A Crash Course in the Unified Modeling Language

Jürgen Börstler
jubo@cs.umu.se
http://www.cs.umu.se/~jubo

A Little History

- Situation in the early ´90s
  - Explosion of OO methods/notations
  - No agreed terminology
  - No standards
  - No method that satisfy a users´ needs completely
  - Booch, OMT, and OOSE among the most popular methods
  - ➨ New generations of methods/notations
- Unification efforts

Rational	OPEN consortium
Booch, Rubaugh (10/94), Jacobson (fall ´95), UML consortium (during ´96)	Firesmith, Graham, Henderson-Sellers, Yourdon,…
Unified (v0.8, 10/95)	Toolbox approach
UML (v0.9, 06/96)	Tailorable
OMG Standardization	Integration with UML?

Current Status

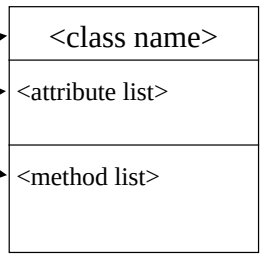
- OMG approval of version 1.1 in Nov ´97
- Version 1.3 released in fall ´98
- XMI (a kind of XML/UML standard)
- Version 2.0 is work in progress
- Notations

Use case diagram	Statechart diagram
Class diagram	Activity diagram
Interaction diagram	Object diagram
Sequence diagram	Component diagram
Collaboration diagram	Deployment diagram

Class Diagrams

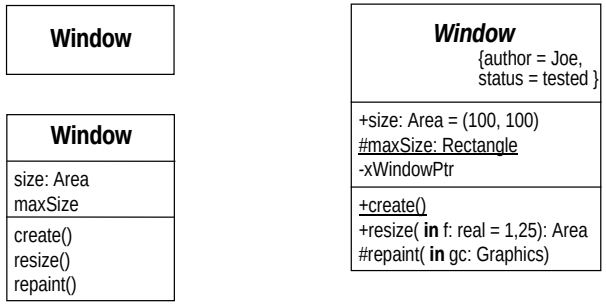
- Static model of the problem (analysis) or the solution (design)
- Describe the types of objects and their (static) interrelationships
- Main elements:
  - Classes
  - Attributes
  - Methods
  - Relationships
    - Dependency
    - Association
    - Aggregation/Composition
    - Generalisation
    - Realisation

strength



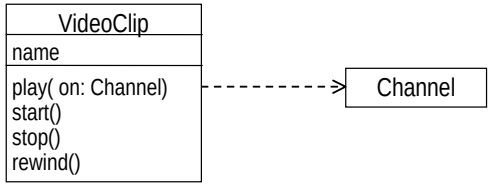
Class Notations

- Analysis
- Design



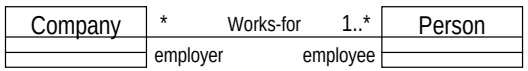
Dependency

- Dependencies define a *usage* relationship
- A change in the used thing may affect things that use it
- Dependencies can have names



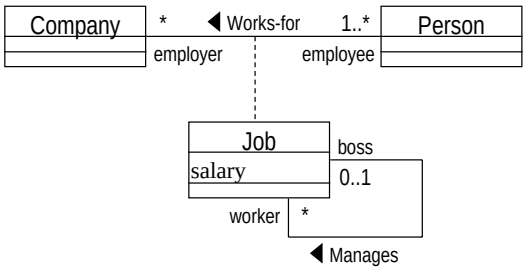
Association 1

- Associations specify structural relationships
- It specifies that objects are interconnected
- Associations can be navigated in both directions (default)
- Associations are quite similar to the relationships known from ER modelling
- Associations have cardinalities
- Associated classes may have roles



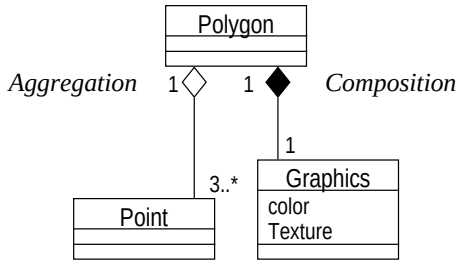
Association 2

- Associations may have attributes and behaviour (*association classes*)
- Association classes are quite similar to the relationship types known from ER modelling
- Association names may have reading directions



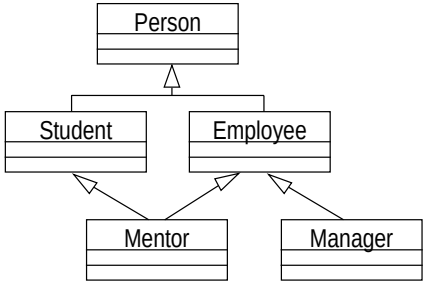
Aggregation and Composition

- Aggregation and composition are specific associations
- They denote the *part-of* or *has* relationship
- Composition is the stronger form
  - Parts are not shared
  - Parts are existence dependent on the whole

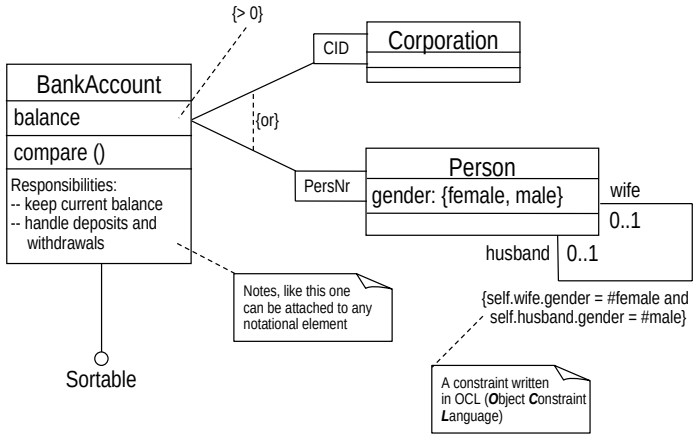


Generalisation

- Generalisation specifies the *is-a* relationship
- It is the strongest relationship between classes
- Subclasses inherit properties from superclasses
- Shown as class hierarchies
- Also known as specialisation or inheritance



Notes, Constraints, Qualifiers, and Interfaces

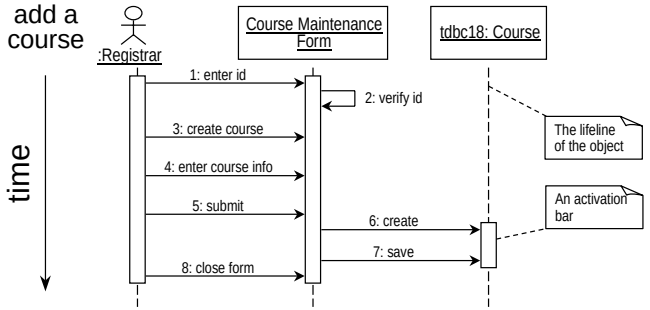


Some Guidelines for Class Diagrams

- Keep the analysis model simple
- Many simple classes are better than a few complex ones
- Use meaningful and familiar names
- Avoid “god classes,” behaviour should be distributed among classes
- Use multiple inheritance very sparely
- Associate methods with the providers of a service, i.e. the “responsible” classes
- Name all associations
- Document your model carefully

Sequence Diagrams 1

- Graphical notation for scenarios
- Describe how objects collaborate to provide a service
- Good tool to uncover missing behaviour and/or missing objects/classes



Sequence Diagrams 2

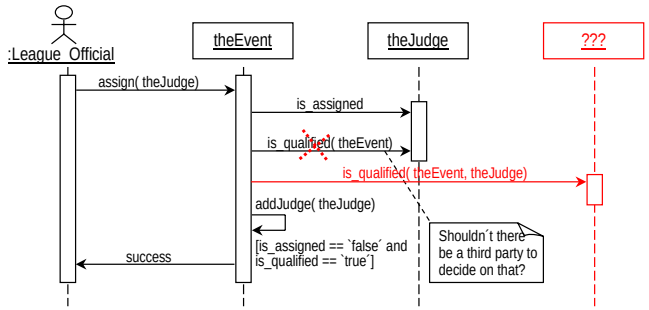
- Provide lots of annotations
  - Comments (script text)
  - Object creation and deletion
  - Message return
  - Conditional messages and Iteration
  - …
- Supports notations for concurrency
  - Timing and activation
  - Asynchronous messages
- Collaboration diagrams are semantically equivalent, but focus on structural relationships between objects
- Sequence diagrams ≈ OIDs ≈ MSCs

Some Guidelines for Sequence Diagrams

- Keep the analysis scenarios simple (do not overspecify your OIDs)
- Analysis scenarios should usually be initiated by an actor
- Discriminate instances of the same class
- Make all assumption clear, use pre- and postconditions
- Concentrate on the major scenarios
- Manage consistency between use cases, scenarios, and OIDs
- Focus on general behaviour, do not fix methods too early
- ➨ Document your findings in your class diagram (and glossary)

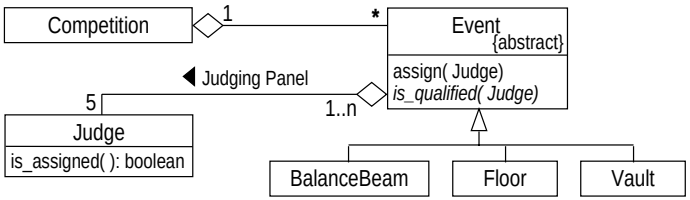
The GSS Case Study Continued 1

- Assign a judge to an event
  - An event has a judging panels assigned to it
  - The judging panel consists of “qualified scorers for this event”
- How could this work?



The GSS Case Study Continued 2

- Alternative 1: Let Event handle qualified scorers
  - Make Event an abstract class
  - Defer determination of qualified scorers to specific event classes

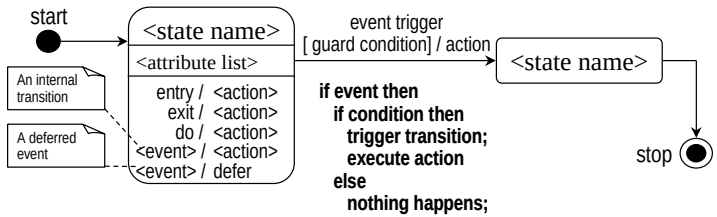


The GSS Case Study Continued 3

- Alternative 2..n:
  - Try to give other existing classes this responsibility
    - League
    - Equipment
    - …
  - Define a special purpose class
  - Find out where this responsibility lies in the “real world”

Statechart Diagrams 1

- Graphical notation for class behaviour modelling
- Describe all possible states and state changes triggered by external stimuli
- Good tool to describe complex object life cycles
- Good tool to uncover missing state information and behaviour



Statechart Diagrams 2

- States can be nested
- Substates may be concurrent
- Transitions may have multiple sources and/or destinations

Some Guidelines for Statechart Diagrams

- Use STDs when there is (complex) state-dependent behaviour
- Use STDs when you are not sure when things can (are allowed to) happen
- Ensure that all events are covered
- Ensure that all states are reachable
- Ensure that all states can be exited
- Ensure that all transitions can be triggered
- Check each pair of states for missing transactions
- Check consistency with other models

➡ Document your findings

Activity Diagrams 1

- Graphical notation for workflow modelling
- Describe concurrent activities on a high level
- Useful to describe (business) use cases, operations, or scenarios (instead of OIDs)
- A special kind of statechart

Activity Diagrams 2