Graphical Systems Modeling
with UML / SysML

Class diagrams

© Piotr Ciskowski
Perspectives - views:

- **architecture**
  - class diagram
  - object diagram
  - composite structure diagram
  - package diagram

- **Logical view**

- **Use case view**
  - system scope & functionality
    - use case diagram
    - package diagram

- **Dynamic View**
  - behavior
    - sequence diagram
    - activity diagram
    - state machine diagram
    - interaction overview diagram
    - communication diagram
    - timing diagram
    - package diagram

- **Deployment View**
  - hardware
    - deployment diagram
    - package diagram

- **Implementation View**
  - software
    - component diagram
    - package diagram
Class diagram

- best known - object oriented modeling / programing
- static structure diagram
- presents system’s structure

- graphic representation of static elements of system’s domain and relations between them

- classes - generalization, type
- objects - instance, exemplar
- relations
Classes & Objects
Object - class

- **object** - each entity (a person, an animal, a being, a notion, a concept, a thing...) that is important in the domain of the system being modeled in the context of the problem being solved

  - structure, statics - attributes, features, fields
  - behavior, dynamics - operations, services, methods

- **class** - a generalization of objects,
  - a set of objects with the same features, operations, relations, constraints - meaning

  - an abstract notion (not abstract class!)
### Class

- **name:** noun
- **symbol:** rectangle (with compartments)
- **different levels of detail**

<table>
<thead>
<tr>
<th>Client</th>
<th>Account</th>
<th>Reservation</th>
</tr>
</thead>
</table>
|        | - number  
|        | - type    | + makeReservation()  
|        |           | + confirmReservation()  
|        |           | + cancelReservation()  
|        |           | + verifyClient()      |

<table>
<thead>
<tr>
<th>Credit Card</th>
</tr>
</thead>
</table>
| - cardNumber : long  
| - holderName : char  
| - type : byte  
| - expiryDate : date  
| + authorize() : void  
| + block() : void     |

**tool:** Enterprise Architect
Class

- name: noun
- symbol: rectangle (with compartments)
- different levels of detail

tool: Visual Studio
Class

- type, kind, set of elements
- name: noun, capital letter
- attributes - features, characteristics of objects
  - static structure
  - name: small letter
- operations - actions
  - services
  - messages
  - dynamics, behavior
  - name: small letter
Class - object

- class = type, set
- object = instance, exemplar, identity

<table>
<thead>
<tr>
<th>WashingMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>- className : char</td>
</tr>
<tr>
<td>- model : char</td>
</tr>
<tr>
<td>- serialNumber : int</td>
</tr>
<tr>
<td>- load : byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>myWashingMachine : WashingMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>- className = &quot;WM8&quot;</td>
</tr>
<tr>
<td>- load = 10</td>
</tr>
<tr>
<td>- model = &quot;Big Franny&quot;</td>
</tr>
<tr>
<td>- serialNumber = 156749</td>
</tr>
</tbody>
</table>
Class

- stereotypes, responsibilities

<table>
<thead>
<tr>
<th>WashingMachine</th>
<th>WashingMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>«market info»</td>
<td>«market info»</td>
</tr>
<tr>
<td>-brandName</td>
<td>-brandName</td>
</tr>
<tr>
<td>-model</td>
<td>-model</td>
</tr>
<tr>
<td>«tech info»</td>
<td>«tech info»</td>
</tr>
<tr>
<td>-load</td>
<td>-load</td>
</tr>
<tr>
<td>«clothes»</td>
<td>«clothes»</td>
</tr>
<tr>
<td>+ setProgram</td>
<td>+ setProgram</td>
</tr>
<tr>
<td>+ wash</td>
<td>+ wash</td>
</tr>
<tr>
<td>+ dry</td>
<td>+ dry</td>
</tr>
<tr>
<td>«maintenance»</td>
<td>«maintenance»</td>
</tr>
<tr>
<td>- selfCleaning</td>
<td>- selfCleaning</td>
</tr>
<tr>
<td># diagnose</td>
<td># diagnose</td>
</tr>
<tr>
<td>- measureLoad</td>
<td>- measureLoad</td>
</tr>
</tbody>
</table>

take dirty clothes, clean them, dry them, do not ruin'em!!!
CLASS DIAGRAM ELEMENTS
Class diagram

level of complexity

- **conceptual**
  - only key entities
  - no details
  - informal names of classes, attributes and operations

- **analytic**

- **implementation**
  - «implementation class»
  - proper names

  - plus: data types
  - responsibilities
  - visibility
  - scope
  - multiplicities
  - association classes
  - compositions and aggregations
  - generalizations
  - dependencies
  - realizations
Class diagram

naming conventions - classes:

- conceptual level: auction trading session żądanie
- implementation level: Auction tradingSession Zadanie

naming conventions - attributes:

- conceptual level: temporary disk free space, adres wysyłki, cell phone no.
- implementation level: tempDiskFreeSpace, adresWysylki, PhoneNoCellular

naming conventions - operations:

- conceptual level: block SIM card send an SMS
- implementation level: blockSIMcard() sendSMS()
Class diagram

additional elements:

• **constraints** - conditions, limitations of a given element
  - text, formula, pseudo-code,
    OCL (*Object Constraint Language*)
  - in French brackets

![Diagram of WashingMachine class with attributes and methods]

**tools:** Enterprise Architect, Visual Paradigm, Visual Studio
Class diagram

additional elements:

- **notes** - anywhere

![WashingMachine Class Diagram](image)

- check if drying program no. 4 is compliant with Polish Norm no. 111/2012
- load = 8, 9 or 10 kg

**Tool:** Visual Studio
Class diagram

additional elements:

- **responsibilities**
  - auxiliary
  - after preliminary scenarios analysis
  - before full implementation (attributes, operations)

- **CRC card**
  - responsibilities + collaboration

<table>
<thead>
<tr>
<th>WashingMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>responsibilities</td>
</tr>
<tr>
<td>wash dirty clothes, dry them, do not damage</td>
</tr>
</tbody>
</table>

**tool:** Enterprise Architect
additional elements:

- visibility:
  - public
  - private
  - protected
  - packet
Class diagram

additional elements:

• visibility:
  + public
  - private
  # protected
  ~ packet

• rule:
  - attributes
  + operations: getters & setters
Class diagram

additional elements:

- **scope:**

  instance - **classifier**
  
  static

  attributes and operations
## Class diagram

### data types - depend on modeling tool

<table>
<thead>
<tr>
<th>Logic</th>
<th>Boolean</th>
<th>.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>Byte</td>
<td>Byte</td>
</tr>
<tr>
<td></td>
<td>Integer</td>
<td>Short</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long</td>
</tr>
<tr>
<td>Floating</td>
<td>Single</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>Character</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td>Object</td>
<td>Object</td>
</tr>
<tr>
<td></td>
<td>Currency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Variant</td>
<td></td>
</tr>
</tbody>
</table>
Class diagram

attributes notation - implementation level
- \([< + - ~ #>] [ / ] <name> [ : <type> ] [ [ <multiplicity> ] ] [ = <initial value> ] [ { <property> } ]
- \(/ - derived attribute
  - formula usually in attached note
- properties: ordered, [not] unique, readOnly etc.

methods notation - implementation level
- \([< + - ~ #>] <name> [ ( <parameter list> ) ] [ : <property> ]
- parameter list - with commas

parameters notation - implementation level
- \([ in out inout return ] <name> [ : <type> ] [ [ <multiplicity> ] ] [ = <initial value> ] [ { <property> } ]\)
### Class diagram

**attributes, methods, parameters notation - implementation level**

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>- premium : Single = 10000</td>
</tr>
<tr>
<td>- volume : Double = 1000000</td>
</tr>
<tr>
<td>- tick : Percent = 0,01</td>
</tr>
<tr>
<td>- tickValue : Byte = 25</td>
</tr>
<tr>
<td>+ countTicks (sellPrice:Double, buyPrice:Double) : Single</td>
</tr>
<tr>
<td>+ execute(ticks:Single, tickValue:Byte, contracts:Integer) : Double</td>
</tr>
</tbody>
</table>

*tool: Enterprise Architect, example based on: Wrycza et al., Język UML 2.0 w modelowaniu...*
RELATIONS
Relations

types of relations:

- association
- generalization
- dependency
- realization
Relations

ASSOCIATIONS
Relations

- **association**
  - a link between elements
  - a semantic relation between classes
  - binary
  - n-ary

- extras:
  - name
  - roles
  - navigation
  - multiplicity
  - aggregation
Relations

- association
  - a link between elements
  - a semantic relation between classes

- named - unnamed

Diagram:
- Project is led by Manager
- Project leads Manager
## Relations

- **association**
  - a link between elements
  - a semantic relation between classes

- **multiplicity**
  
<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>exactly one</td>
</tr>
<tr>
<td>n</td>
<td>exactly n</td>
</tr>
<tr>
<td>*</td>
<td>many</td>
</tr>
<tr>
<td>0..1</td>
<td>zero or one</td>
</tr>
<tr>
<td>0..*</td>
<td>zero or more</td>
</tr>
<tr>
<td>0..n</td>
<td>zero to n</td>
</tr>
<tr>
<td>1..n</td>
<td>one to n</td>
</tr>
<tr>
<td>1..*</td>
<td>one or more</td>
</tr>
<tr>
<td>n..m</td>
<td>between n and m</td>
</tr>
<tr>
<td>n..*</td>
<td>more than n</td>
</tr>
<tr>
<td>n,m,p,q</td>
<td>n or m or p or q</td>
</tr>
</tbody>
</table>
Relations - examples

- associations - names, directions, multiplicity:

  - Teacher **teaches** Student
  - Customer Service **serves** Client
  - House **has** Chimney: 1..1
  - Tricycle **has** Wheel: 1..3
  - Egg Container **contains** Egg: 1..6, 12, 24
Relations

- **association**
  - a link between elements
  - a semantic relation between classes

- special kinds:
  - multiple
  - recursive
  - qualified
Relations

- **association class**
  - precise description of association
  - association itself - has some attributes

  - association = class:
    - name
    - attributes
    - operations

  - one association - one object of association class
Relations

AGGREGATIONS
Relations

- **aggregation**
  - „has a”
  - part-whole
  - binary
  - contained class - container
  - segment - aggregat
Relations

- **composition**
  - „owns a”
  - part-whole

- **difference:**
  - object are made of other objects
  - aggregation: weak: components can exist alone - independent
  - composition: strong: destroy the object = destroy components

- **aggregation** or attribute
  - simple components - attributes
  - complex components - aggregations
Relations

CONSTRAINTS
Relations

- aggregation constraints
Generalizations

Relations
Relations

- **generalization**
  - inheritance - class hierarchy
  - superclass - subclass
  - root - leaf
  - abstract classes

- **generalization**
  - object of the child class is a kind of an object of the parent class
  - if not - association
Relations

- generalization
  - constraints:  
    - \{complete\} - \{incomplete\}
    - \{disjoint\} - \{overlapping\}
Dependencies & Realizations
Relations

- **dependency**
  - one class depends on the other
  - changing one class $\Rightarrow$ changes in the other
    - independent $\Rightarrow$ dependent
Relations

- dependency
  - one class depends on the other
  - changing one class ⇒ changes in the other
  - independent  dependent

- object of one class among parameters of other class operation
- objects of one class send messages to objects of the other class
- objects of one class contain objects of the other class

- scale of changes
Relations

- realization
  - class - interface
  - interface - a set of operations
    - services provided by class
    - a contract
  - using class - dependency
    providing class - implementation «realize»
  - abstract class - class without objects
    interface - class without implementation
SUGGESTIONS
Class diagram - suggestions:

- do not over...
  - draw
  - relate
  - detail - moderate notation

- big picture → details

- use many diagrams:
  - for different: needs, perspectives, fragments, detail levels
  - not all classes, not all details
  - structure + behavior
  - business - technical