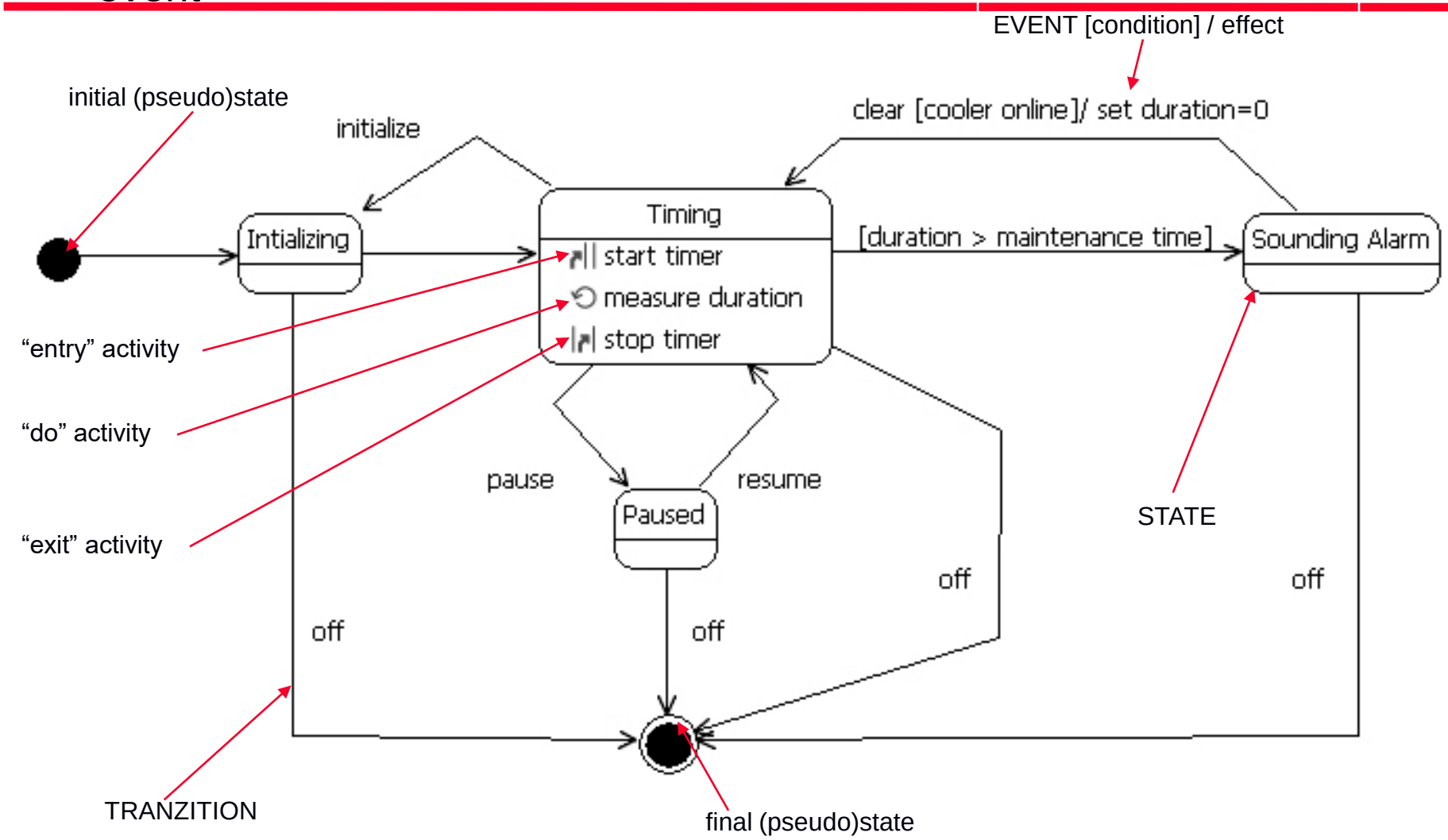
Examples of invariant conditions:

- set of attributes values
- condition of working on an activity
- condition of waiting for an event.

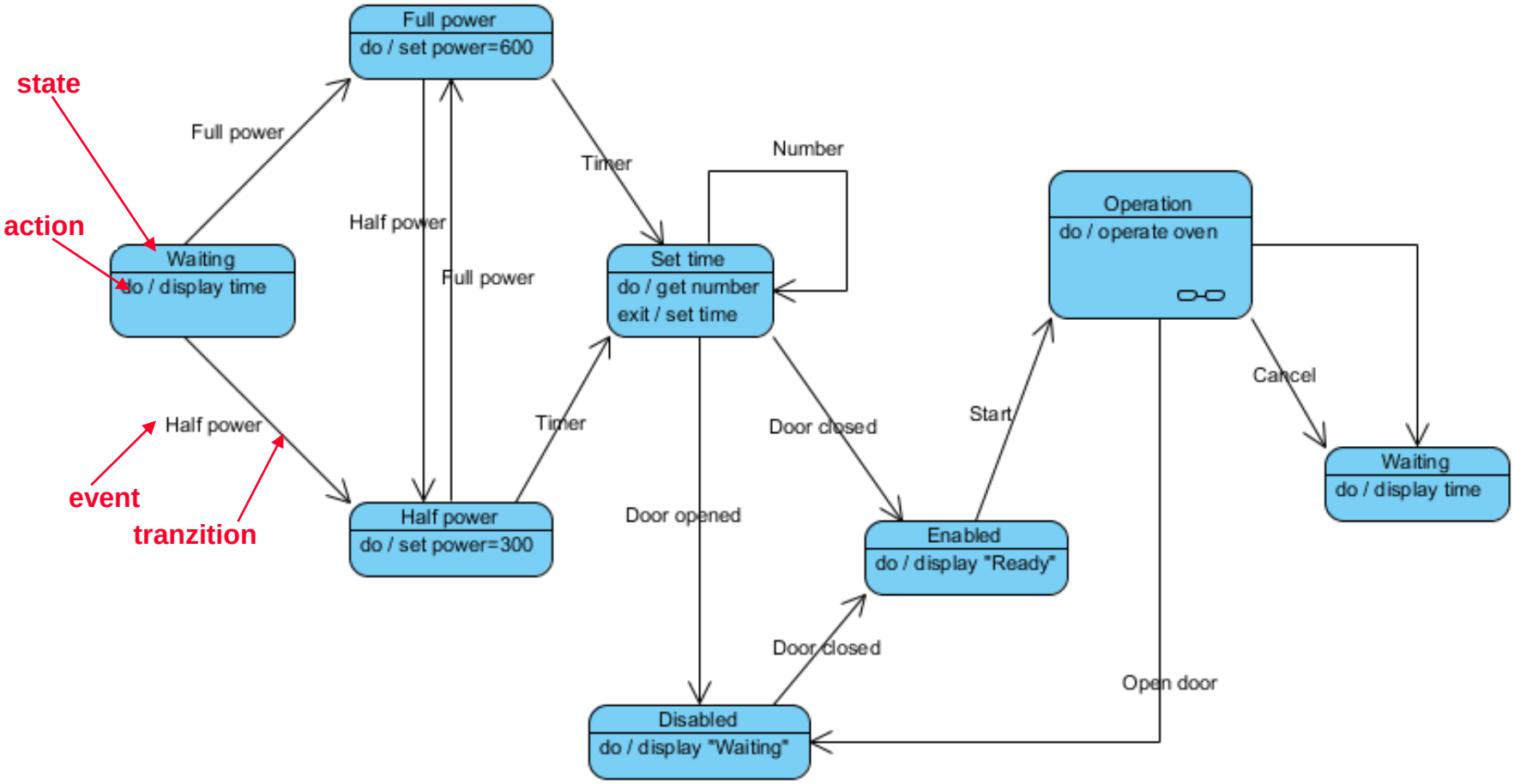
MAIN MODELLING ELEMENTS:

- state
- transition
- event

# State machine diagram (SMD)



# Example: Microwave oven model



# Example: Microwave oven state description

| State | Description |
|---|---|
| Waiting | The oven is waiting for input. The display shows the current time. |
| Half power | The oven power is set to 300 watts. The display shows ‘Half power’. |
| Full power | The oven power is set to 600 watts. The display shows ‘Full power’. |
| Set time | The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set. |
| Disabled | Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’. |
| Enabled | Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’. |
| Operation | Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding. |

# Example: Microwave oven stimuli

| Stimulus | Description |
|---|---|
| Half power | The user has pressed the half power button |
| Full power | The user has pressed the full power button |
| Timer | The user has pressed one of the timer buttons |
| Number | The user has pressed a numeric key |
| Door open | The oven door switch is not closed |
| Door closed | The oven door switch is closed |
| Start | The user has pressed the start button |
| Cancel | The user has pressed the cancel button |

# Structuring state models

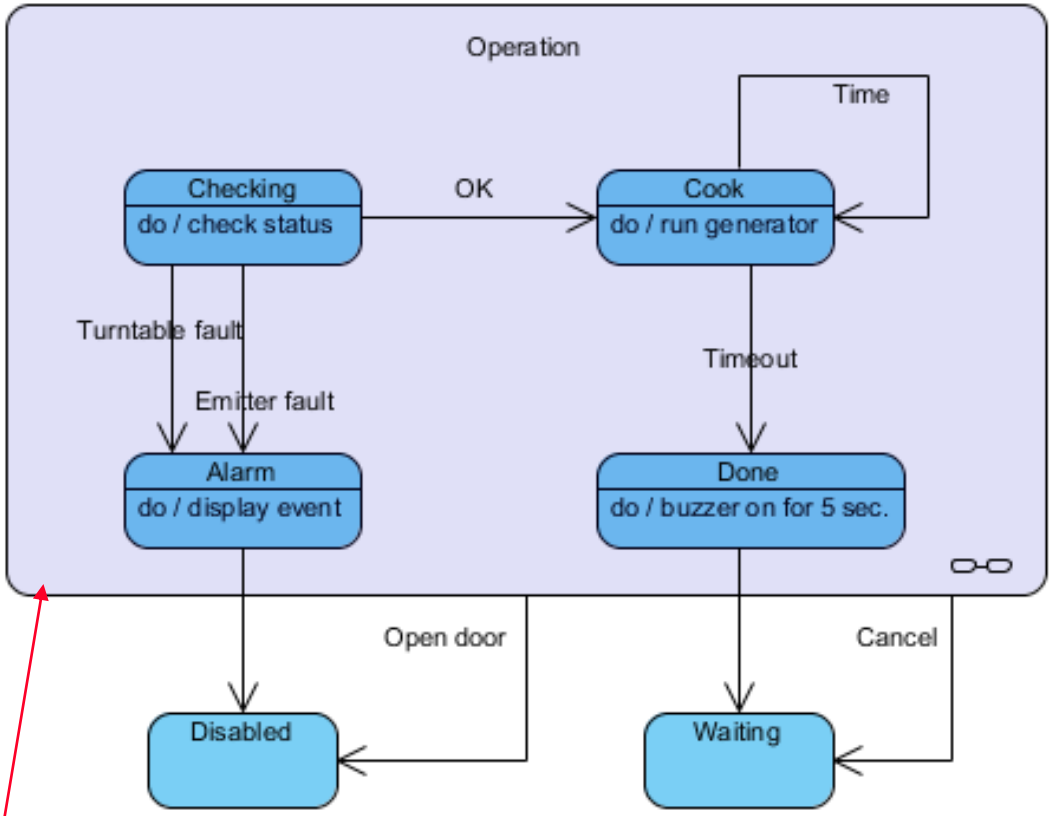
Necessary in large system models, with a big number of possible states.

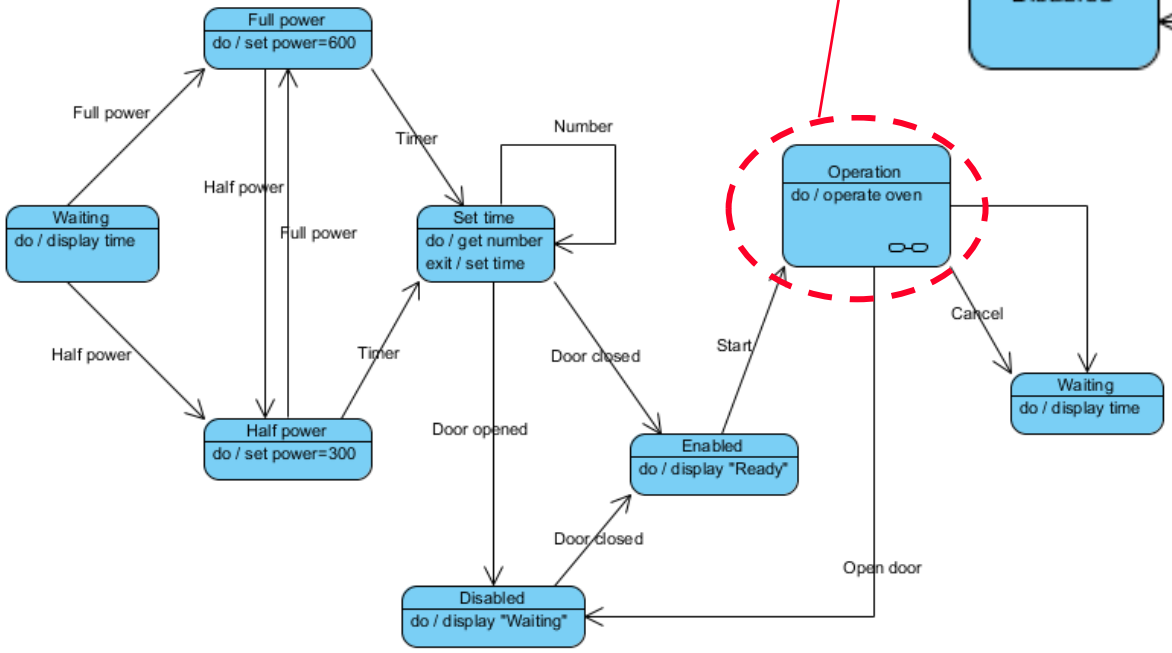
***Superstate***

- notion that encapsulates a number of separate states.
- looks like a single state on a high-level model.
- is expanded in more detail in a separate diagram.

Ex. “Operation” superstate.

Example:
"Operation"
superstate



# Topics covered

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# Semantic data models

Used to describe the ***logical structure of data*** processed by the system.

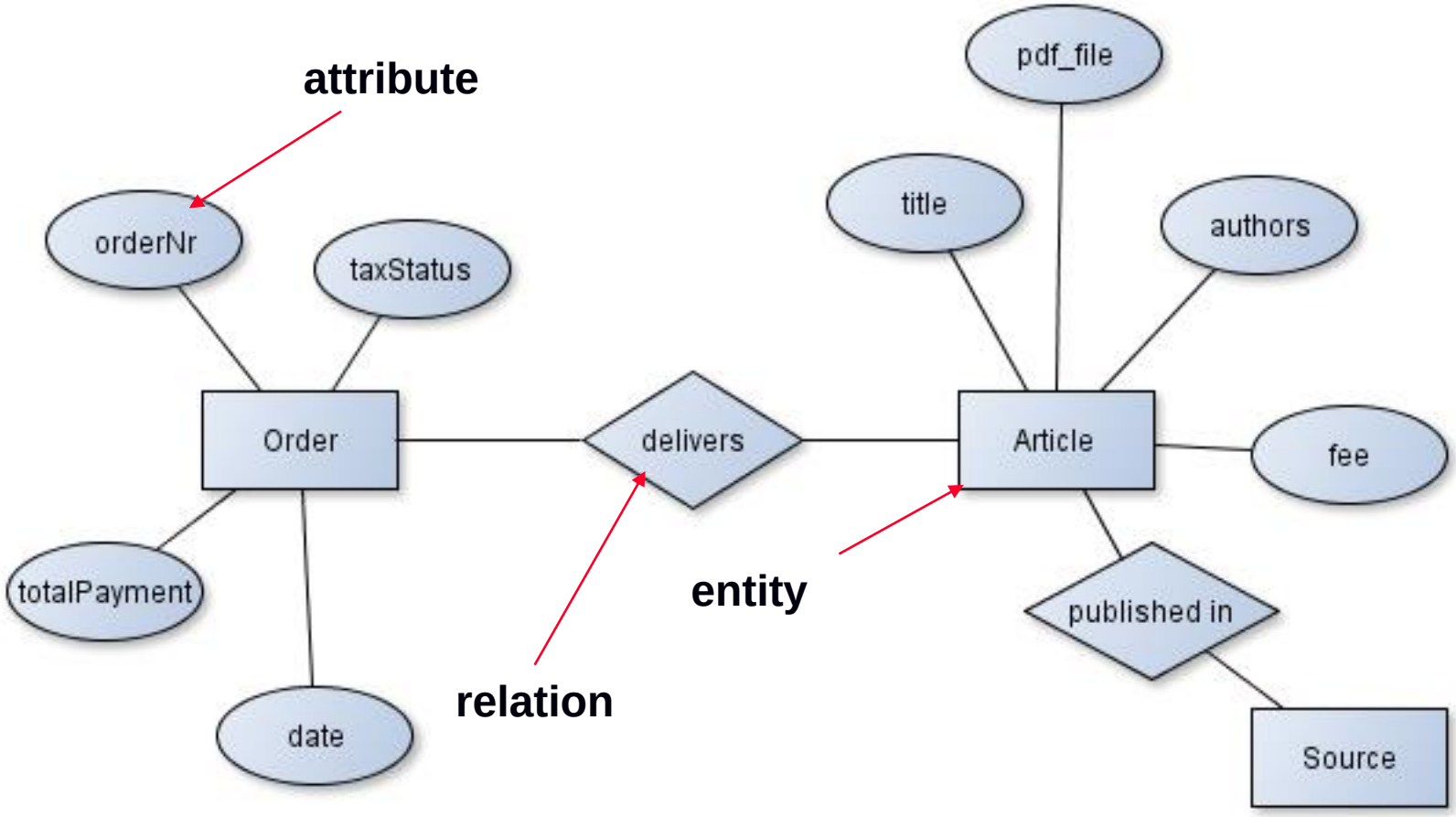
Notation:

- *entity-relation-attribute* (ERA) model - sets out the entities in the system, the relationships between these entities and the entity attributes.

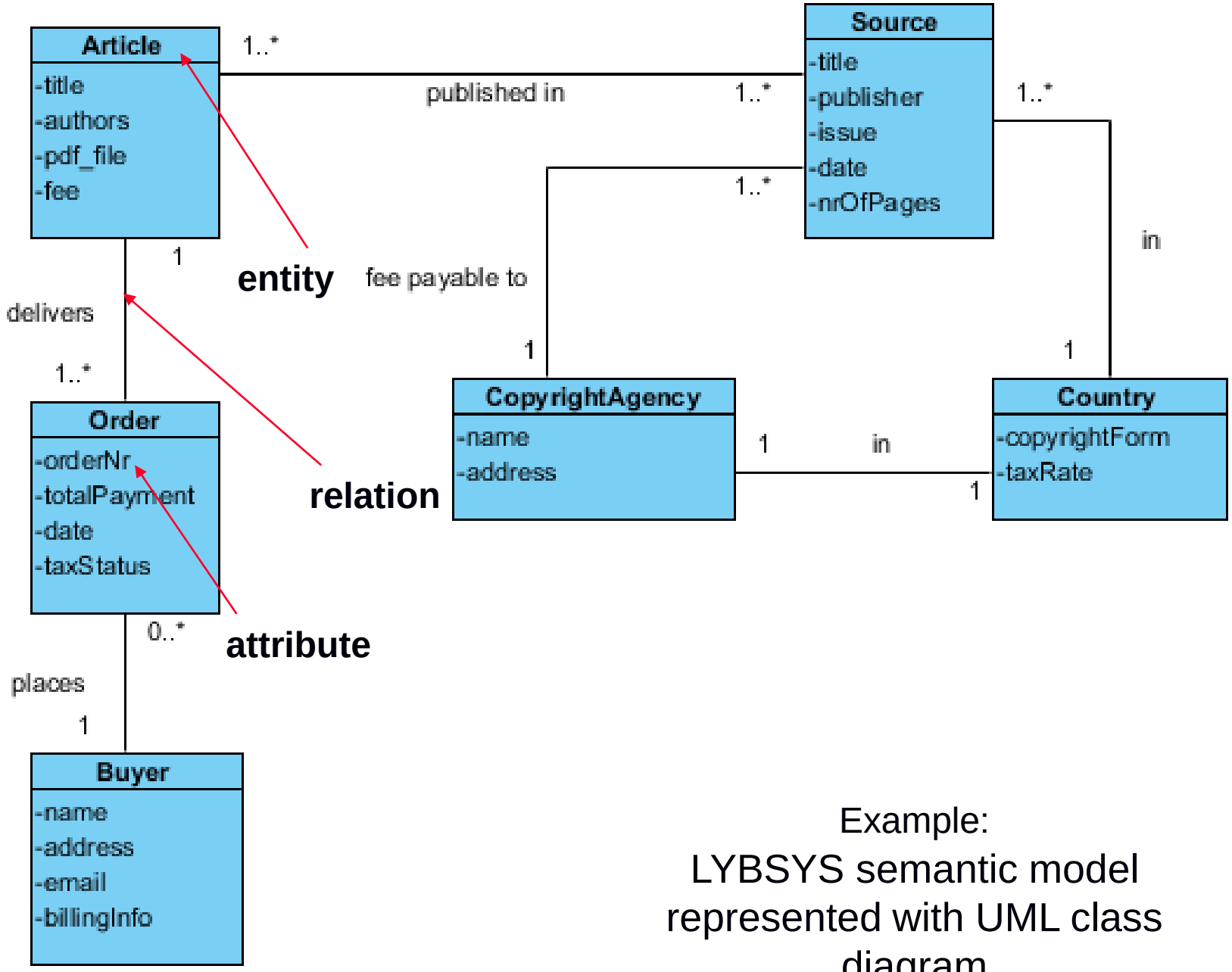
- *simplified* object *classes* (attributes, NO operations) and *associations* in a UML class diagram.

Complemented with details represented in *data dictionary*.

# Example: LYBSYS semantic model represented with ERA diagram




Excerpt from the LYBSYS semantic model.




Example:
LYBSYS semantic model represented with UML class diagram

# Data dictionaries

Data dictionaries are *lists of all of the names* used in the system models.

- *Descriptions* of the *entities*, *relationships, attributes* and *services* are included.
- Supplement diagrams with more detailed descriptions stored in a repository (*data dictionary*).
  - Advantages
    - Support name management and avoid duplication;
    - Store of organisational knowledge linking analysis, design and implementation;
  - Many CASE workbenches support data dictionaries.

# Example: LIBSYS data dictionary entries

| Name | Description | Type | Date |
|---|---|---|---|
| Article | Details of the published article that may be ordered by people using LIBSYS. | Entity | 03.03.13 |
| authors | The names of the authors of the article who may be due a share of the fee. | Attribute | 03.03.13 |
| Buyer | The person or organisation that orders a copy of the article. | Entity | 03.03.13 |
| fee-payable-to | A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee. | Relation | 02.03.13 |
| Address (Buyer) | The address of the buyer. This is used to any paper billing information that is required. | Attribute | 04.03.13 |

# Formative evaluation

1. Map the type of UML diagram to what can be represented with it.

https://forms.gle/JWP6H5fGFpqAhYn29

Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# User interface prototyping

- *Essential* in requirements analysis

*Aim* : to allow users to gain direct experience with the interface.

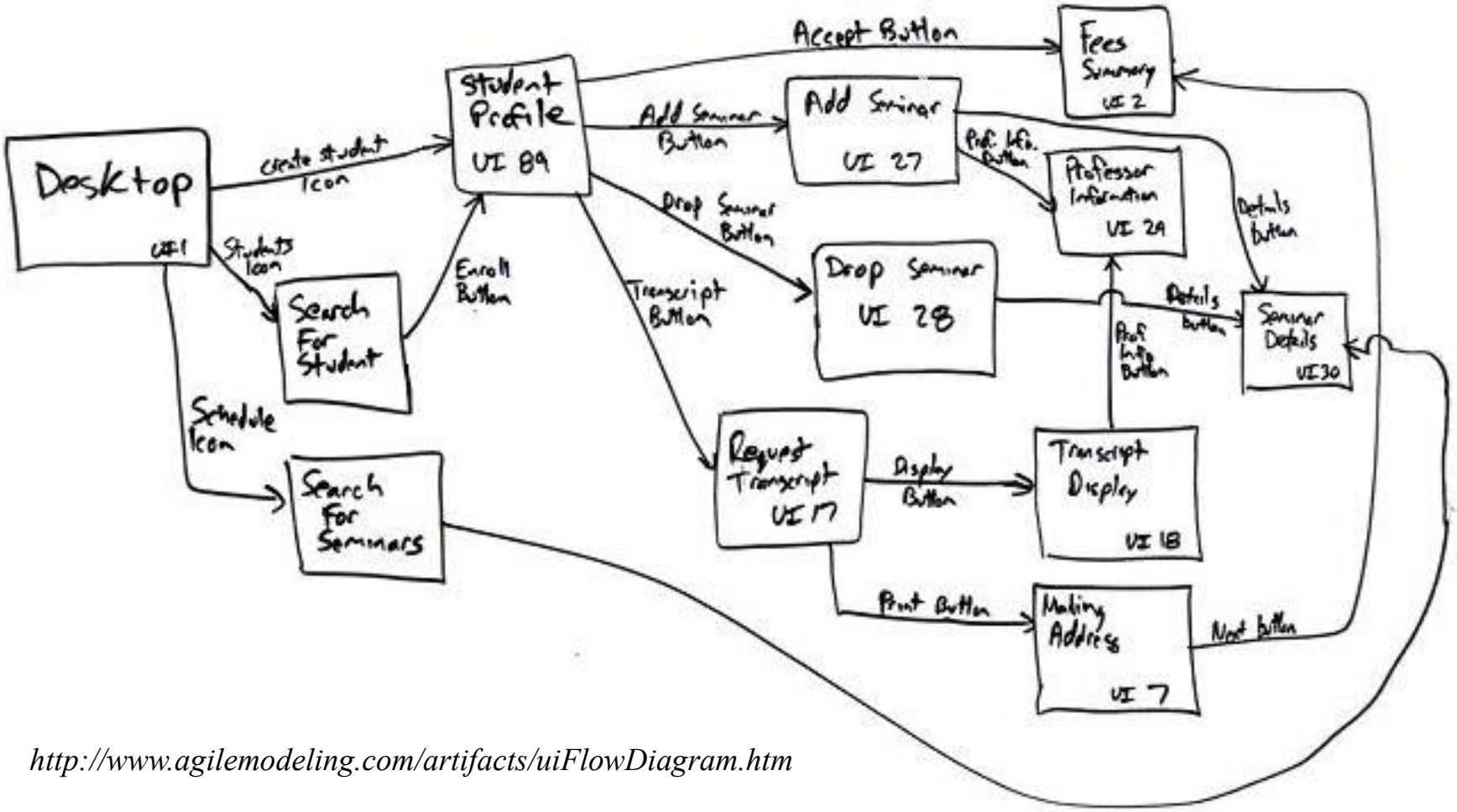
- Without such *direct user experience*, it is impossible to judge the *usability* of an interface.

- Prototyping may be a two-stage process:
  - Early in the process, paper/whiteboard prototypes may be used;
  - The design is then *refined*, and increasingly sophisticated automated prototypes are then developed.

List of tools at: http://c2.com/cgi/wiki?GuiPrototypingTools

# User interface prototyping

Focus on the main use cases and their interfaces.



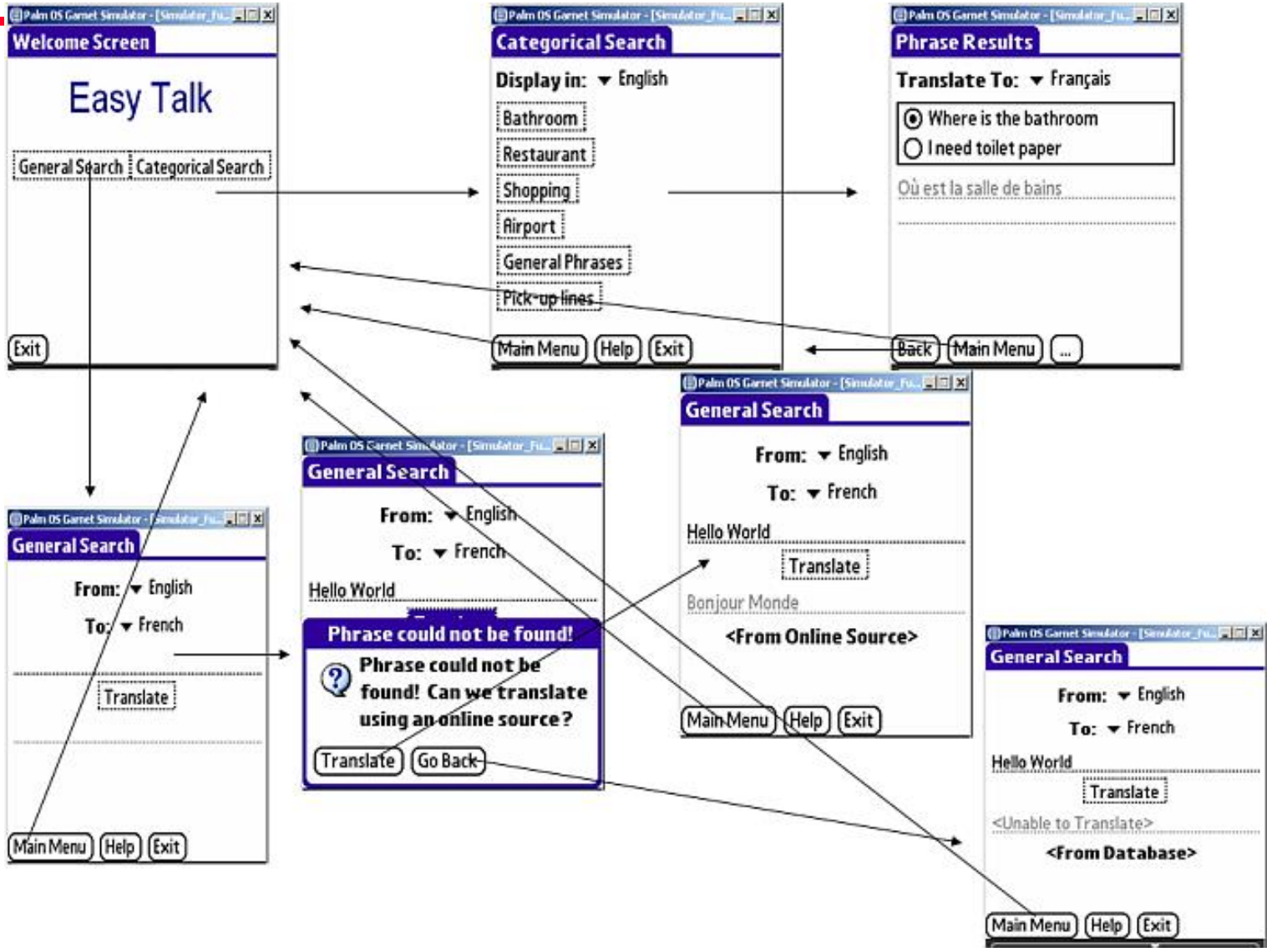

*http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm*

Aspects analyzed:

- Visual looks and display
- Interaction with the user

and screens flow

# User interface prototyping



Context models

Interaction models

Structural models

Behavioral models

Data models

UI prototype

Model-driven engineering

# MDE

- Model-driven engineering (MDE) - approach to software development where *models* (rather than programs) are the *principal outputs of the development process*.
- Programs - *generated automatically* from the models.
- The fundamental notion behind model-driven engineering is that *completely automated transformation* of models to code should be possible.
- 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*.

# MDE - Discussion

- 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
  - Some abstractions in a model are not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

# MDA

***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.

# Model types

*Computation independent model (CIM)*

- Models the important *domain abstractions* used in a system. CIMs are sometimes called *domain models*.

*Platform independent model (PIM)*

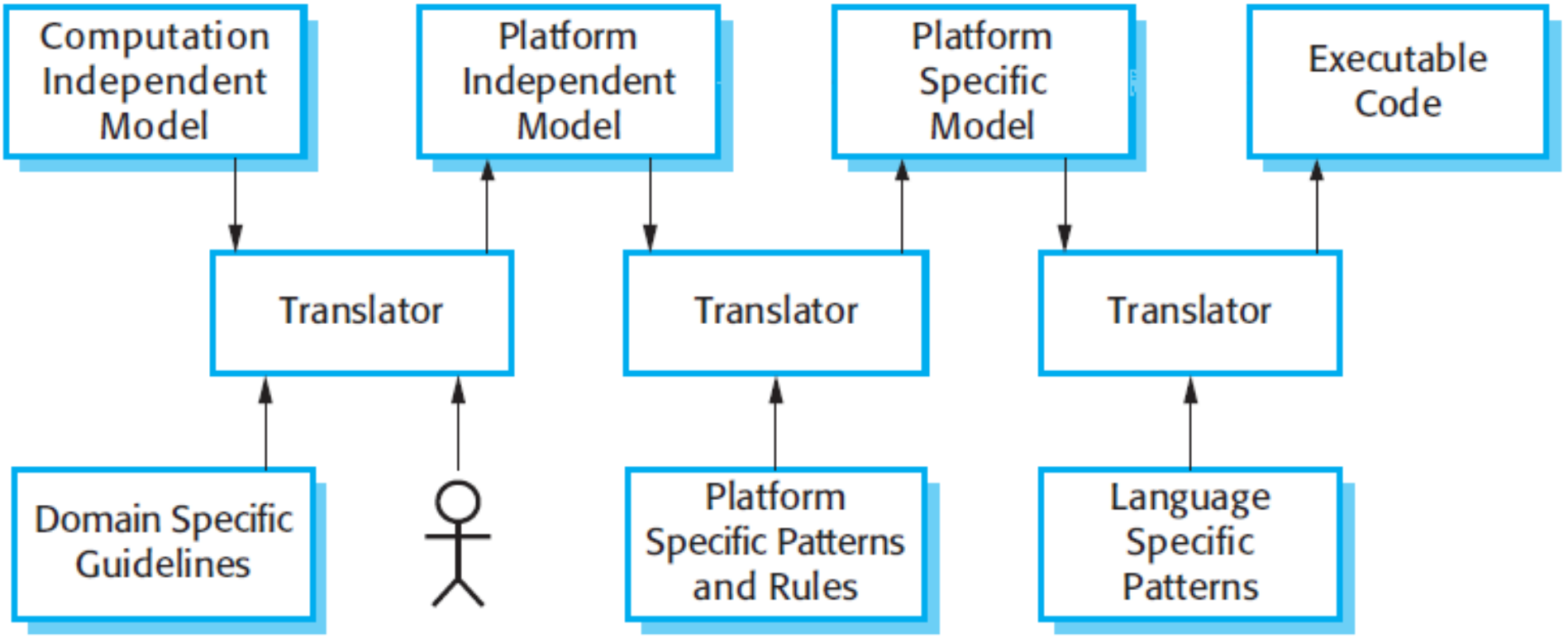
- Models the operation of the system without reference to its implementation.
- Usually described using UML models that show the *static system structure* and how it *responds to external and internal events*.

*Platform specific models (PSM)*

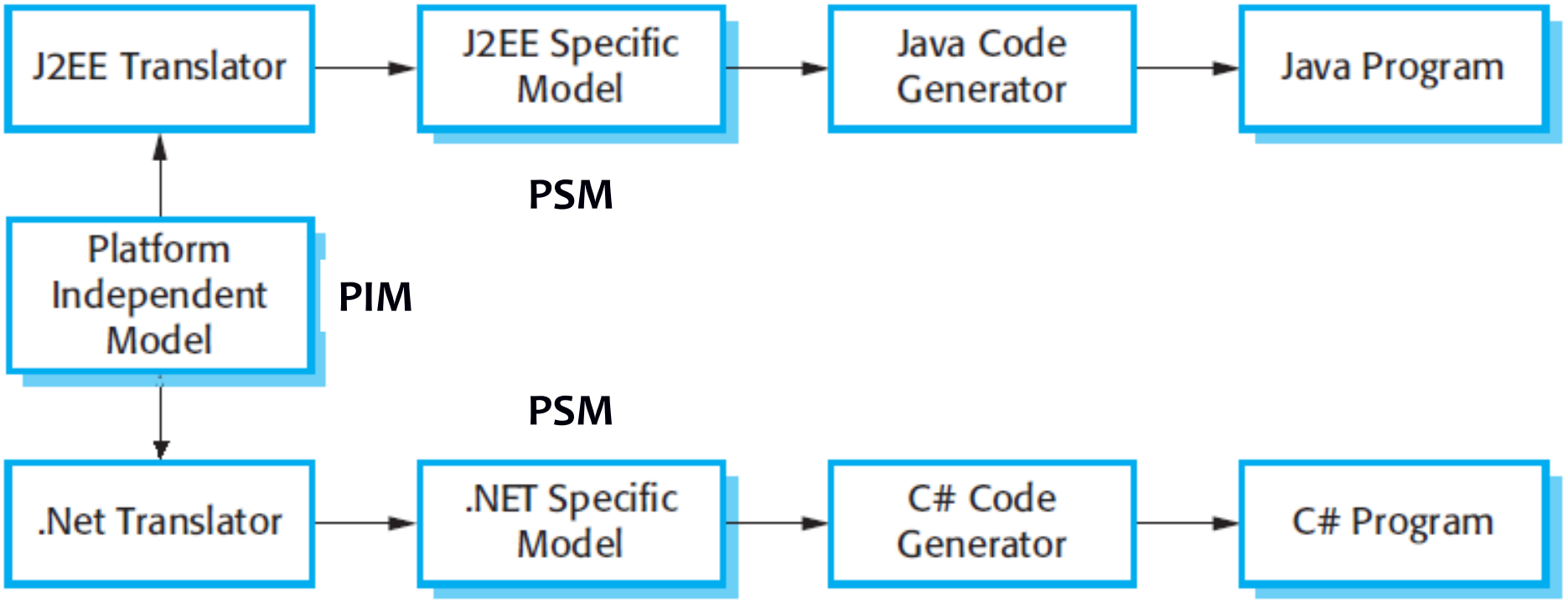
- Transformations of the platform-independent model, with a separate PSM for each application platform.

Obs. In principle, there may be more layers of PSM, with each layer adding some platform-specific detail.

# MDA transformations



# Multiple platform-specific models



# Formative evaluation

1. Why GUI prototyping is important ?
2. What is the importance of MDE (Model Driven Engineering) ?
3. Shortly describe the types of models in MDA (Model Driven Architecture). Specify their relationships.

https://forms.gle/Ktjbpby1FGTR8MpSA

# Key points

- A *model* is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- *Context models* show how a system that is being modeled is positioned in an environment with other systems and processes.
- *Use case diagrams* are used to describe the functions exposed by the system
- S*equence diagrams* are used to describe interactions between a system and external actors and between system objects.

# Key points

- *Structural models* show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations. Component diagrams are used to describe the system structure at runtime.

- *Data models* show the logical structure of data processed by the system. ERD or simplified class diagrams may be used, complemented with data dictionaries.

# Key points

- *Behavioral models* are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- *Activity diagrams* may be used to model the processing of data, where each activity represents one process step.
- *State diagrams* are used to model a system's behavior in response to internal or external events.

# Key points

- *UI prototyping* is essential in requirements analysis. It may be a two stages process.

- *Model-driven engineering* is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.