

Data and Process Modeling

Lecture 3, Fall 2005
UML – structure diagrams

UML – why?

- Unified Modeling Language: a standard
- Discussing design at a higher level of abstraction than code
- Graphical models of object oriented systems
- A common language that allows people from multiple disciplines to communicate
- To learn more:
 - www.uml.org
 - www.omg.org

Agenda

- Introduction to UML
 - UML as a language
 - Basic principles of object orientation
- Logical models
 - Class diagrams
 - Object diagrams
- Software architecture
 - Package diagrams
 - Component diagrams
 - Deployment diagrams

UML as a language

- Prescriptive and descriptive rules
- Suppression of information depending on relevance in use
- Different modes of use: sketch, blueprint, programming language
- How to translate UML into code?

Basic principles of object orientation

A class: "A description of a collection of objects sharing structure, behavioral pattern, and attributes"

An object: "An entity with identity, state and behaviour"

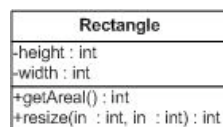
(Matthiesen et al. 2000)

- Object oriented modeling: working out the structure, relationships and behaviour of objects
- Classes and instances
- Inheritance and polymorphism
- Modularity and encapsulation

Class diagrams

Classification of data into types

- How to identify class candidates?
 - Nouns or noun phrases
 - Cohesion: few clear responsibilities
 - Abstraction, classification, selection
- Notation
 - Naming: singular with a capital letter
 - Different levels of detail depending on purpose



Object Oriented Analysis & Design

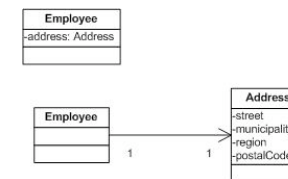
"The structure of an OO system reflects the structure of reality" (or does it?)

- Modelling a system/modelling reality: which parts of reality to bring into the model?
- The difference between analysis and design: domain perspective versus system perspective
- Analysis objects and design objects

Class diagrams

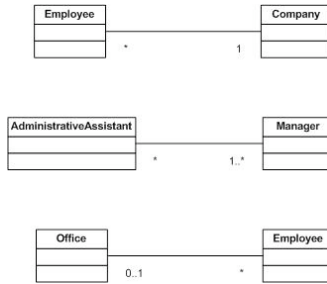
Attributes & operations

- For both operations and attributes, different levels of details can be specified on the diagram
- Operations
`visibility name (parameter-list) : return-type {property-string}`
- Attributes
`visibility name : type multiplicity = default {property-string}`
- Attribute or association?



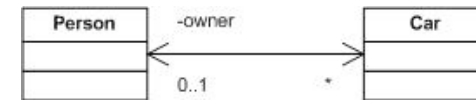
Class diagrams Associations - multiplicity

- Remember to make the multiplicity explicit!
- The association can be labeled to support understanding
- If there is a need for the association to hold information, consider using an association class
- Read every association in both directions to see if it makes sense



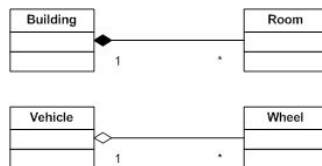
Class diagrams Bidirectional associations

- If you have a Person, you can find his or her car, and if you have a Car you can find the owner



Class diagrams Aggregation & composition

- Aggregation
 - The aggregate is composed of the parts
 - When something controls the aggregate, it also controls the parts
- Composition
 - If the aggregate is destroyed, the parts are destroyed as well
- Aggregation often implies *propagation*, where operations are propagated to the parts



Class diagrams Generalization hierarchies

- Inheritance: the philosophical and the practical approach
- Polymorphism and dynamic binding
- Subtyping and subclassing
- Ex. "avoiding multiple inheritance and letting an instance change class – the Player-role Pattern"

Patterns

- A pattern is the outline of a reusable solution to a general problem encountered in a particular context
- Studying patterns is an effective way to learn from the experience of others
- Patterns as language constructs
- Many kinds: modeling patterns, design patterns, architectural patterns

Constraints, notes & descriptive text

- Sometimes the graphical UML notation is not enough
 - If you communicate with people who are not present
 - If you want to add semantics that can be interpreted by a computer, e.g. rules
- By descriptive text is meant larger documents describing the system, supplementing and/or highlighting aspects of the UML diagrams
- Notes can be attached directly to the diagrams as a small block of text with a bent corner
- A constraint is a special kind of note, written in a formal language

Constraints

- Constraints are written in a formal language called OCL (Object Constraint Language)
- A constraint expresses a logical statement that should evaluate to true
- Ex. {edge -> size() = 1}
 - the number of edges in a line should always equal one
- Ex. {edge -> first().startpoint = edge -> last().endpoint}
 - a polygon should be a closed loop

Dependencies

- Where will changes have an effect?
- Loose coupling = a minimum of dependencies
- Dependency keywords
 - <<call>>, <<realize>>, <<use>>
- Dependencies are shown with dashed arrows



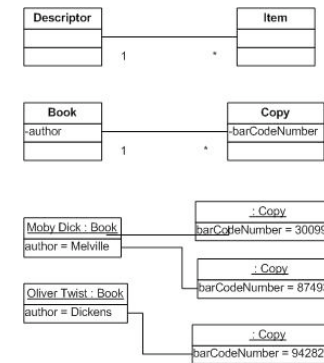
Class diagrams and object diagrams

- The class diagram is an abstract representation of all possible configurations of objects
- The object diagram is one concrete example of such a representation, describing the relationship between instances at runtime
- The connections between boxes are links, or pointers, NOT associations with multiplicity
- However, it is consistent with the class diagram, e.g. multiplicities are the same
- The object diagram can only show associations, not generalisations – why?

Software architecture

- Designing the global organisation of a software system
- Dividing software into subsystems
 - Deciding how these will interact
 - Determining their interfaces

Example of object diagram Item-Descriptor Pattern



Architectural model

Reasons to develop it

- To enable everyone to better understand the system
- To allow people to work on individual pieces of the system in isolation
- To prepare for extension of the system
- To facilitate reuse and reusability

Architectural model

Typical contents

- Different views of the system
 - Logical breakdown
 - Dynamics of interaction among components at runtime (topic of the next lectures)
 - Data shared among subsystems
 - The components that will exist at runtime, and the machines and devices on which they are located

Architectural model

Challenges

- Producing a relevant picture of a large and complex system
- It should be easy to understand (even by clients) by looking at different views and how these relate to each other
- The architecture model should be stable, allowing new features to be added with only minor changes to the overall model

Architectural patterns

- Multi-Layer
- Client-Server
- Broker
- Transaction processing
- Pipe-and-Filter
- Model-View-Controller
- Service-Oriented
- Message-Oriented

How to develop an architectural model?

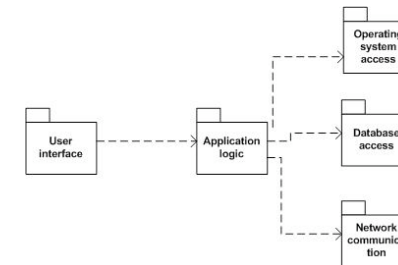
1. Sketch outline
 - Domain model and use cases
 - Main components
 - Architectural patterns
2. Identify interaction between components
 - Decide how data and functionality will be distributed among components
 - Consider reusing existing frameworks
3. Finalize the interfaces of each component
4. Define final class diagrams and interaction diagrams

Describing an architecture using UML

- All UML diagrams can be useful, but here we will cover
 - Package diagrams
 - Component diagrams
 - Deployment diagrams

Package diagrams

- When designing packages, use the principles of *cohesion* and *coupling*
 - The content can be reused together
 - Minimize the number of dependencies
- Dependencies are shown with dashed arrows
- It is dependencies between *elements* of different packages
- A change made to the interface of a package will require modification to packages that depend upon it



Component diagrams

- Whereas package diagrams show the logical grouping of design elements, component diagrams show the physical grouping
- What is a component?
 - Something that can be sold and upgraded independently
 - Provides an interface to other components
- Relationships between components:
 - A component may execute another component, or a method in the other component
 - A component may generate another component
 - Two components may communicate with each other using a network

Deployment diagrams

- Show the physical layout of the system
 - Where the artifacts (files) physically reside
 - Sometimes the files are grouped together in components
 - They can be represented in a class notation, with additional comments
- The artifacts are located at *nodes*
 - Devices or execution environments
- The nodes are connected with communication paths