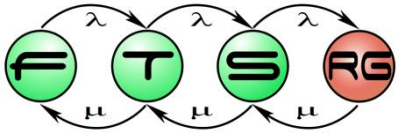
Requirements Engineering and Domain Modeling

Ákos Horváth, Dániel Varró

Model Driven Software Development

Lecture 2



Requirements Engineering by Use Case Analysis

Requirements analysis

- **Requirements engineering (RE)** is the process of *identifying*, *organizing*, and *documenting* the continuously changing requirements of a project
- **Requirement**: a condition or capability to which the system must conform
- An *early identification of requirements* is critical for the quality of the system under design
  - consistent?, complete? unambiguous?
- Gathering of requirements is a very complex engineering task
  - „Requirements do not come from the air”
  - an iterative refinement process with regular control

Definition of requirements analysis

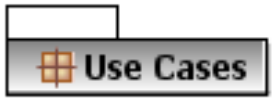
Identification of

- Goals: the objectives of the system
  - Why do we need the SW?
- Services („operationalization”)
  - What functionality do we need to design?
- Constraints
  - Restrictions of the design process (e.g. cost, deadlines)
- Responsibilities to each requirement (SW vs. human)

Categorization of RE

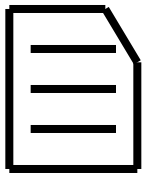
- **High-level (System-level) requirements**
  - Feature (FEAT): high-level product requirement from the customer's point of view
  - Stakeholder needs (NEED):
  - The agreement between the customer and the system analyst documented in the vision document
- **Low-level (Software-level) requirements**
  - Software requirements
  - **Actor**: someone or something outside the system that interacts with the system
  - **Use case** (UC): a functional requirement
  - **Supplementary requirement** (SUPL): a non-functional requirement

Main documents of RE



- **Use Case model**
  - Actors, Use Cases, Subsystems
  - Scenarios as workflow

- Architectural description:
  Detailed textual description of
  - Use cases
  - Scenarios



- Glossary (Szójegyzék)
  - Precise definition of common terms

- GUI prototype
  - Communication with end users

Sample Requirements

Requirements of a Table Game Championship Manager System

Verbal Requirements

- Design a system for organizing championships of table games (chess, go, backgammon, etc.)
- Requirements:
  - A player should register and log in to the system before using it.
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the championship can be started by the organizer.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.

Passive sentences should be avoided!

Verbal Requirements

- Design a system for organizing championships of table games (chess, go, backgammon, etc.)
- Requirements:
  - A player should register and log in to the system before using it.
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.

Verbal Requirements (cont.)

- Requirements (cont.):
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship
  - The actual game is played between existing clients, which is outside the scope of the modelled system.
  - Both players should report the result and the moves after each game using a web form. A win scores 1 point, a draw ½, and a loss 0.
  - If players report contradicting results, the organizer should judge who the winner is. The organizers penalizes the cheating player by a 1 point penalty.
  - When all games are finished, the organizer should close the championship by announcing the winner. Then he or she may start organizing a new championship.

Missing Requirements

- A game should be finished within a given deadline (time limit).
- If none of the two players have reported the result within this deadline,
  then both players are considered to be losers.
- If only one player has reported the result, then his (or her) version is considered to be the official result.

- NOTE: New requirements will emerge during UC analysis (especially when detailing UCs).
  An iterative requirements engineering process is highly recommended.

Best Practice: Requirements

- A requirement should contain
  - a short description
  - a stand-alone sentence / paragraph
- English:
  - Avoid passive sentences
  - Use the following auxiliaries:
    - Positive: shall/must, should, may,
    - Negative: must not, may not
- Detail them with parameters:
  - Priority, Status, Difficulty, Responsibility, Risk

Elements of Use Case Diagrams by Example

Definition of Use Cases

- **Use cases (használati eset)** capture the functional requirements of a system
- UCs describe
  - the typical interactions
  - between the users of a system and
  - the system itself,
  - by providing a narrative of how a system is used
- A set of scenarios tied together by a common user goal
- Verb + Noun (Unique)!

M. Fowler: UML Distilled.
3rd Edition. Addison-Wesley

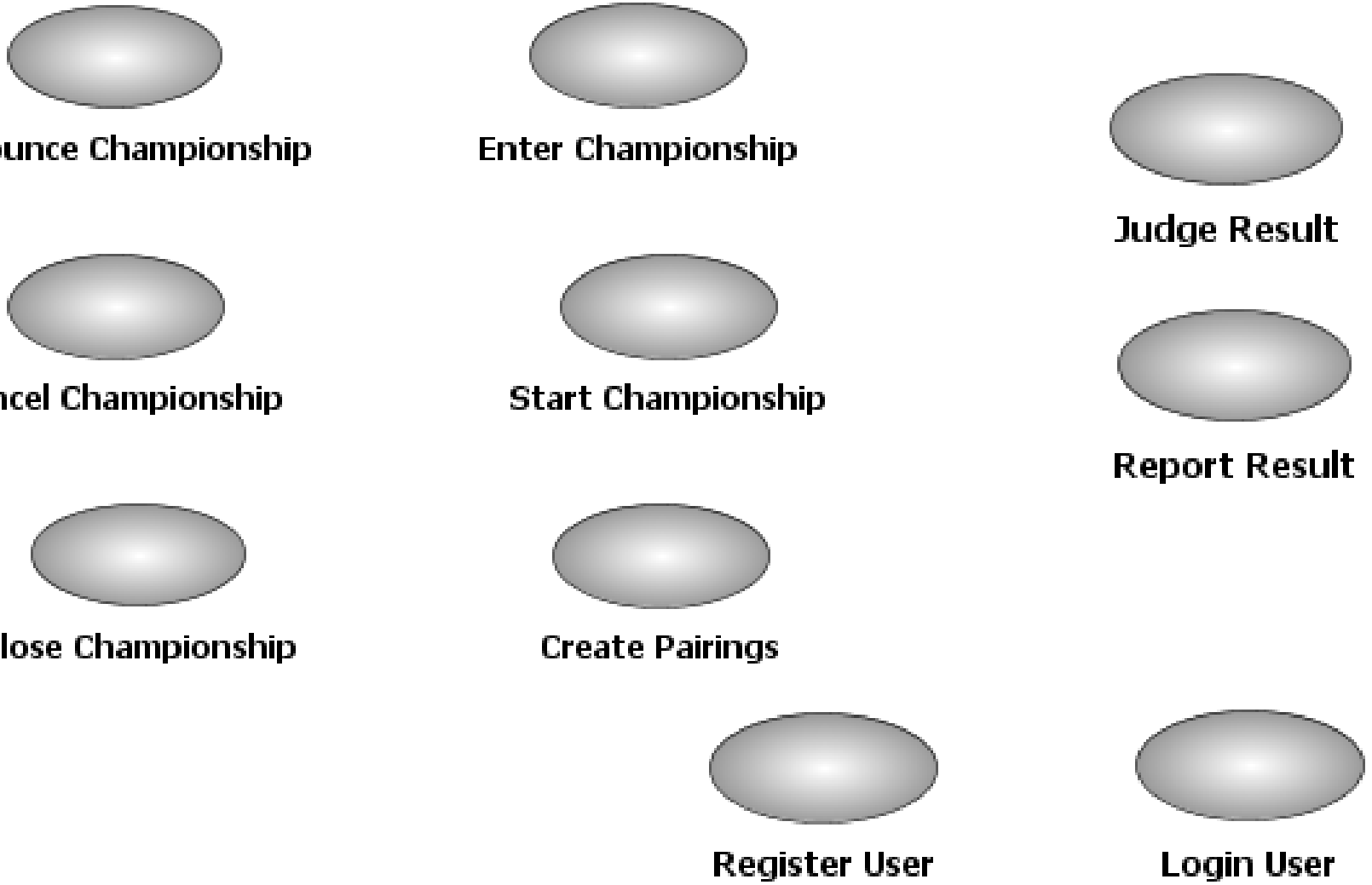
From Verbal Requirements to Use Cases

- Requirements:
  - Each registered player may announce a championship.
  - A player should register and log in to the system before using it.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.

Verbal Requirements (cont.)

- Requirements (cont.):
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship
  - The actual game is played between existing clients, which is outside the scope of the system system.
  - Both players should report the result and the moves after each game using a web form. A win scores 1 point, a draw ½, and a loss 0.
  - If players report contradicting results, the organizer should judge who is the winner. The organizers penalizes the cheating player by a 1 point penalty.
  - When all games are finished, the organizer should close the championship by announcing the winner. Then he or she may start organizing a new championship.

(Initial) Collection of Use Cases



Definition of Actors

- **Actor (aktor)** is a **role** that a user plays with respect to the system.
  - Primary actor: calls the system to deliver a service
  - Secondary actor: the system communicates with them while carrying out the service
- Relationship of UCs and Actors
  - A single actor may perform many use cases;
  - A use case may have several actors performing it.
- One person may act as more than one actor,
  - Example: A person may organize a championship and may participate in another
- An actor is outside the boundary of the system

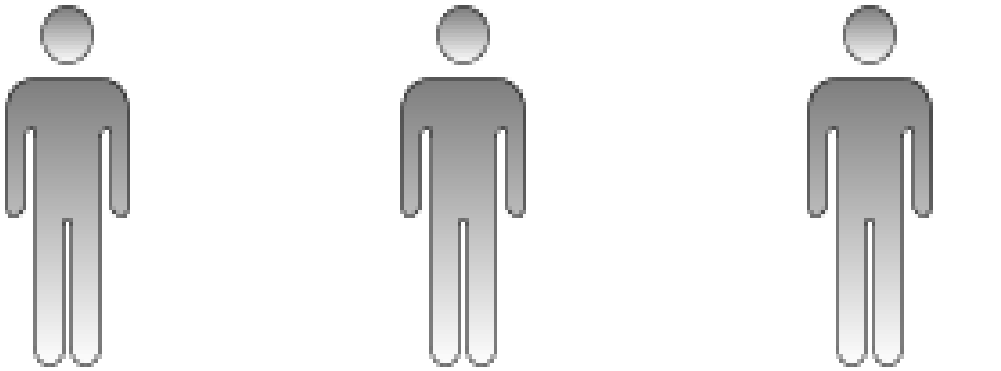
From Verbal Requirements to Use Cases

- Requirements:
  - Each registered player may announce a championship.
  - A player should register and log in to the system before using it.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.

Verbal Requirements (cont.)

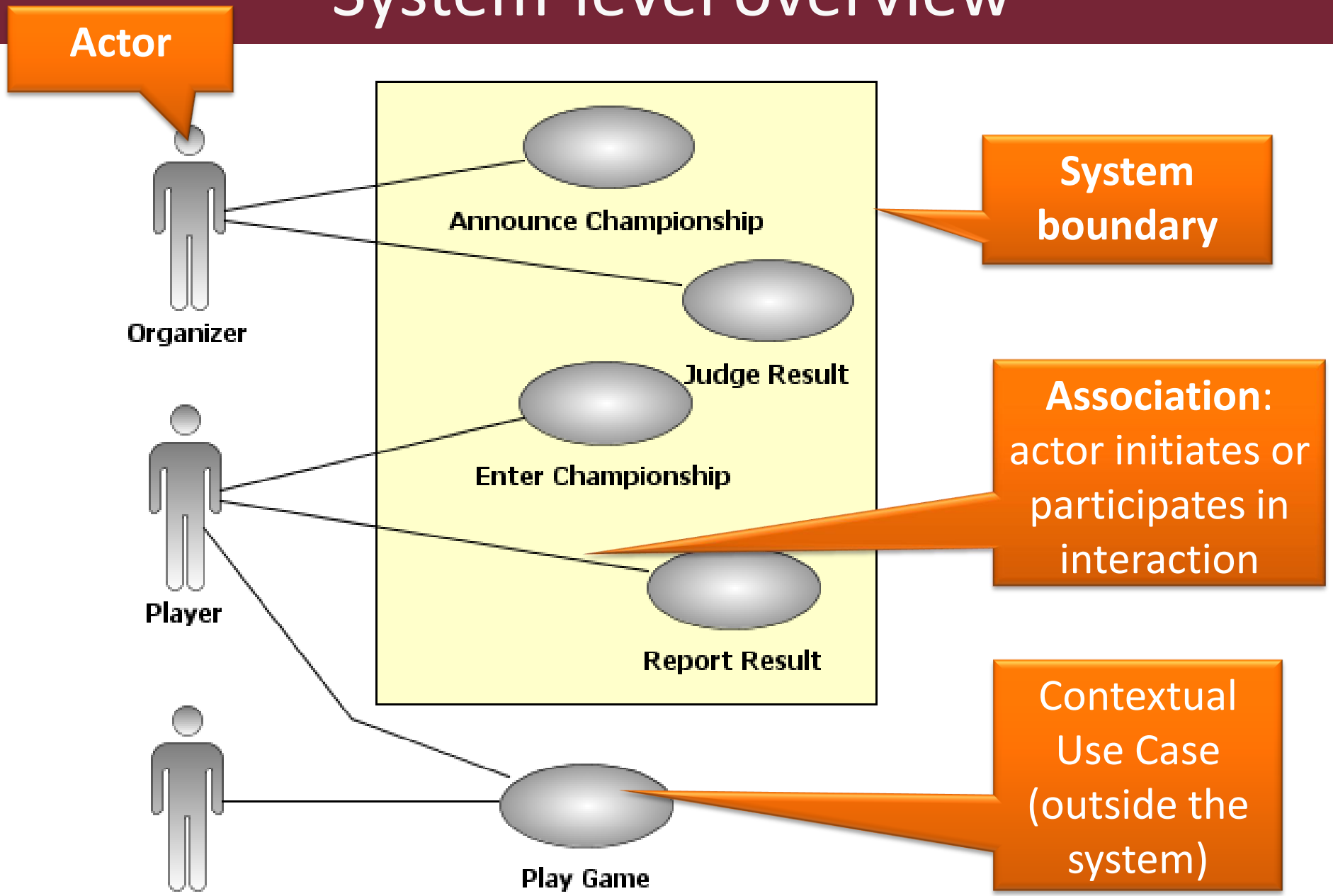
- Requirements (cont.):
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship
  - The actual game is played between existing clients, which is outside the scope of the system system.
  - Both players should report the result and the moves after each game using a web form. A win scores 1 point, a draw ½, and a loss 0.
  - If players report contradicting results, the organizer should judge who is the winner. The organizers penalizes the cheating player by a 1 point penalty.
  - When all games are finished, the organizer should close the championship by announcing the winner. Then he or she may start organizing a new championship.

(Initial) Collection of Actors

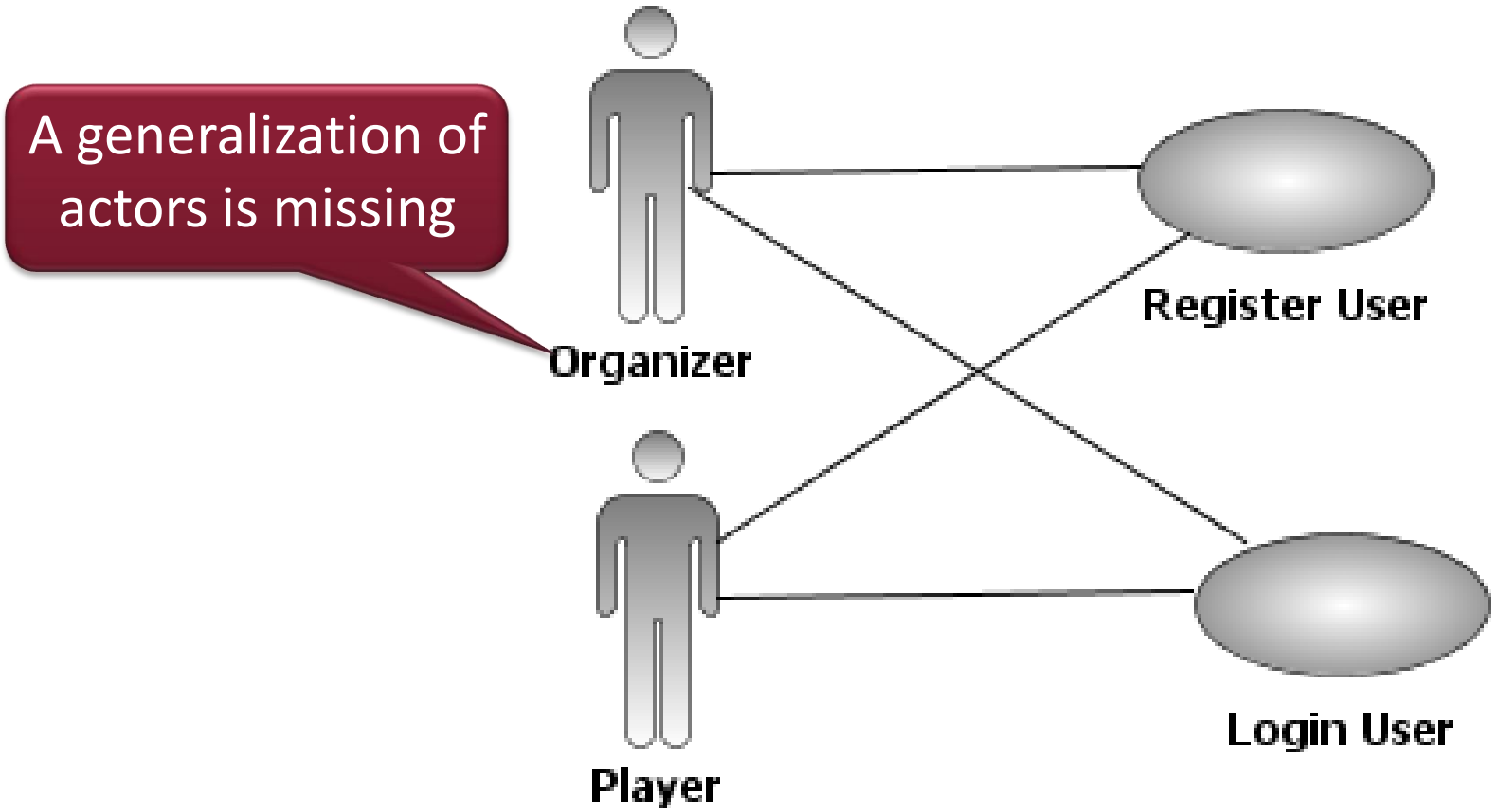


Relations between UCs and Actors

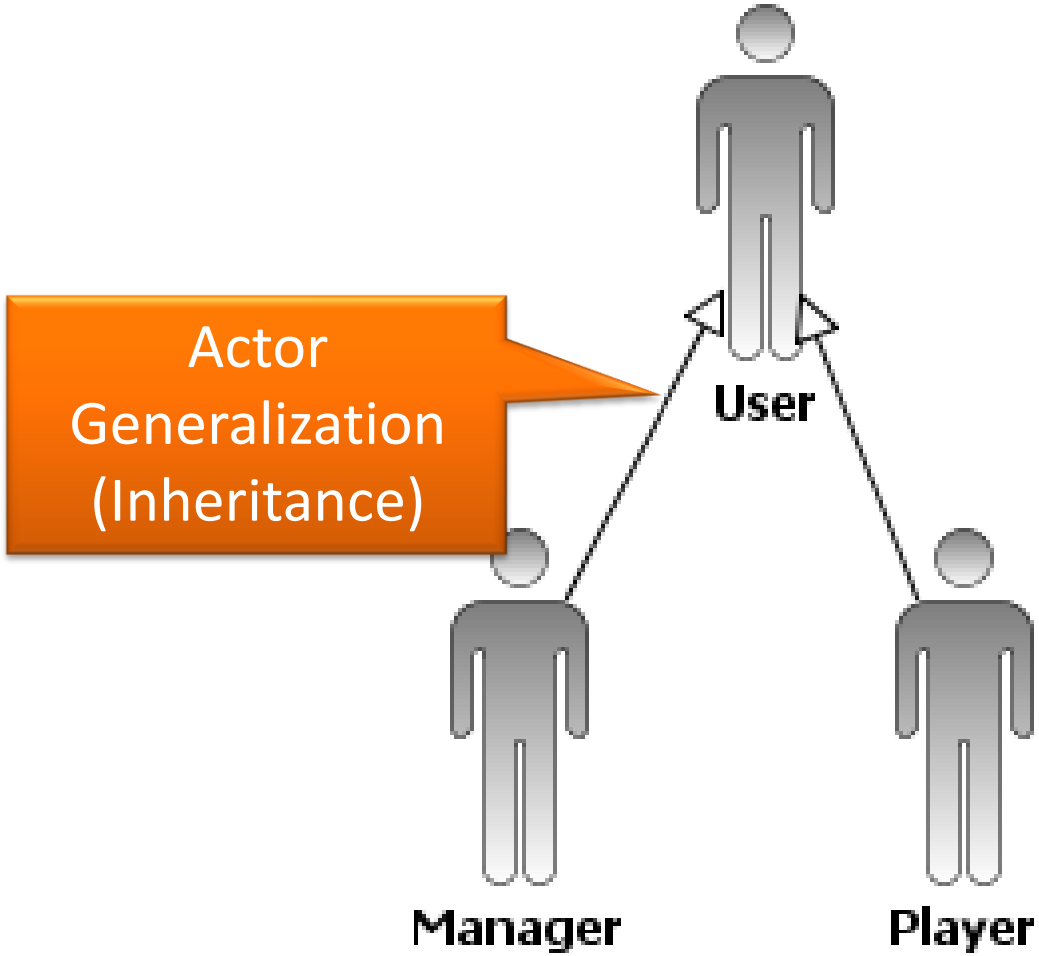
System-level overview



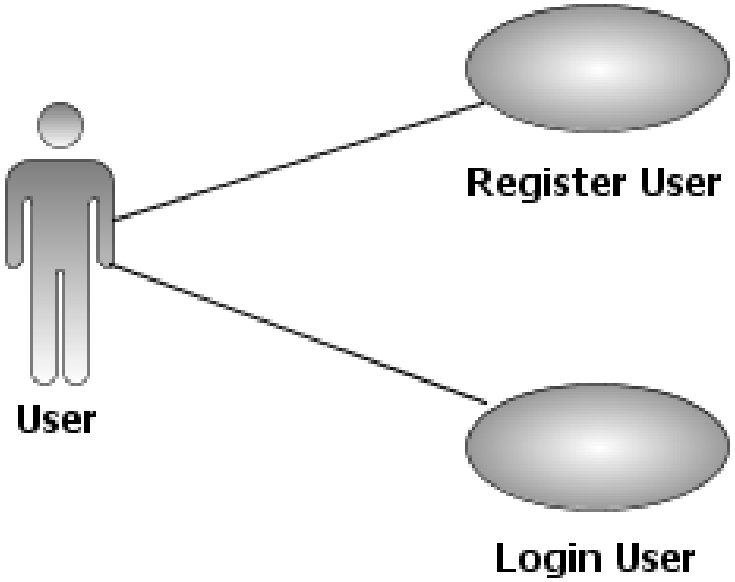
Anti-pattern: UC diagrams



Overview of Actors

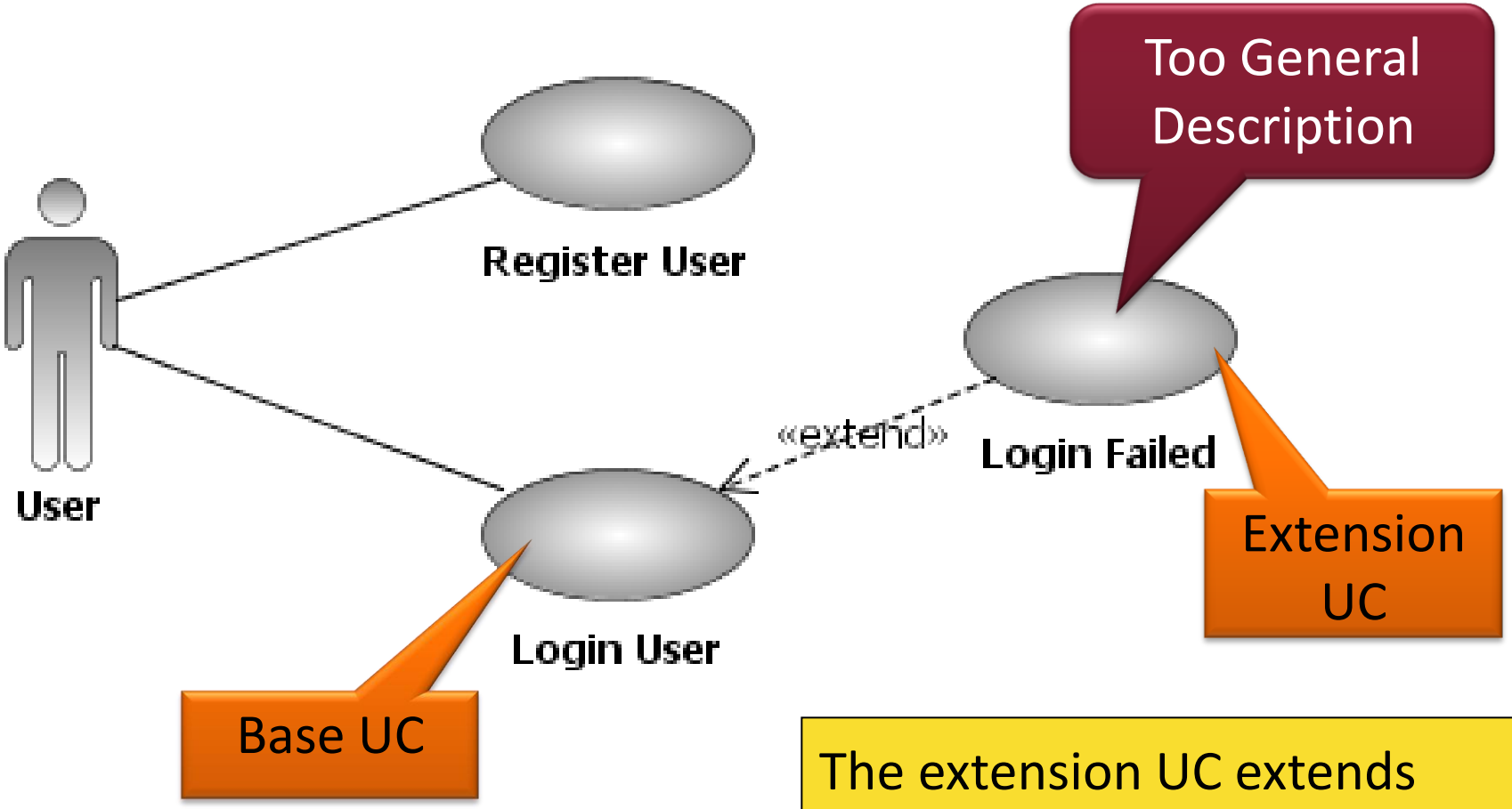


User Management



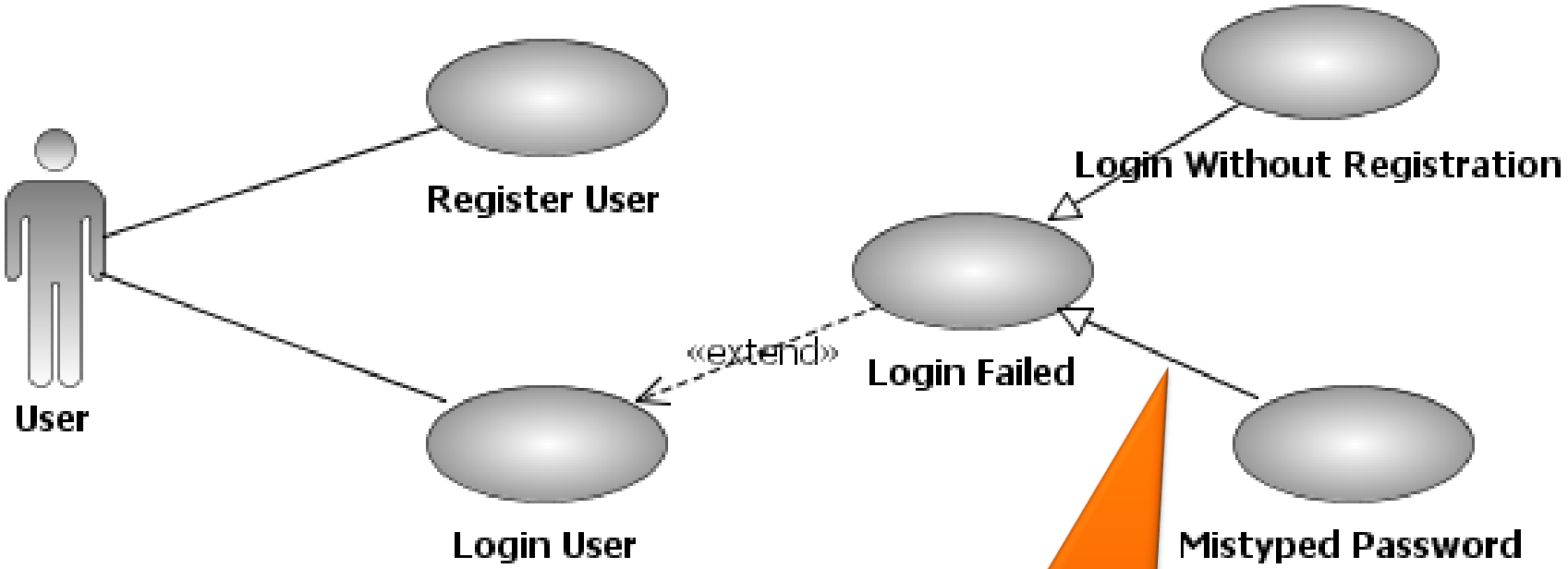
What happens if
- the user's password is incorrect?
- a user is not registered, but attempts to login?

Extend relationship



The extension UC extends core functionality by handling unusual (exceptional) situation

Refinement of Use Cases



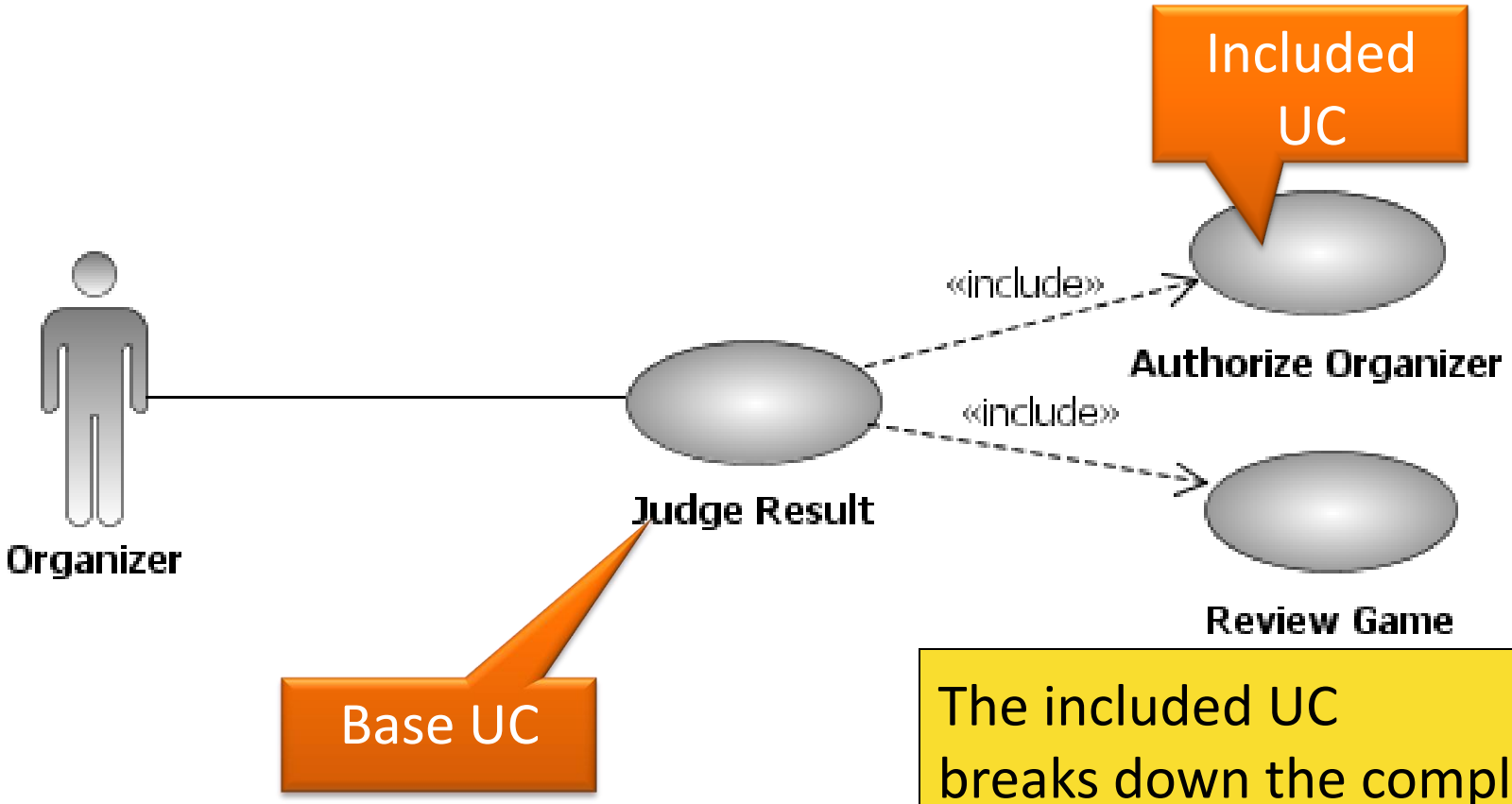
Use Case Generalization (Inheritance)

How to handle complex functionality?

Judge result =

- Check if the organizer is the judge
- Analyze the game
- Decide on result
- Report the result

Include relationship



The included UC breaks down the complex core functionality into more elementary steps

Summary: UC Relations

- Association (Asszociáció)
  - actor – use case
  - the actor initiates (or participates) the use of the system
- Extend (Bővítés)
  - use case – use case
  - a UC may be extended by another UC (typically solutions for exceptional situations)

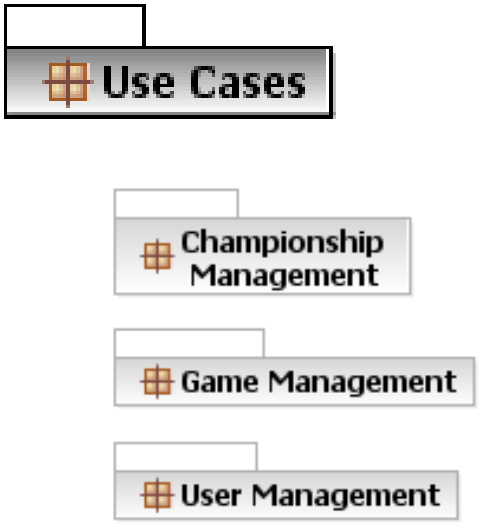
Summary: UC Relations

- Generalization (Általánosítás)
  - actor - actor
  - use case – use case
  - a UC or actor is more general / specific than another UC or actor
- Include (Beszúrás)
  - use case – use case
  - a complex step is divided into elementary steps
  - a functionality is used in multiple UCs

Best practices of UC analysis

Best practices: Grouping

- Grouping UCs
  - Identify functional building blocks
  - Group them into packages
  - NOTE: related by functionality, NOT by role
- Grouping actors:
  - Keep actors in a package within the subsystem they exclusively belong
  - Global actors: in top-most package



Best practices: Naming and arrangement

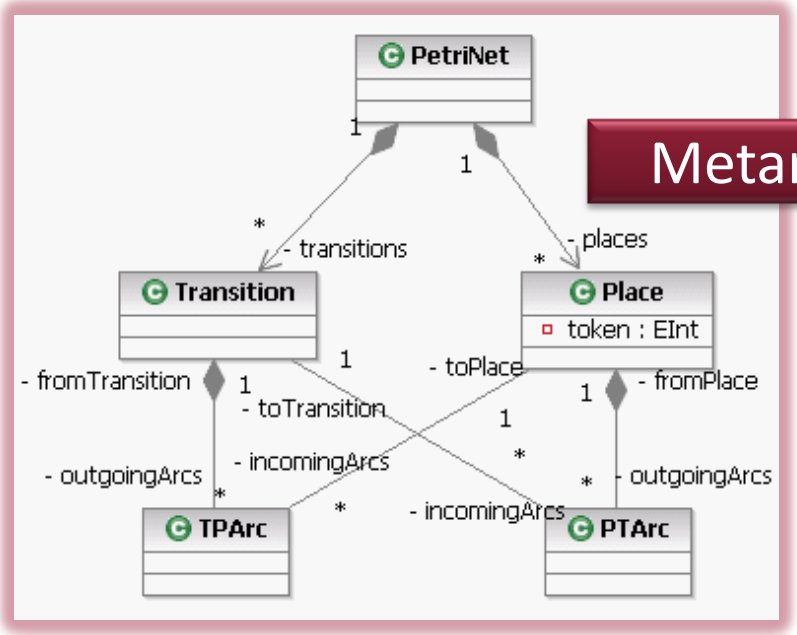
- Actors
  - Name actors according to their roles and avoid using job titles
  - Divide complex roles into multiple actors
  - Start the diagram by placing the most important actor in the top left corner
- Use Cases
  - Use domain specific verbs for UCs
  - Avoid technical descriptions – UCs are frequently for non-technical reader
- Relationships
  - Avoid crossing or curved lines when drawing relations
  - Use <<extend>> and <<include>> relations „lightly”
  - Place them into the appropriate functional block

Main guideline:
UC diagrams
should be SIMPLE

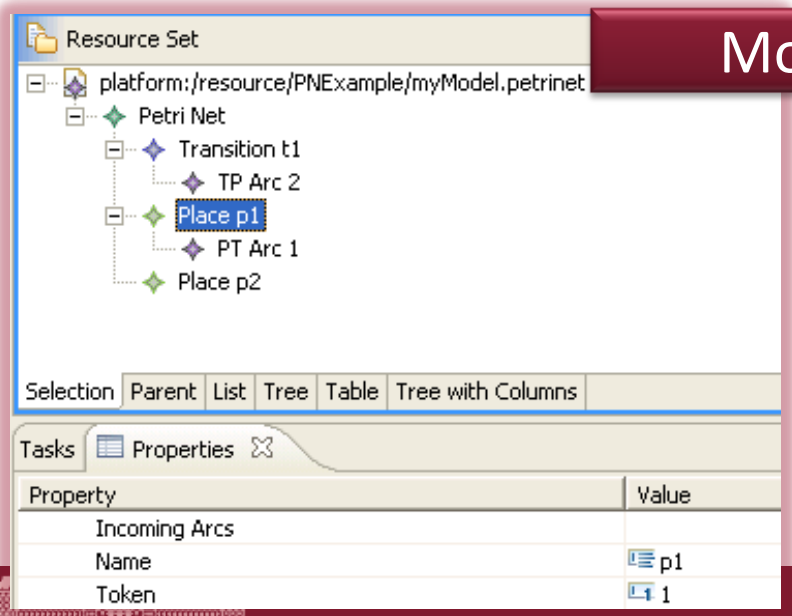
Domain Modeling

(Metamodeling)

Metamodel: Specify Concepts an Appl. Domain



Metamodel



Model

- Metamodel:
  - Precise specification of domain concepts
  - A language for defining the abstract syntax of a DSM
- Goal: to define...
  - Basic concepts
  - Relations between concepts
  - Attributes of concepts
  - Abstraction / refinement (Taxonomy, Ontology) between model elements
  - Aggregation
  - Multiplicity restrictions
  - Derived features

Metamodels and instance models

Reference / Association

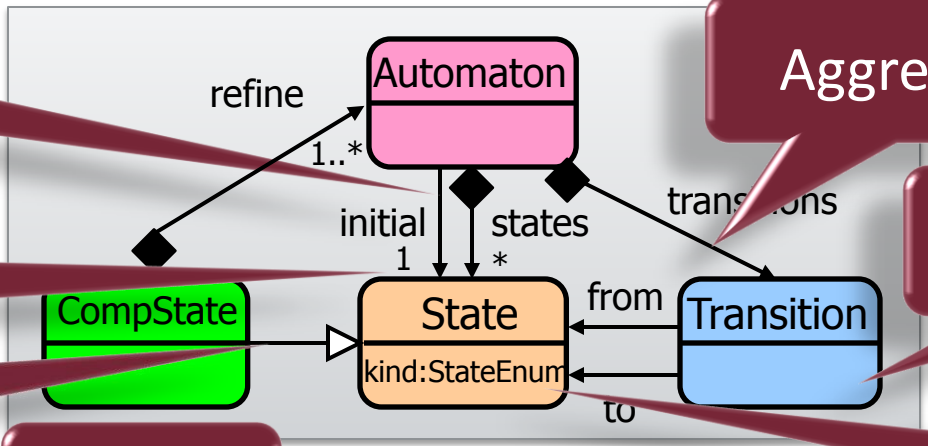
Multiplicity

Generalization

Aggregation

Class

Attribute



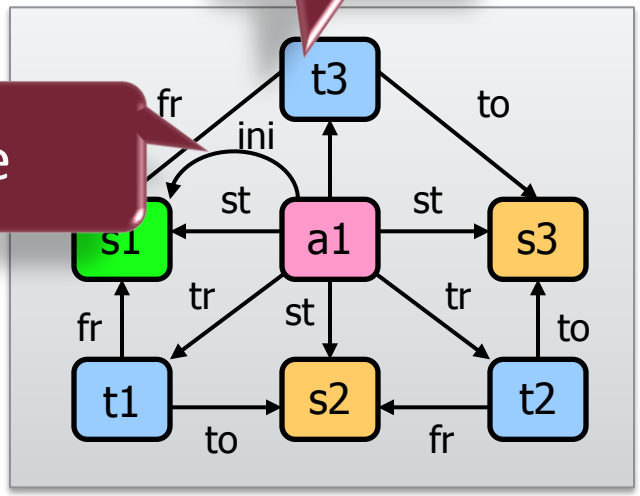
Metamodel level

Metamodel

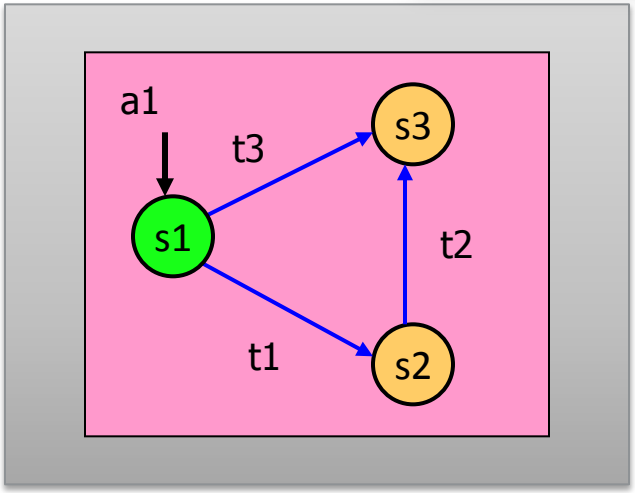
Object

Model level

Role



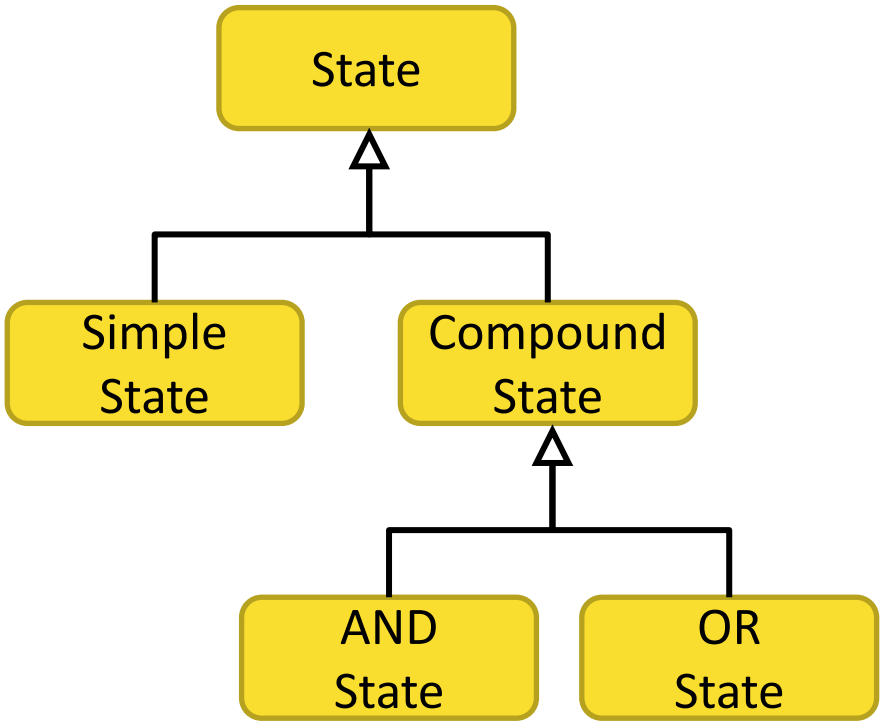
Abstract syntax



Concrete syntax

Classes and Objects

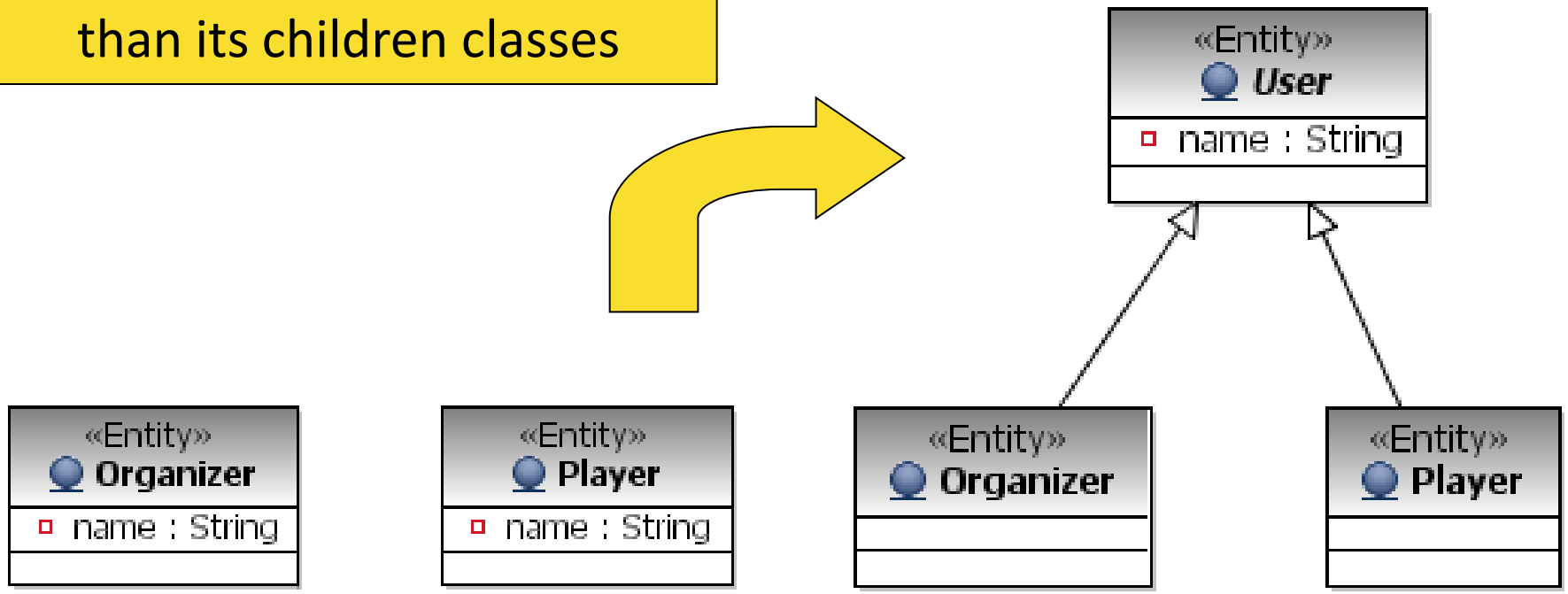
Type hierarchy



- Generalization
  - ≅Inheritance
  - Transitive
  - Reflexive? / Irreflexive?
- How to read?
  - ***SimpleState*** is a subclass of ***State***
  - ***State*** is a superclass of ***SimpleState***
- Substitutability
  - ✓ Subclass instead of Superclass
  - × ~~Superclass instead of Subclass~~

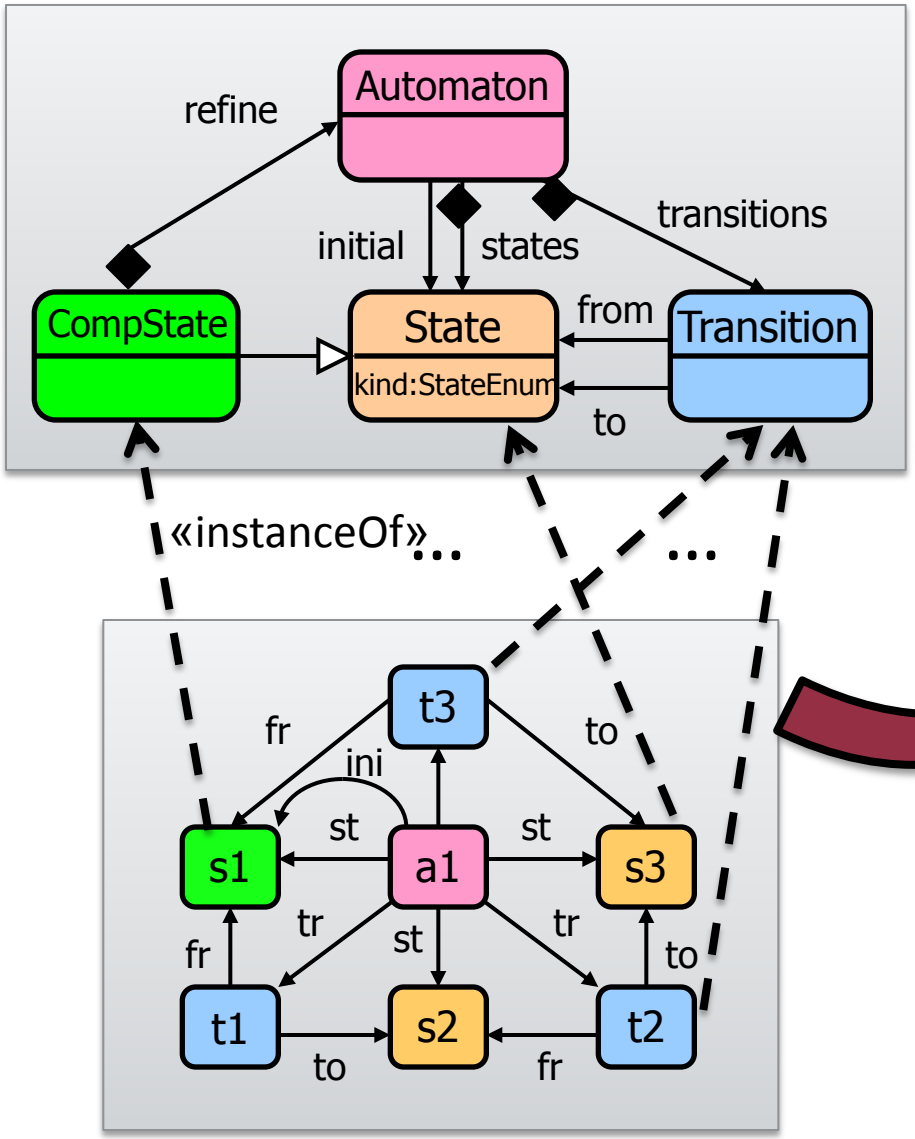
Typical Use of Generalization

Parent class is more general than its children classes



Aim: Lift up common attributes and methods to the superclass

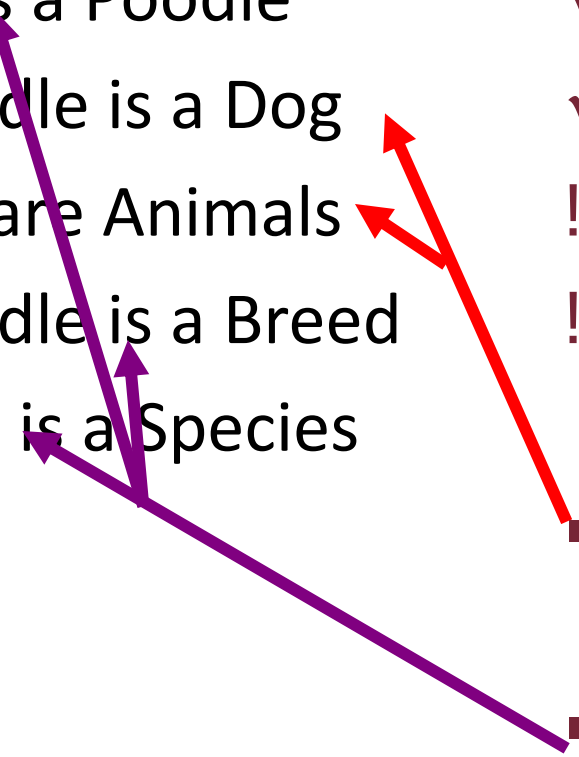
Type conformance /Instantiation /Classification



- Each model element is ***an instance of*** *(conforms to)* a metamodel element
- **Direct type**:
  - No other type exists lower in the type hierarchy
  - s1 → CompState
- **Indirect type**:
  - Superclass of the direct type
  - s1 → State

Classification vs. Generalization

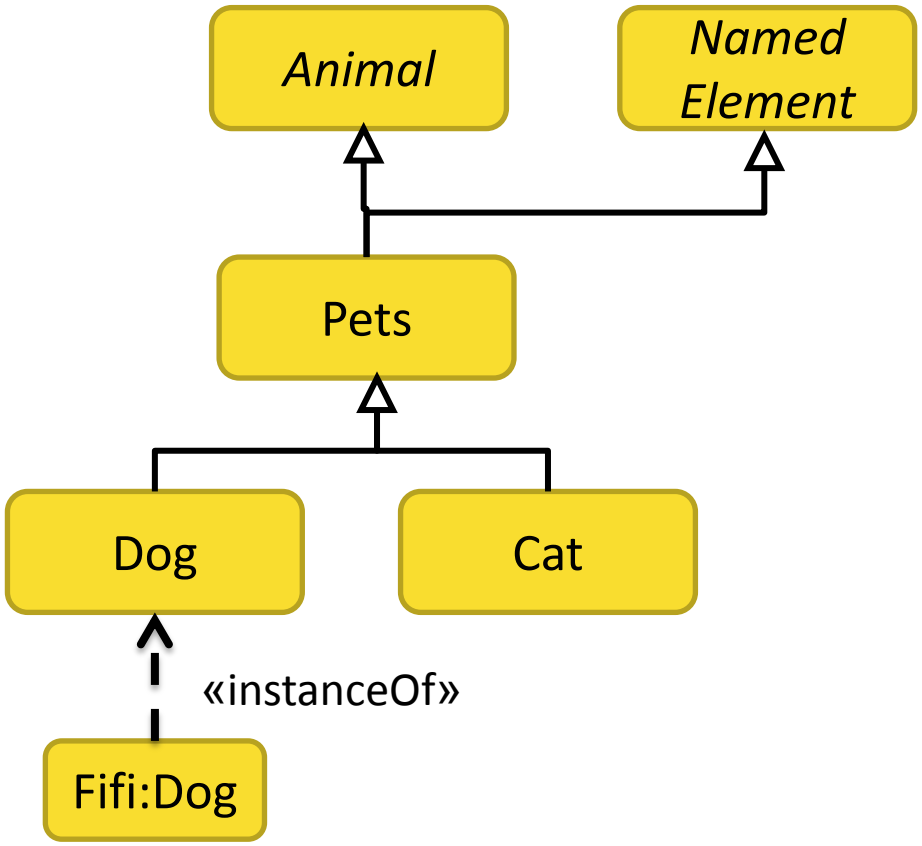
1. Fido is a Poodle
2. A Poodle is a Dog
3. Dogs are Animals
4. A Poodle is a Breed
5. A Dog is a Species



- ✓ 1+2 = Fido is a Dog
- ✓ 1+2+3 = Fido is an Animal
- ! 1+4 = Fido is a Breed
- ! 2+5 = A Poodle is a Species

- Generalization (SupertypeOf) is transitive
- Classification (InstanceOf) is NOT transitive

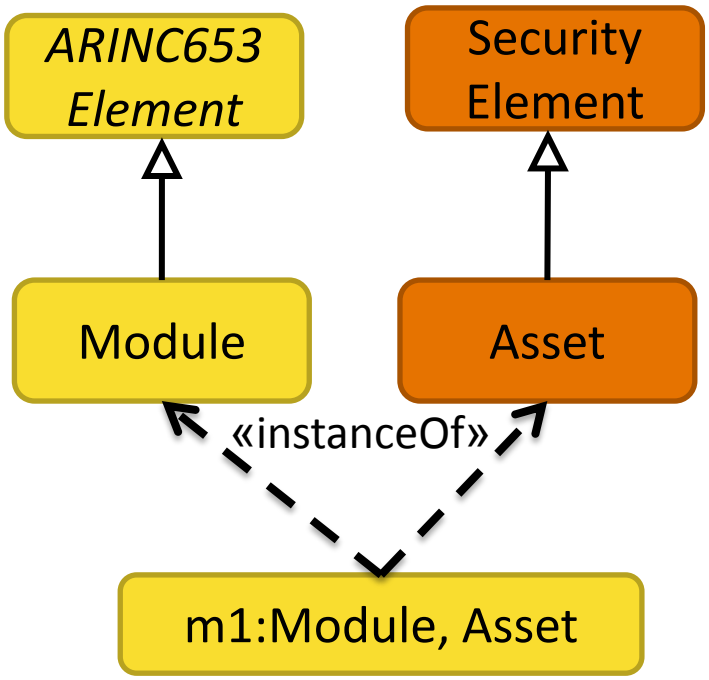
Multiple inheritance



How many types does Fifi have?

- Multiple inheritance:
  - A class in the metamodel has more than 1 supertype
  - Typical use: merge features from different classes
    - One is generic, thus reused in different domains (cf. NamedElement)
    - Other is a general but domain-specific superclass (cf. Animal)
- Restriction:
  For each model element:
  a single type

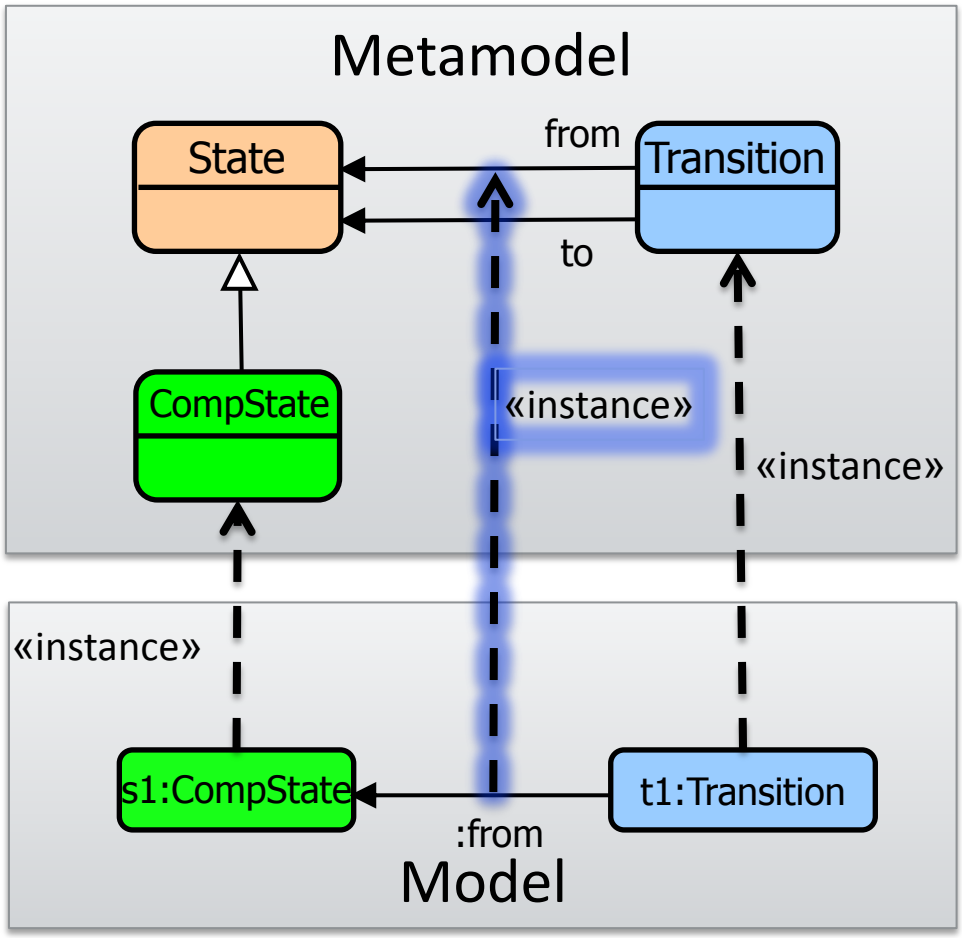
Multiple classification



- Multiple typing / classification:
  - One model element typed against multiple metamodels
  - Rationale: Multi-paradigm / view modeling
  - UML Stereotypes
- Restriction:
  For each model element: a single type in a domain

References and Links

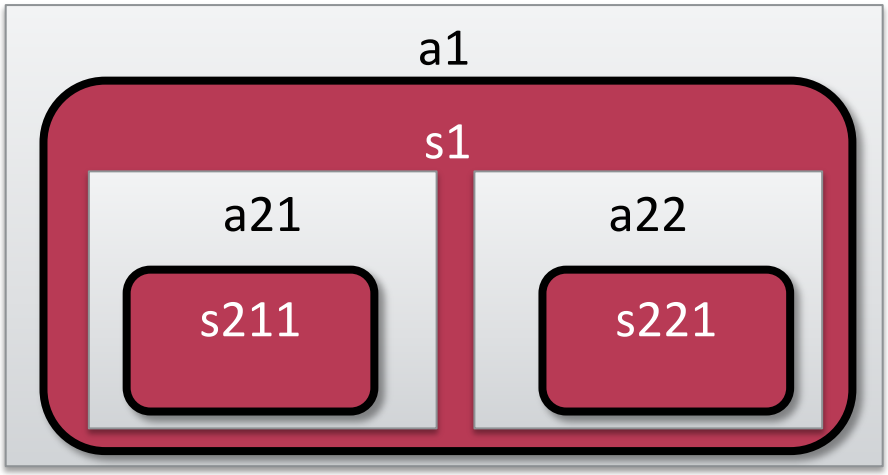
Type conformance of references

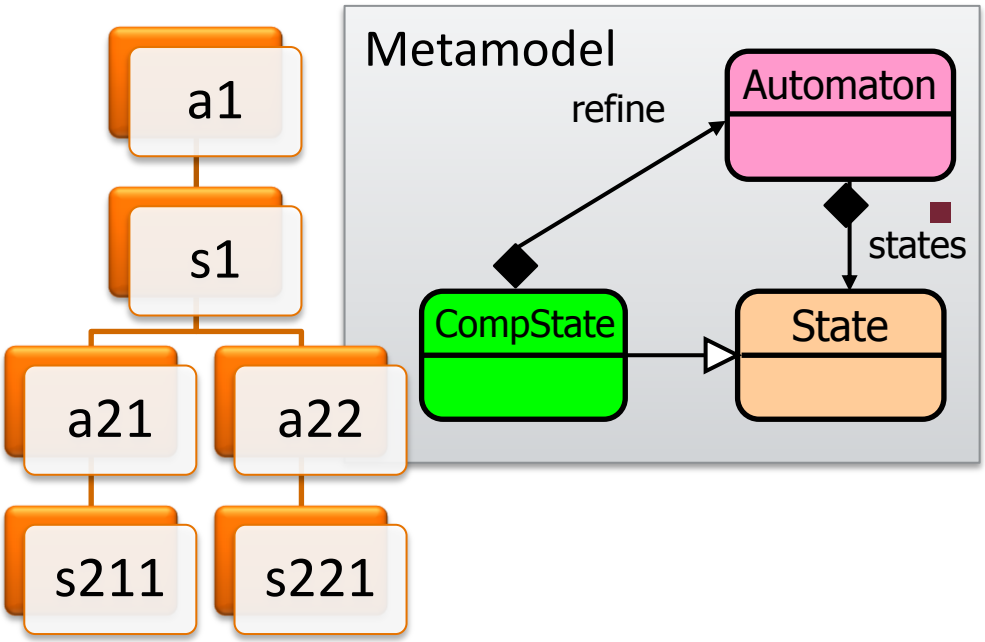


- A link in a model is **type conformant** if
  - ***type(src(link))*** is subtype of ***src(type(link))***
  - ***type(trg(link))*** is subtype of ***trg(type(link))***
  - Informally:
    - The type of the source object is a subtype of the source class of the link's type.
    - The type of the target object is a subtype of the target class of the link's type.

Can you define generalization for references?

Containment hierarchy

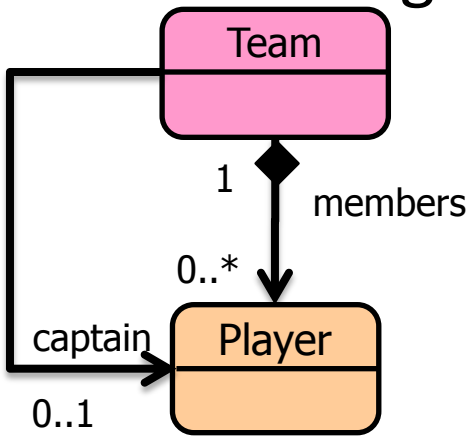




- Each model element has a unique parent
  - N children → 1 parent
  - Single root element
- Aggregation as relationship:
  - Defined in the metamodel along reference edges
  - Provides restriction for instance models
- Circularity
  - No circular containment (in the model)
  - Aggregation relations in the metamodel may be circular (hierarchy)

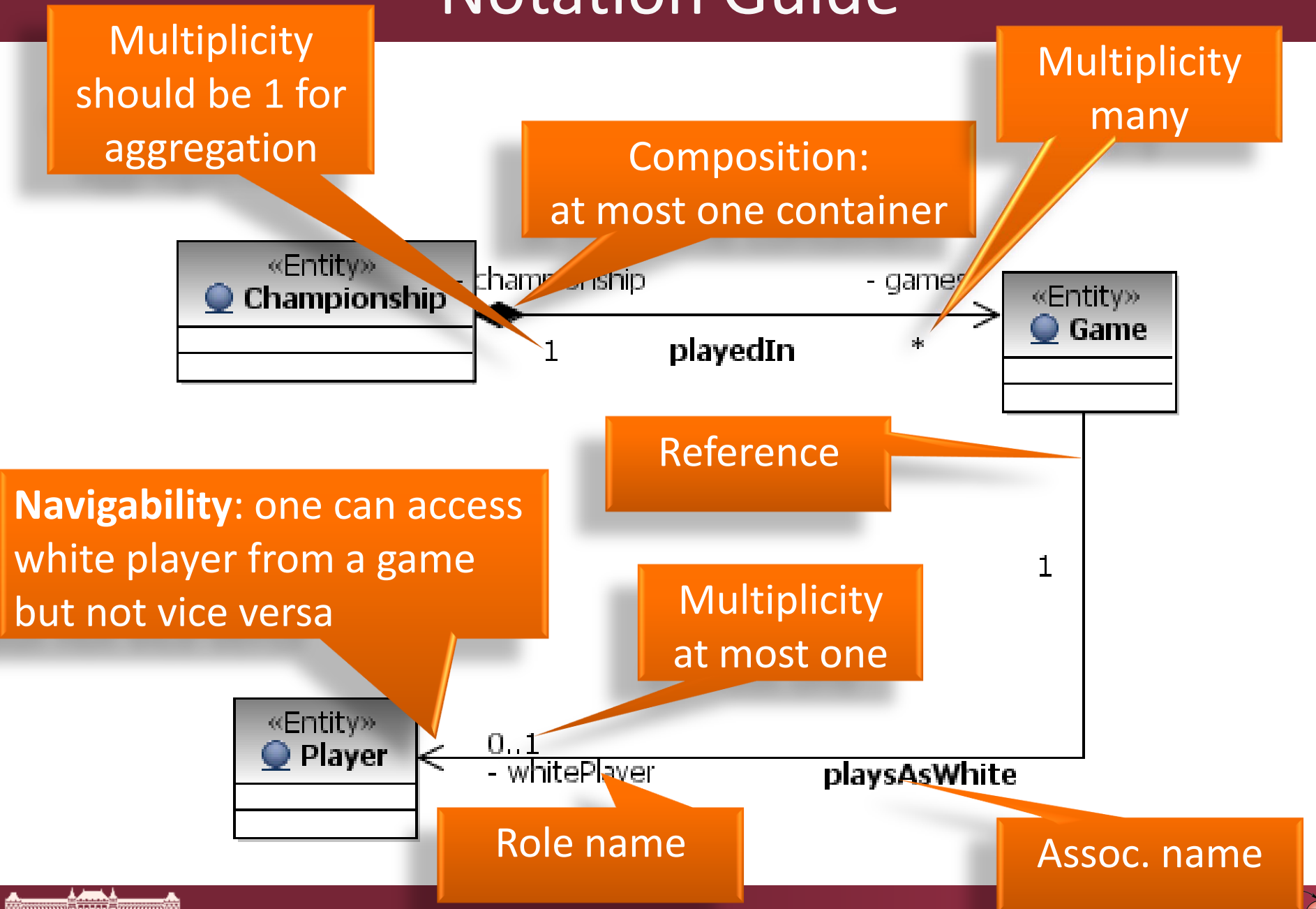
Multiplicity restrictions

- Definition: Lower bound .. Upper bound
  - Lower bound: 0, 1, (non-negative integer)
  - Upper bound: 1, 2, … * (positive integer + any)
- Scope:
  - References: allowed number of links between objects of specific types
  - Attributes: e.g. arrays of strings (built-in values)



Which are the most common multiplicity definitions in practice?

Notation Guide

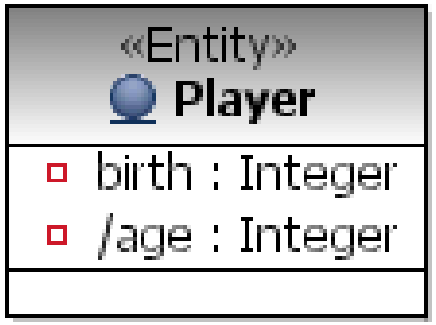


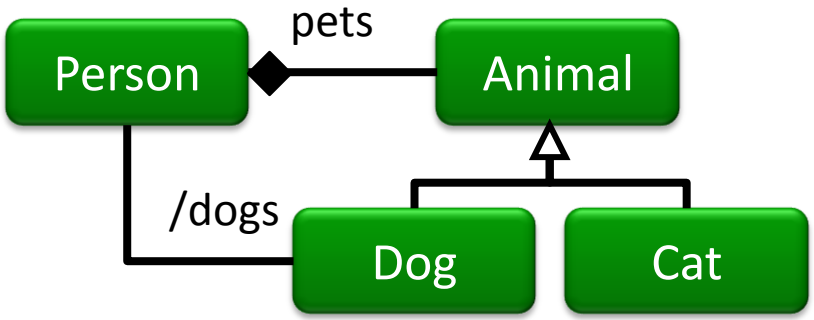
Advanced Concepts and Best Practices

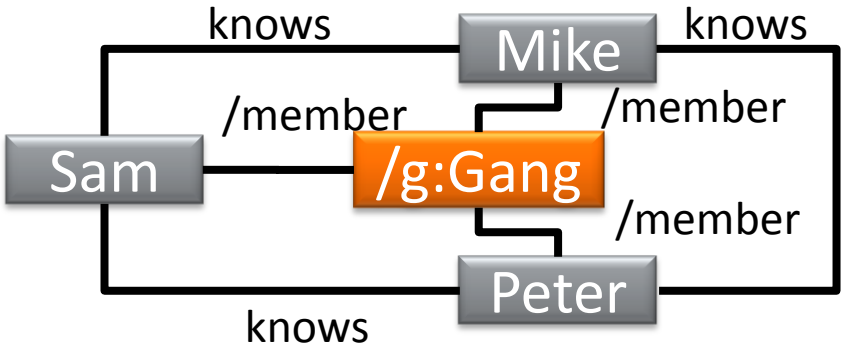
In Domain Modeling

Derived Features

- A derived feature can be calculated from others
  - Usage: helpers for designers / tools
  - It need not be persisted
  - Automatic updates
- Derived attributes:
  `age = currYear – birth`
- Derived references:
  `dogs = -- pets --> Dog`
- Derived objects:
  - „Gang”:
    everyone knows everyone

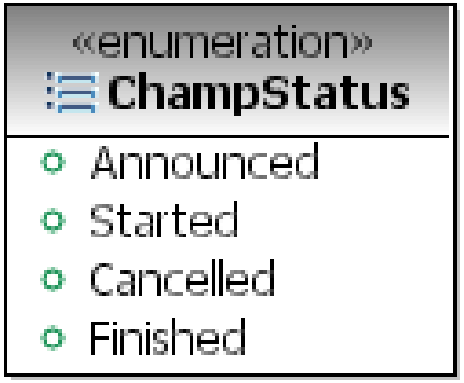






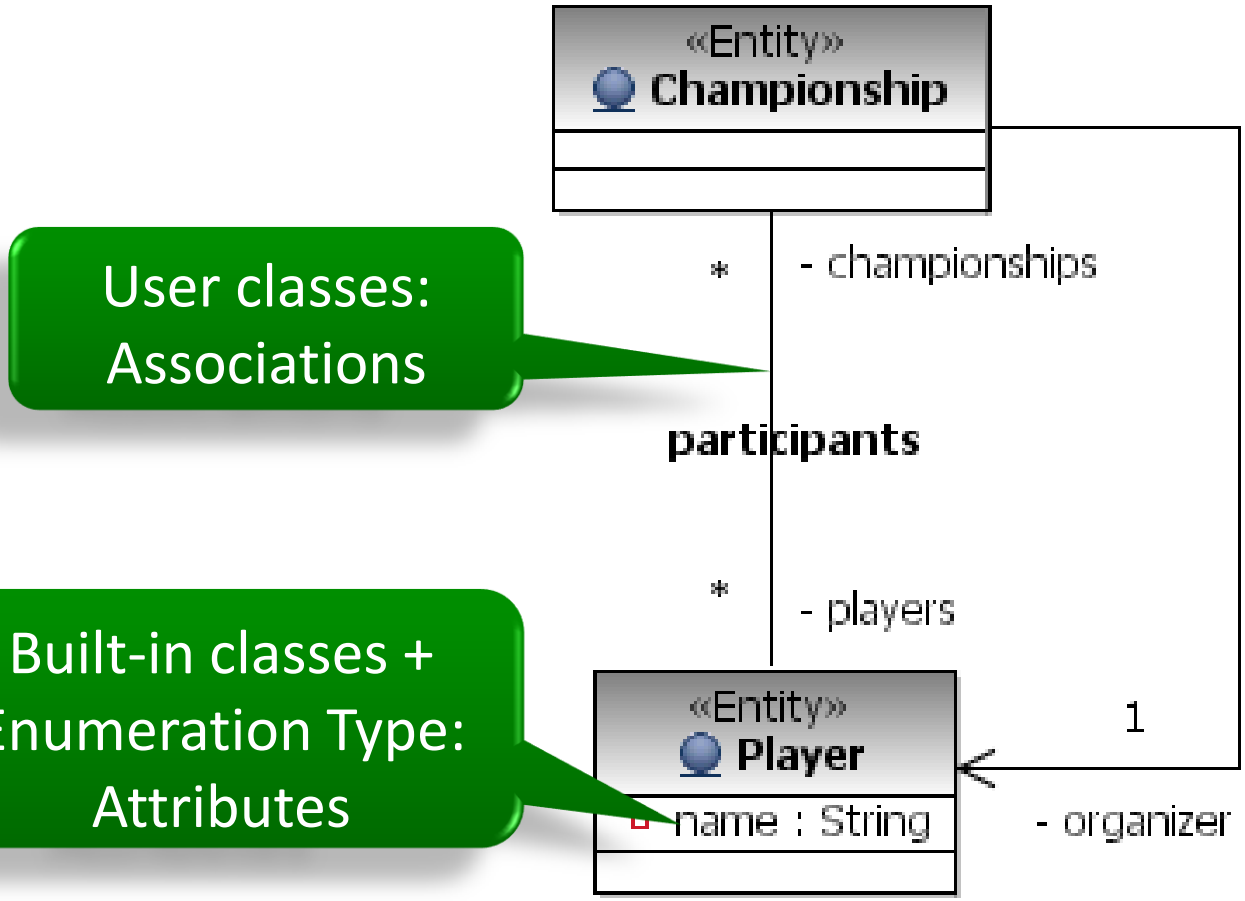
Enumerations

- Enumeration:
  - a fixed set of symbolic values
  - represented as a class with values as attributes

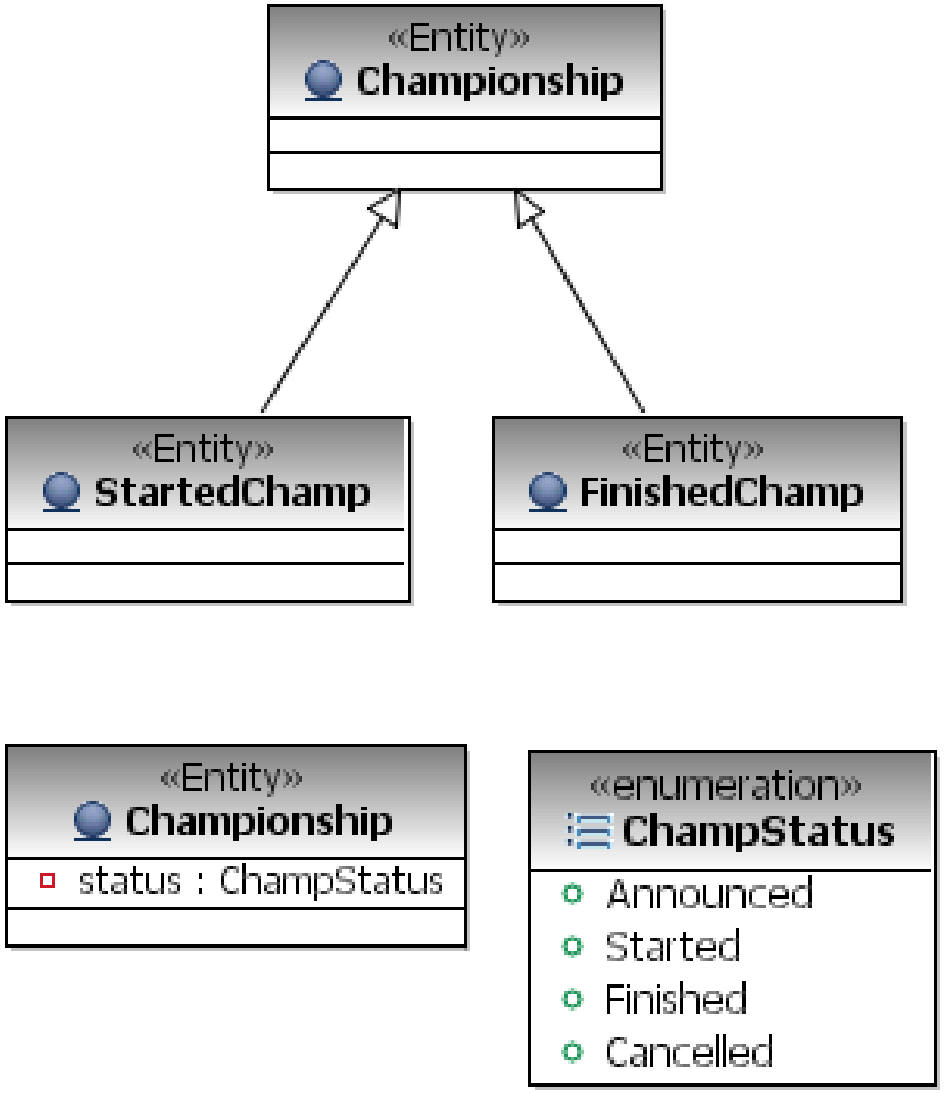


- Usage:
  - Frequently define possible states
  - Use enumerations instead of hard-wired String literals whenever possible

Built-in classes vs. User defined classes

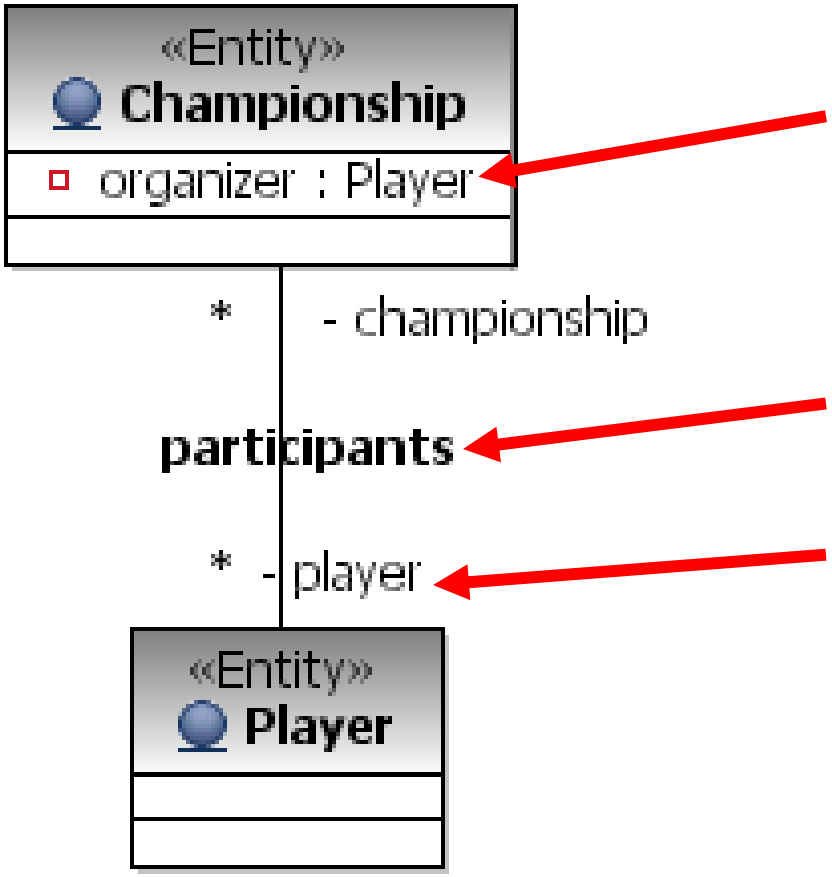


When to avoid generalization?



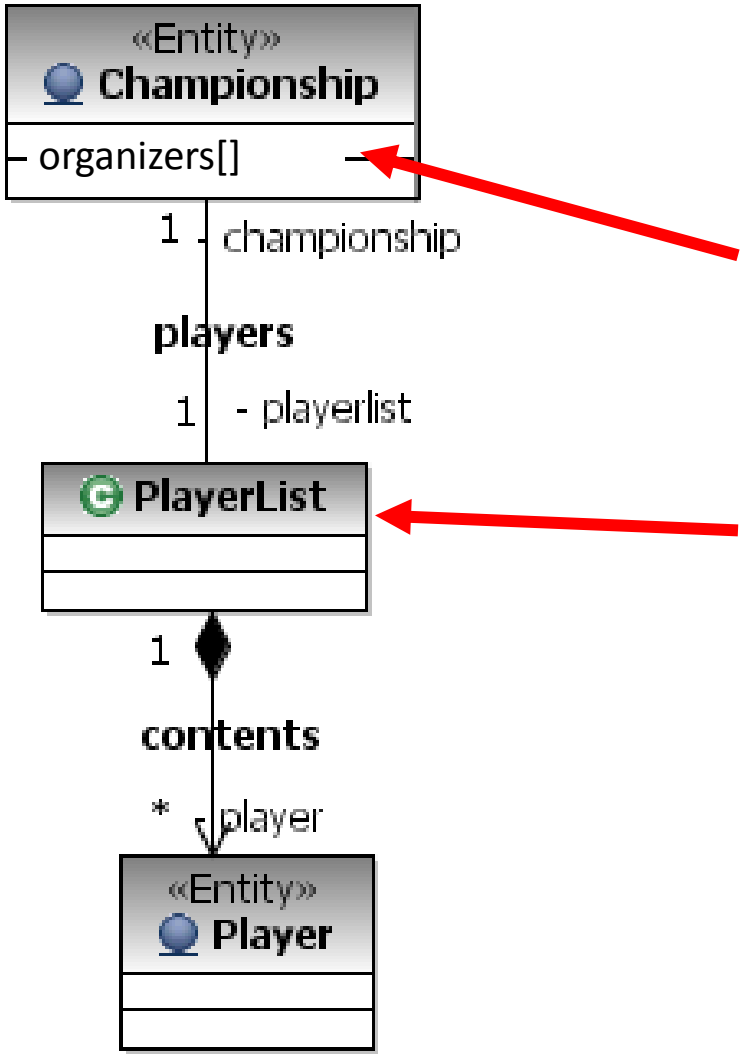
- What happens if a started championship is finished?
- Problem: Retyping of an object is required
- NOTE:
  Use status attribute with enumeration values to store the state of an object that can change

What is Bad Design/Smell here?



- Properties of a user defined type (class) should rather be denoted explicitly
  - OK, if multiplicity is 1
- Naming of associations:
  - prefer verbs to nouns
  - OK: *participatesIn, participantsOf*
- Naming of roles:
  - 1: singular
  - *: plural
  - OK: *players, championships*

What is Bad Design/Smell here?



- Arrays in attributes
  - Solution:
    an *organizes* association
- Explicit lists
  - Solution:
    a single *playsIn* association
- NOTE:
  Lists and arrays are programming constructs and not domain elements!

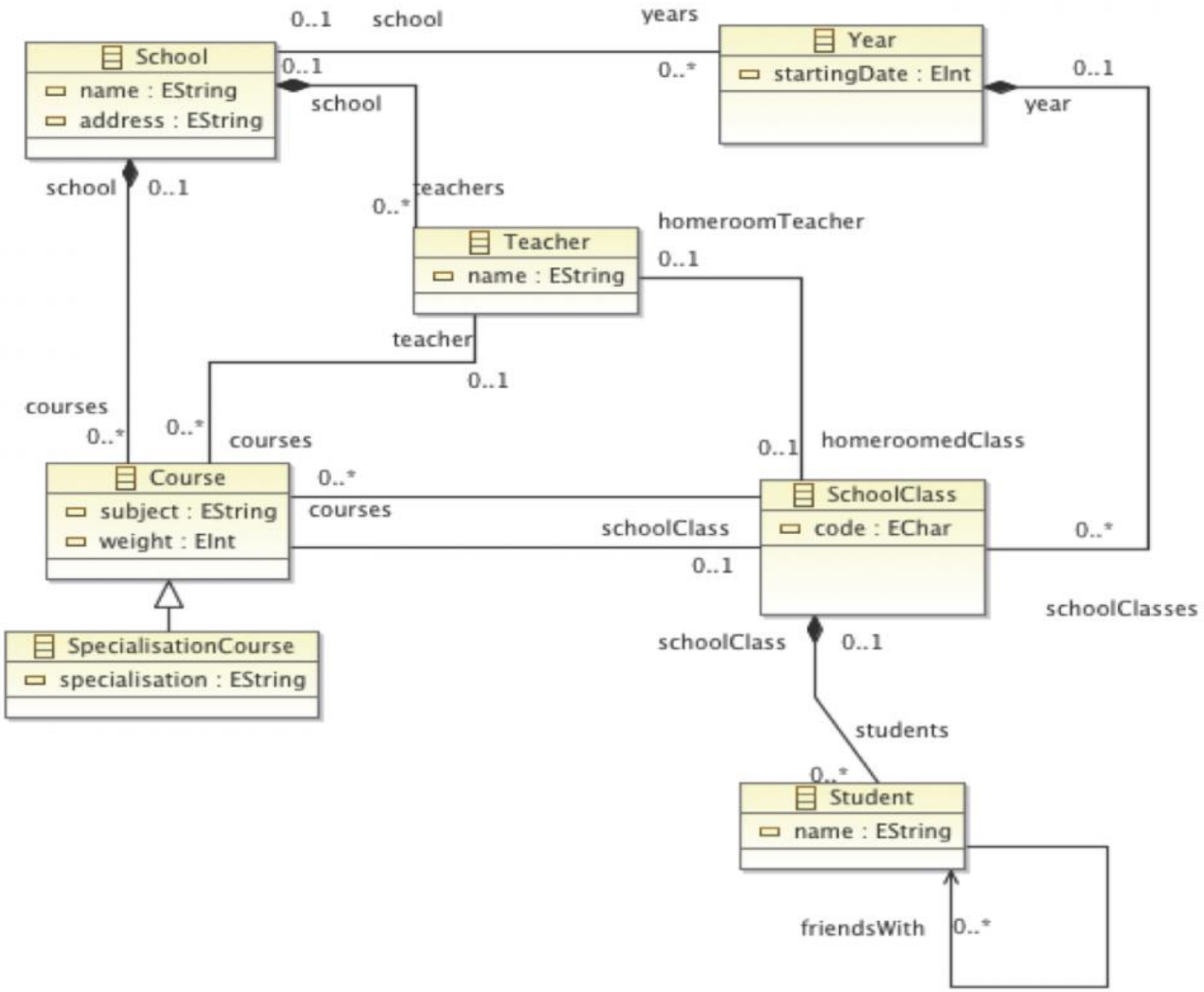
Domain Modeling Examples

Practical exercises

The School Domain

- A school (identified by its name and address) has teachers as employees who teach courses (identified by their subject) in different years.
- Each class in a specific school year has a headmaster (homeroom) teacher
- Students of a specific year attend their own classes, and they may be friends with each other
- Teachers and students are identified by their names.
- Specialization courses can be taken by 11th and 12th grade students

The School Domain



Paper Review System: The Story

- The paper review system is used by authors who log in electronically for the conference and then fill in a form including their name, the most important attributes of the paper to be submitted (such as title, abstract), and mark the conference topics related to the paper. The paper itself is usually submitted by a later deadline using the paper ID received when registering the paper. Later the authors may observe the reviews received for their paper. If their paper gets accepted by the program committee, the final version of the paper needs to be uploaded to the system
- The paper review system is also used by the reviewers, who receive their login parameters in email. They need to fill in their contact details for the conference chair when logging in to the system for the first time.
- After skimming through the titles and abstracts of submitted papers, each reviewer indicates their conflicts (i.e. those paper where the authors are close colleagues or former co-author). He or she also indicates those topics where he or she is an expert.
- The conference chair assigns the papers to at least three reviewers using semi-automated assistance from the system. The basis of assignment is the relevant topics indicated by the reviewers.
- The reviewers fill a review form to evaluate the paper from different aspects including a three-line summary, originality, strong and weak points, reviewer's confidence, author comments, confidential comments. The most important part is the overall recommendation, which can be a score and a textual assessment ranging from strong reject to strong accept.
- Finally, the conference chairs decide on the acceptance or rejection of each paper and send a notification mail to the authors together with the reviews of the paper.

The Paper Review System