Model Driven Architecture
UML Diagrams

Prof. Dr. Peter Thiemann

Universität Freiburg

10.05.2006
Kinds of UML Diagrams

UML defines several kinds of diagrams that model different aspects of software:

- **structural**: class diagram, package diagram, object diagram, component diagram, deployment diagram
- **behavioral**: use case diagram, sequence diagram, collaboration diagram, statechart diagram, activity diagram
<table>
<thead>
<tr>
<th>diagram</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>classes and their relationships</td>
</tr>
<tr>
<td>package</td>
<td>grouping mechanism for class diagrams</td>
</tr>
<tr>
<td>object</td>
<td>snapshot of a system state</td>
</tr>
<tr>
<td>component</td>
<td>organization of physical software parts</td>
</tr>
<tr>
<td>deployment</td>
<td>physical resources of a system; assignment of software components to hardware</td>
</tr>
<tr>
<td>diagram</td>
<td>content</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>use case</td>
<td>describe goal-directed interactions of external actors with the system</td>
</tr>
<tr>
<td>sequence</td>
<td>communication and interaction between objects; ordering of messages</td>
</tr>
<tr>
<td>collaboration</td>
<td>object diagram with extensions for message flow and sequencing</td>
</tr>
<tr>
<td>statechart</td>
<td>dynamic behavior to external stimuli; reactive and concurrent systems</td>
</tr>
<tr>
<td>activity</td>
<td>description of control flow between activities; concurrency</td>
</tr>
</tbody>
</table>
UML Ingredients Important for MDA

class diagram defines static structure of the implementation
statechart diagram specify dynamic behavior of objects

OCL (uses in class diagrams)
- definition of invariants
- specification of operations

action semantics definition of operations
Class Diagrams

- representation of **classes** and their **structural relationships**
- no behavioral information
- UML concrete syntax is graph with
  - **nodes** (boxes): classes
  - **edges** (different kinds of arrows and lines): various relationships between classes
- may contain interfaces, packages, relationships, as well as instances (objects, links)
- degree of detail depends on phase
Classes

### Student

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>matriculationNumber</td>
<td>issueCertificate ()</td>
</tr>
<tr>
<td>name</td>
<td>enterGrade ()</td>
</tr>
<tr>
<td>grades</td>
<td>listDegrees ()</td>
</tr>
<tr>
<td>count</td>
<td></td>
</tr>
</tbody>
</table>

- only name compartment obligatory
- additional compartments may be defined (responsibilities, events, exceptions, ...)
Contents of Name Compartment

1. Optional stereotype
   Extension mechanism:
   Changes meaning, may influence visual appearance

2. Class name
   Abstract classes indicated by italics

3. Optional property list of tagged values
   {abstract}, {leaf, author=”John Doe”}
   Extension mechanism
Example for Stereotypes

<table>
<thead>
<tr>
<th>«enumeration»</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>red</td>
</tr>
<tr>
<td></td>
<td>green</td>
</tr>
<tr>
<td></td>
<td>blue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>«abstract»</th>
<th>Ticket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>venue</td>
</tr>
<tr>
<td></td>
<td>price</td>
</tr>
<tr>
<td></td>
<td>validity</td>
</tr>
</tbody>
</table>
Attributes compartment

Syntax of an attribute

\[
[visibility] [/] name \[ : type \][[ multiplicity ordering ]]
[= default] \{ properties \}
\]

- **visibility**: +, #, −, ~
- **name**: derived attribute
- **type**: classifier name / PL type
- **multiplicity**: interval (def: 1)
- **ordering**: ordered, unique, ...
- **default**: language dependent
- **properties**: e.g., {frozen}

- class attributes underlined
Visibility

- from Design/Implementation Level
  - +, public
  - #, protected
  - -, private
  - ~, package
  - alternatively: notation of the implementation language
Multiplicities define intervals of non-negative integers (UML 2.0)

\[
\langle \text{multiplicity} \rangle ::= \langle \text{int} \rangle . . \langle \text{int}^* \rangle | \langle \text{int}^* \rangle \\
\langle \text{int}^* \rangle ::= \langle \text{int} \rangle | ^* 
\]

Most important multiplicities

- 1  exactly one
- 0..1  zero or one
- 0..*  arbitrary many
- *  arbitrary many
- 1..*  at least one
Syntax of an operation

\[ [\text{visibility}] \ name \ ( [\text{parameter-list}] ) \ [ : [\text{return-type}] \ { \text{properties} } ] \]

- **visibility**: 
  +, #, −, ~

- **name**

- **parameter-list**: 
  \( \text{kind} \ \text{name} : \text{type} \)
  \( \text{kind} \in \text{in}, \text{out}, \text{inout} \)

- **return-type**: 
  classifier name / PL type

- **properties**: 
  e.g., \{query\}
  \{concurrency=...\}
  \{abstract\}

- class operations underlined
## Relations in Class Diagrams

### Binary Association
- indicates “collaboration” between two classes
- reflexive association allowed
- solid line between two classes

### Generalization
- indicates subclass relation
- solid line with open arrow towards super class

### Dependency
- indicates implementation dependency
- dashed arrow to dependant entity
- adorned with stereotype to indicate kind of dependency
Variations of Associations

- Multiary associations
- Optional qualifications
  - association name
  - association end name
  - / indicating a derived association
  - decoration with role names
  - navigability (at end, Design)
  - multiplicities (at end, Design)
- Aggregation and composition
- Association classes (attach attributes and operations)
Example: Class Diagram

- **class**
  - class name
  - name of abstract class
  - attribute1
  - attribute2: Typ = default
  - derived attribute
  - class attribute
  - abstractOperation1()
  - op2(parmList): result type

- **inheritance**
  - vehicle
  - flying object
    - car
    - airplane
    - bird

- **association**
  - Class 1
    - n role
  - class 2
    - m role

implementation of op2
Example: Class Diagram with Associations

- reflexive association
- multiple parallel associations
- multiplicities
Example: Navigability of Associations

- A1 ↔ B1: both ends navigable
- A2 ↔ B2: both ends not navigable
- A3 ↔ B3: both ends unspecified
- A4 → B4 navigable but not B4 to A4
- A5 → B5 navigable reverse direction unspecified
Aggregation and Composition

- Aggregation (and composition) indicate a part-of relation
- Composition binds tighter: “existential dependence”
- Graphical notation: open (filled) lozenge at container

Diagram:

- Roof
- House
- Wall
- Door
Constraints on Classes and Associations

- Constraints wrt object state or association
- Notation: \{constraint\}
- Example constraints on associations:
  \{sorted\}, \{immutable\}, \{read-only\}, \{subset\}, \{xor\}
- natural language, pseudo code, predicate logic, . . . , OCL
A statechart diagram is a finite automaton extended with output
(combination of Moore and Mealy automaton)

Deterministic (Mealy) finite automaton: \( (Q, \Sigma, \Lambda, \delta, q_0, F) \)

- \( Q \) set of states
- \( \Sigma \) input alphabet
- \( \Lambda \) output alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \times \Lambda \) transition function
- \( q_0 \in Q \) initial state
- \( F \subseteq Q \) set of final states

Moore automaton associates output with state

Graphical notation extended with operators

- hierarchical states
- composite states
- conditional transitions
Statechart/States
Lifecycle of a Car

Parked
- startEngine()
- openDoor()
- stopEngine()
- closeDoor()

In Motion
- [need gas]

@ gas station
- cleanWindow()
- checkOil()

[Full]

Fueling
- receiveFuel()

Crashed
- wreck()
Statechart/Entry and Exit Actions

Dialing

Start
entry/start tone
exit/stop tone

Partial Dial
entry/number.
append(n)

[number.isValid()]

digit(n)

Dialing
Labels on transitions:

- `event [guard] [/method list]`
  - if present, `guard` must be true to trigger the transition
  - free text or OCL

- “Transitions are instantaneous”
“An event is a noteworthy occurrence […] that may trigger a state transition.” [UML 2 specification]

Kinds of events (signals)

- condition changes from false to true
  - event happens on each such change; guard is evaluated once when its event fires; if the guard is false, then the event is lost
- receipt of explicit signal
- invocation of an operation (call event instance)
- timer event: after period of time or at specified date/time
Signals form a hierarchy

Attributes are event parameters: MouseDown (100, 200)

Elapsed time event: **after** (10 seconds) from entry to current state unless otherwise specified

Time event: **when** (date = 20060514)