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

UML Diagrams/Structural and Static Aspects

diagram	content
class	classes and their relationships
package	grouping mechanism for class diagrams
object	snapshot of a system state
component	organization of physical software parts
deployment	physical resources of a system; assignment of software components to hardware

UML Diagrams/Behavioral and Dynamic Aspects

diagram	content
use case	describe goal-directed interactions of external actors with the system
sequence	communication and interaction between objects; ordering of messages
collaboration	object diagram with extensions for message flow and sequencing
statechart	dynamic behavior to external stimuli; reactive and concurrent systems
activity	description of control flow between activities; concurrency

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

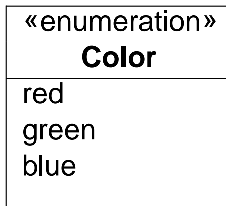
Student	
matriculationNumber	name compartment
name	attributes
grades	
count	class attribute
issueCertificate ()	operations
enterGrade ()	
listDegrees ()	class method

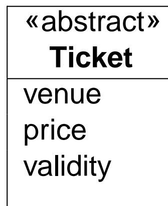
- only name compartment obligatory
- additional compartments may be defined (responsibilities, events, exceptions, . . .)

Contents of Name Compartment

1. optional stereotype
   «abstract», «enumeration», «interface», «controller»
   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





Attributes compartment

Syntax of an attribute

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

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

- class attributes underlined

Visibility

- from Design/Implementation Level
- `+`, `public`
- `#`, `protected`
- `-`, `private`
- `~`, `package`
- alternatively: notation of the implementation language

Multiplicity

Defines interval of non-negative integers (UML 2.0)

$$\langle \textit{multiplicity} \rangle ::= \langle \textit{int} \rangle \texttt{..} \langle \textit{int*} \rangle \mid \langle \textit{int*} \rangle$$
$$\langle \textit{int*} \rangle ::= \langle \textit{int} \rangle \mid \texttt{*}$$

Most important multiplicities

`1`	exactly one
`0..1`	zero or one
`0..*`	arbitrary many
`*`	arbitrary many
`1..*`	at least one

Operations Compartment

Syntax of an operation

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

visibility	**+**, **#**, **-**, **~**	Design, Implementation
name		all phases
parameter-list	*kind name* **:** *type* *kind* $\in$ **in**, **out**, **inout**	Design, Implementation
return-type	classifier name / PL type	(Analysis), Design, Implementa
properties	*e.g.*, {query} {concurrency=...} {abstract}	(Analysis), Design, Implementa

- 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

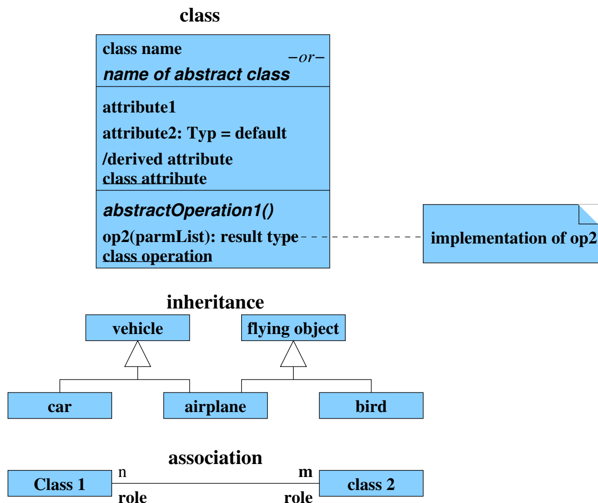
Dependency

- indicates implementation dependency
- dashed arrow to dependant entity
- adorned with stereotype to indicate kind of dependency

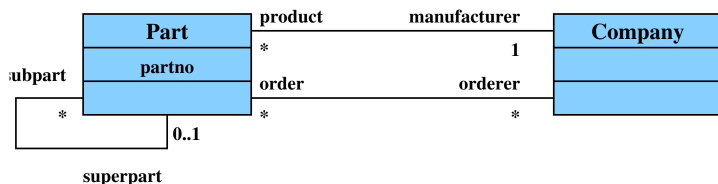
Variations of Associations

- 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

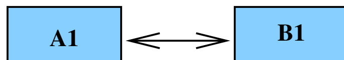


Example: Class Diagram with Associations

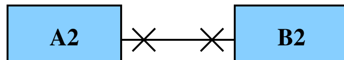


- reflexive association
- multiple parallel associations
- multiplicities

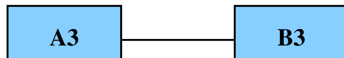
Example: Navigability of Associations



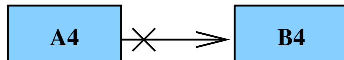
both ends navigable



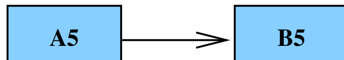
both ends not navigable



both ends unspecified



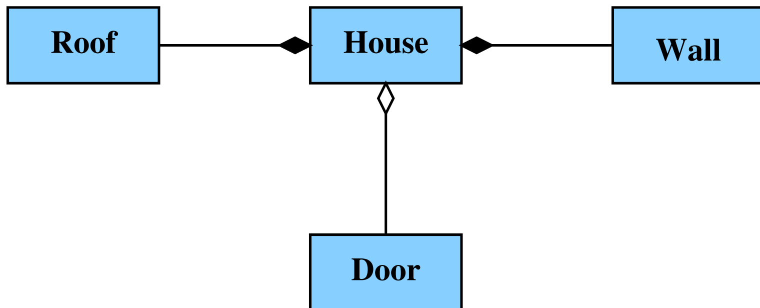
A4–>B4 navigable
but not B4 to A4



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



Constraints on Classes and Associations

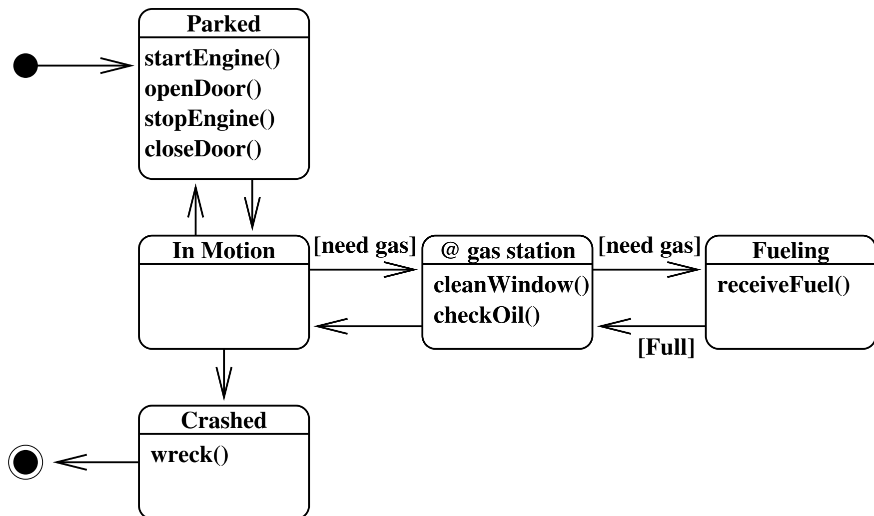
- Constraints wrt object state or association
- Notation: {*constraint*}
- Example constraints on associations: **`{sorted}`**, **`{immutable}`**, **`{read-only}`**, **`{subset}`**, **`{xor}`**
- natural language, pseudo code, predicate logic, . . . , OCL

Statechart Diagrams

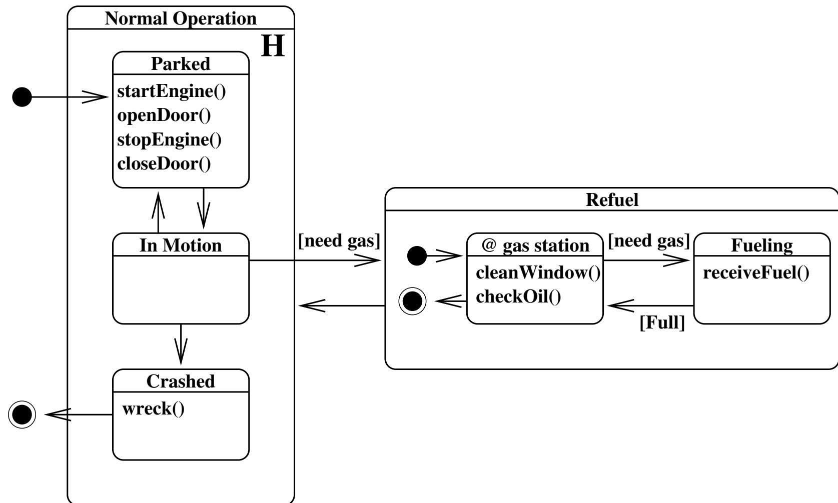
- A statechart diagram is a finite automaton extended with output
  (combinaton of Moore and Mealy automaton)
- Deterministic (Mealy) finite automaton: $(Q, \Sigma, \Lambda, \delta, q_0, F)$
  - Q set of states
  - Σ input alphabet
  - Λ output alphabet
  - $\delta : Q \times \Sigma \rightarrow Q \times \Lambda$ 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

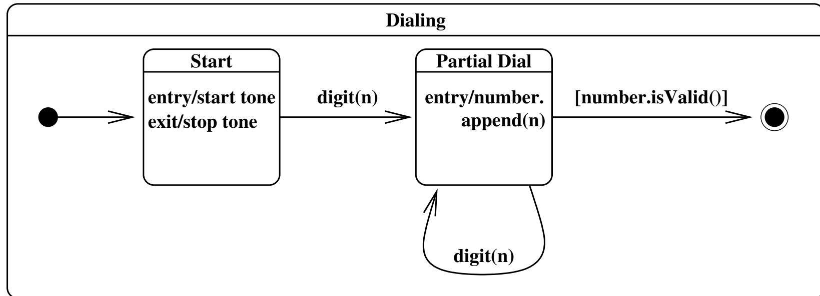
Lifecycle of a Car



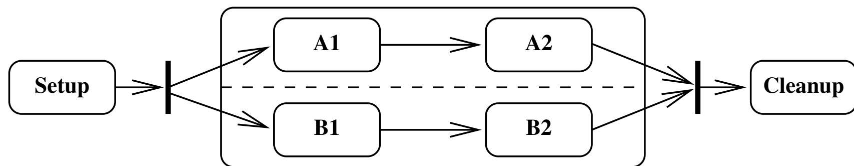
Statechart/Hierarchical States



Statechart/Entry and Exit Actions



Statechart/Concurrent



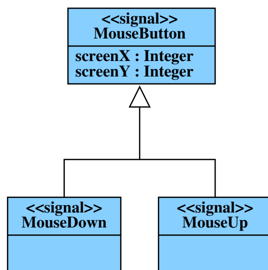
Statechart/Mixed

- Labels on transitions:
  event [*guard*] [/ *method list*]
  - if present, *guard* must be true to trigger the transition
  - free text or OCL
- “Transitions are instanteous”

Statechart/Events

- "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

Statechart/Event Specification



- 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)