UML / UML 2.0 tutorial

Ileana Ober

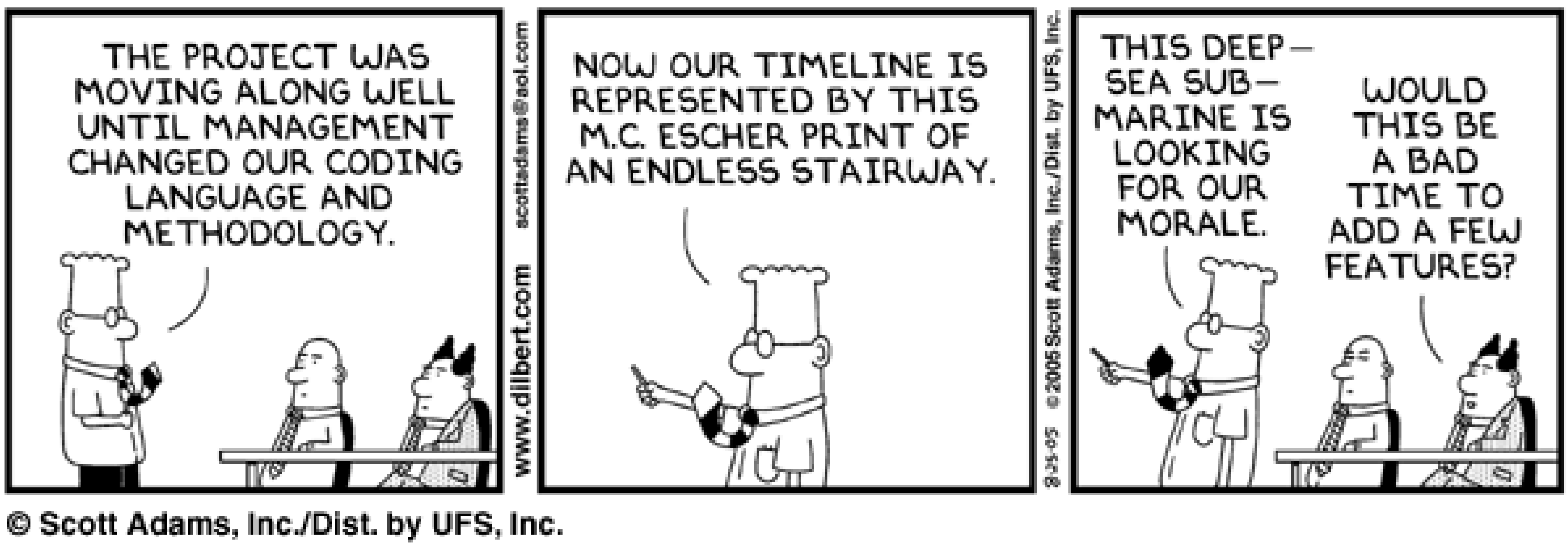
IRIT – UPS, Toulouse, France

http://www.irit.fr/~Ileana.Ober

Modeling in the '80 – '90s

- Lots of (slightly different) languages and design techniques
  - OMT
  - Coad & Yourdon
  - BON
  - SDL
  - ROOM
  - Shlaer Mellor

… Quite a mess



In use with permission from PIB Copenhagen A/S, obtained august 2005

UML

- Sought as a solution to the OOA&D mess
- Aims at
  - Unifying design languages
  - Being a general purpose modeling language

Lingua franca of modeling

Overview

- What is UML?
- Structure description
- Behavior description
- OCL
- UML and tools

Overview

- What is UML?
- Structure description
- Behavior description
- OCL
- UML and tools

UML (Unified Modeling Language)

- Goal: lingua franca in modeling
- Definition driven by consensus rather than innovation
- Standardized by the OMG
- Definition style:
  - Described by a meta-model (abstract syntax)
  - Well formedness rules in OCL
  - Textual description
    - static and dynamic semantics
      (in part already described by WFRs)
    - notation description
    - usage notes

Overview of the 13 diagrams of UML

Structure diagrams

1. Class diagram
2. Composite structure diagram (*)
3. Component diagram
4. Deployment diagram
5. Object diagram
6. Package diagram

Behavior diagrams

7. Use-case diagram
8. State machine diagram
9. Activity diagram

Interaction diagrams

10. Sequence diagram
11. Communication diagram
12. Interaction overview diagram (*)
13. Timing diagram (*)

(*) not existing in UML 1.x, added in UML 2.0

UML principle: diagram vs. model

- Different diagrams describe various **facets** of the model
- Several diagrams of the same kind may coexist
- Each diagram shows a *projection* of the model
- *Incoherence* between diagrams (of the same or of different kind(s)) correspond to an *ill-formed* model
- The coherence rules between different kinds of diagrams is not fully stated

This tutorial looks closer at …

- Use case diagram
- Class diagram
- Composite structure diagram
- Component/deployment diagram
- State machine diagram
- Activity diagram
- Interaction diagrams

Overview

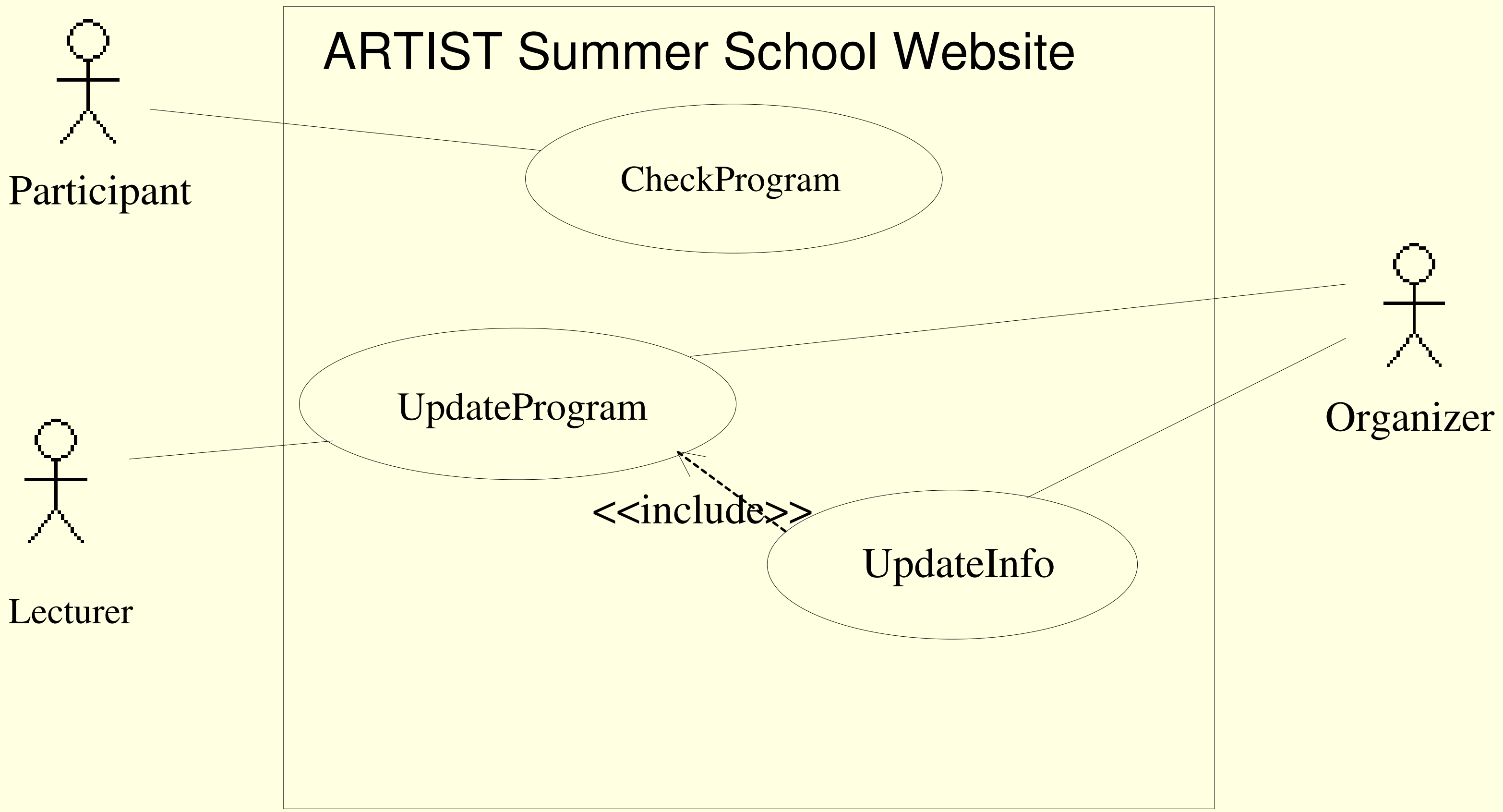
- What is UML?
- Structure description
  - Use case diagram
  - Class diagram
  - Composite structure diagram
  - Communication principles in UML
  - System initialization
- Behavior description
- OCL
- UML and tools

Use case diagram

- Displays the relationship among *actors* and *use cases*, in a given *system*

- Main concepts:
  - System – the system under modeling
  - Actor – external “user” of the system
  - Use case – execution scenario, observable by an actor

Use case diagram example



Use case diagram – final remarks

- Widely used in real-life projects
- Used at:
  - Exposing requirements
  - Communicate with clients
  - Planning the project
- Additional textual notes are often used/required

- User-centric, non formal notation
- Few constraints in the standard

Further reading:

D. Rosenberg, K.Scott Use Case Driven Object Modeling with UML : A Practical Approach, Addison-Wesley Object Technology Series, 1999

I. Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley Professional, 1999

Writing Effective Use Cases Alistair Cockburn Addison-Wesley Object Technology Series, 2001

Overview

- What is UML?
- Structure description
  - Use case diagram
  - Class diagram
  - Composite structure diagram
  - Communication principles in UML
  - System initialization
- Behavior description
- OCL
- UML and tools

Class diagram

- The *most known* and the *most used* UML diagram
- Gives information on model's structural elements
- Main concepts involved
  - Class - Object
  - Inheritance
  - Association (various kinds of)

Let's start with … object orientation

- Why OO?
  - In the first versions, UML was described as addressing the needs of modeling systems in a OO manner
  - Statement not any longer maintained, however the OO inspiration for some key concepts is still there
- Main concepts:
  - Object – individual unit capable of *receiving/sending messages*, processing data
  - Class – pattern giving an abstraction for a set of objects
  - Inheritance – technique for reusability and extendibility

Further reading:

Bertrand Meyer: Object-Oriented Software Construction, 2nd edition, Prentice Hall, 2000

UML Class

- Gives the type of a set of objects existing at run-time
- Declares a collection of methods and attributes that describe the structure and behavior of its objects
- Basic notation:

Automobile
<u>wheelsNO : integer</u> serialNo : integer
fillTank()

Properties of UML classes

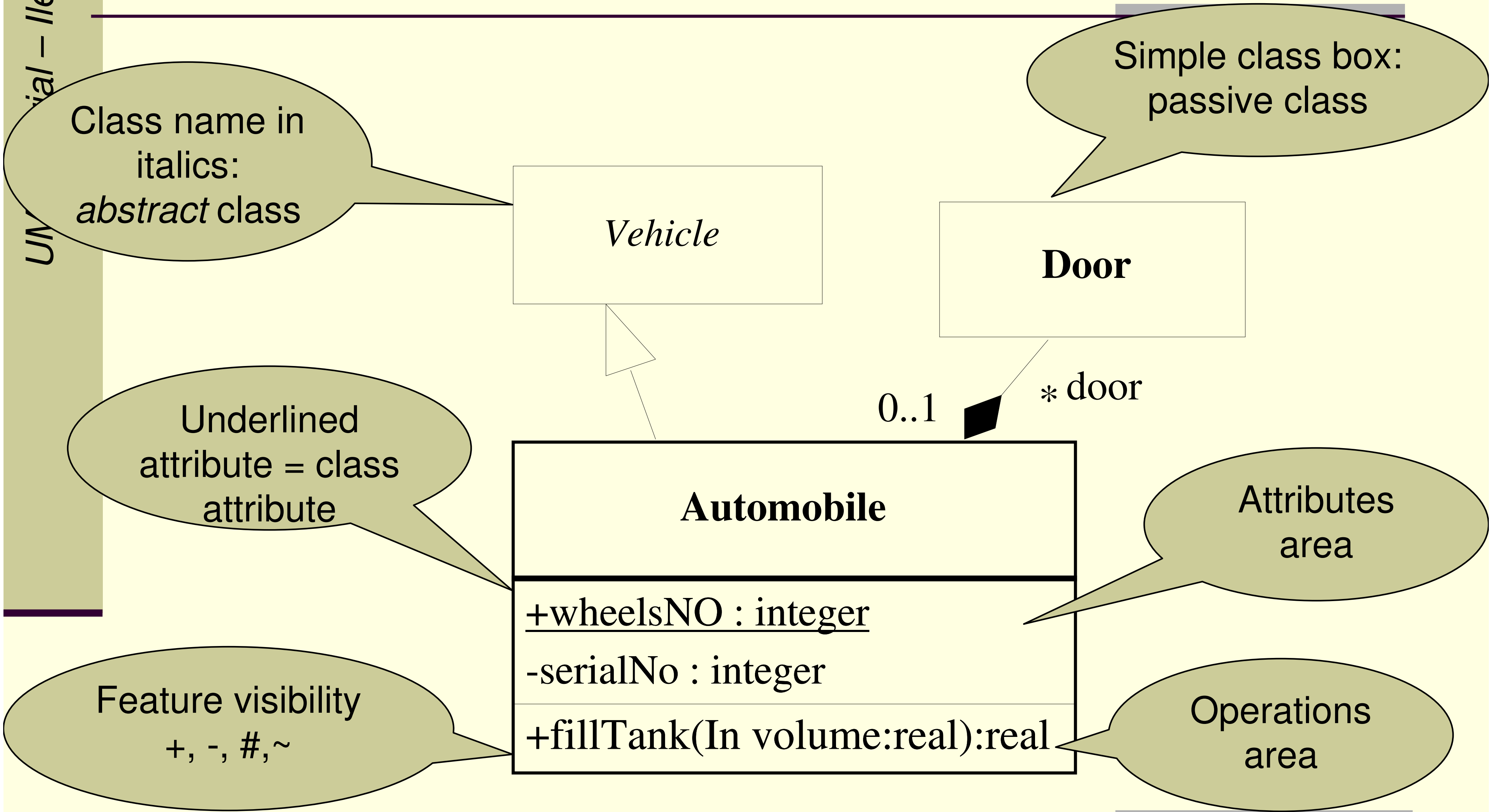
- May own *features*
  - *Structural* (data related) *:* attributes
  - *Behavioral :* operations
- May own *behavior* (state machines, interactions, …)
- May be instantiated
  - except for abstract classes that *can NOT be directly instantiated* and exist only for the inheritance hierarchy

Class features – characterized by

- Signature
- Visibility (public, private, protected, package)
- Changeability (changeable, frozen, addOnly)
- Owner scope (class, instance) – equivalent to `static` clause in programming languages
- Invariant constraint

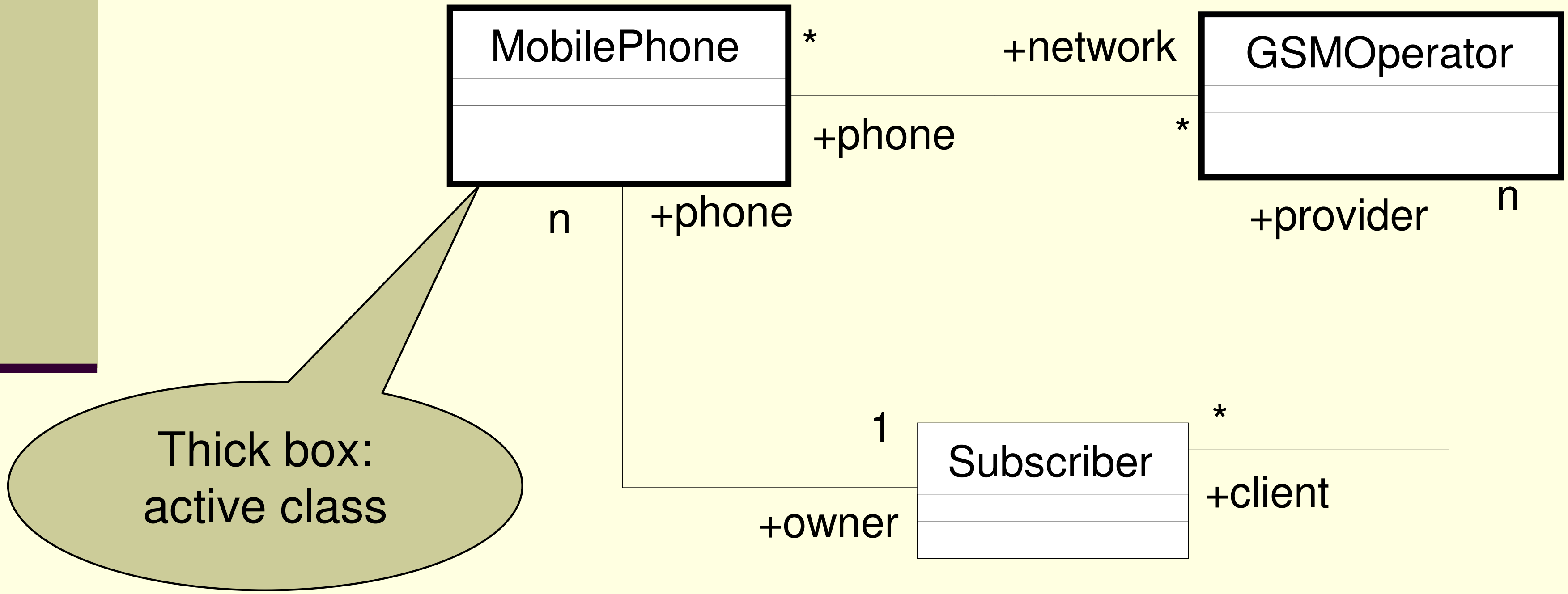
- Additionally, operations are characterized by
  - *concurrency kind*: sequential, guarded, *concurrent*
  - *pre* or *post* conditions
  - body (state machine or action description)

Decode class symbol adornments



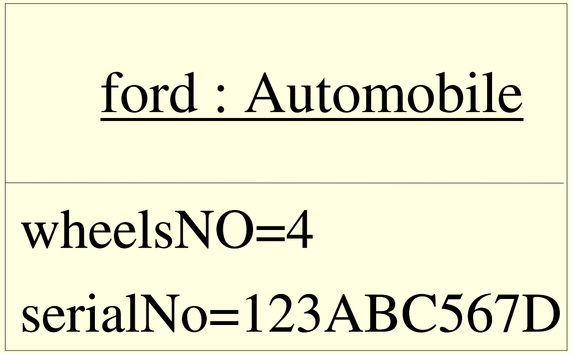
Active / passive classes

- specifies the *concurrency model* for classes
- specifies whether an ***Object*** of the ***Class*** maintains its own thread of control and runs concurrently with other active ***Objects*** (active)



Object

- Instance of a class
- Can be shown in a class diagram
- Notation

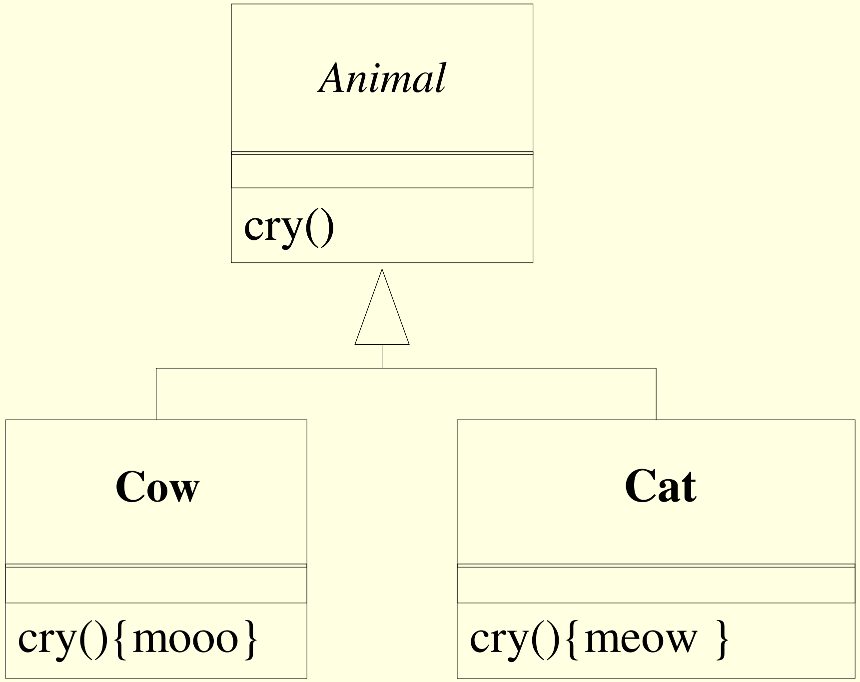


Inheritance

- A.k.a. generalization (specialization)
- Applies mainly on classes
- Other UML model elements can be subject to inheritance (e.g. interface)
  (if you want the exact list go check the UML metamodel for kinds of *GeneralizableElement*s)
- Allows for polymorphism

Inheritance/polymorphism example

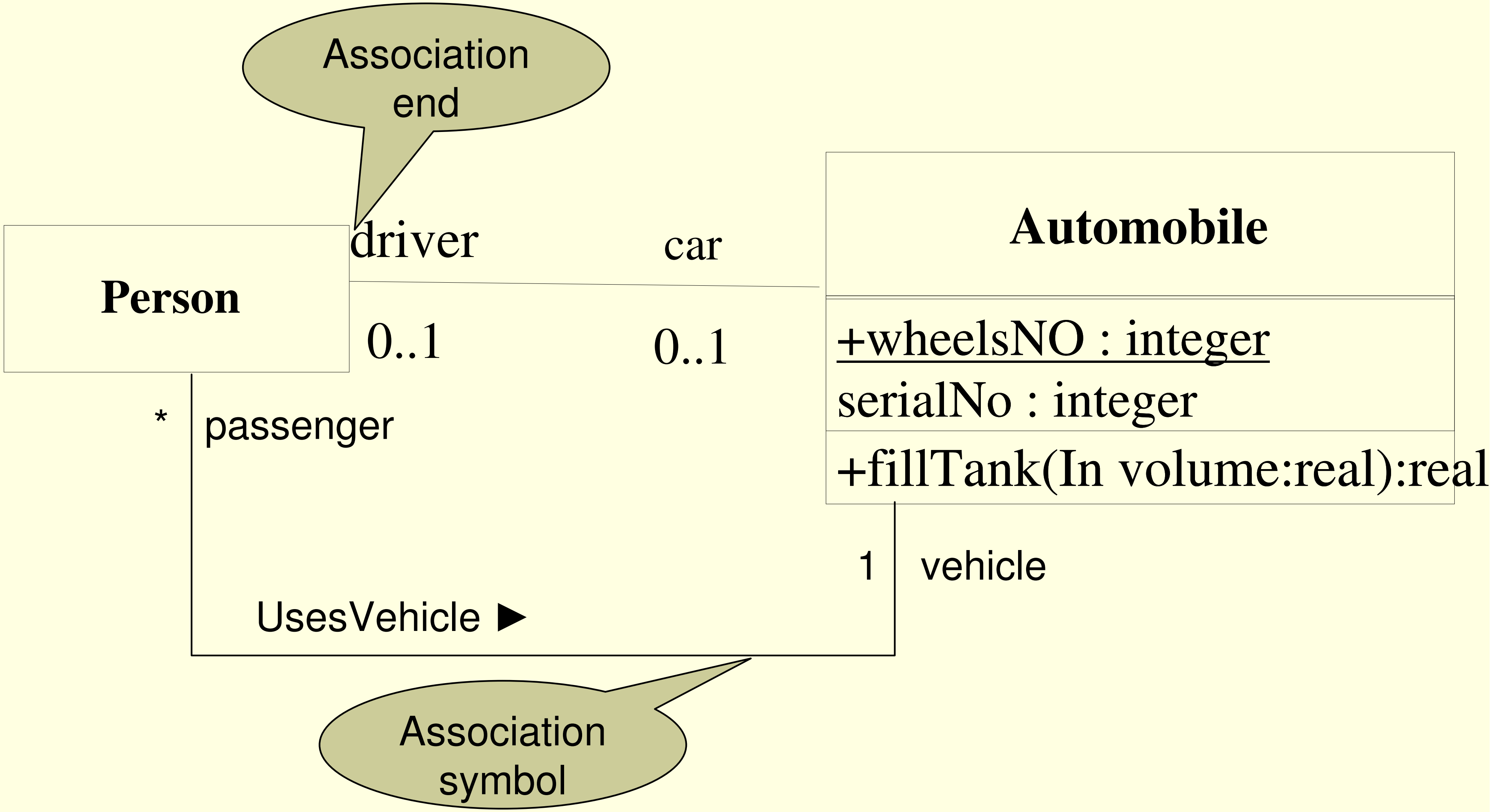
```
Animal a;
Cow cw;
Cat ct;
…..
if (<condition>)
    a:= cw
else
    a:= ct
endif

a.cry()
 --- should be a mooo or a meow
    depending on the <condition>
```



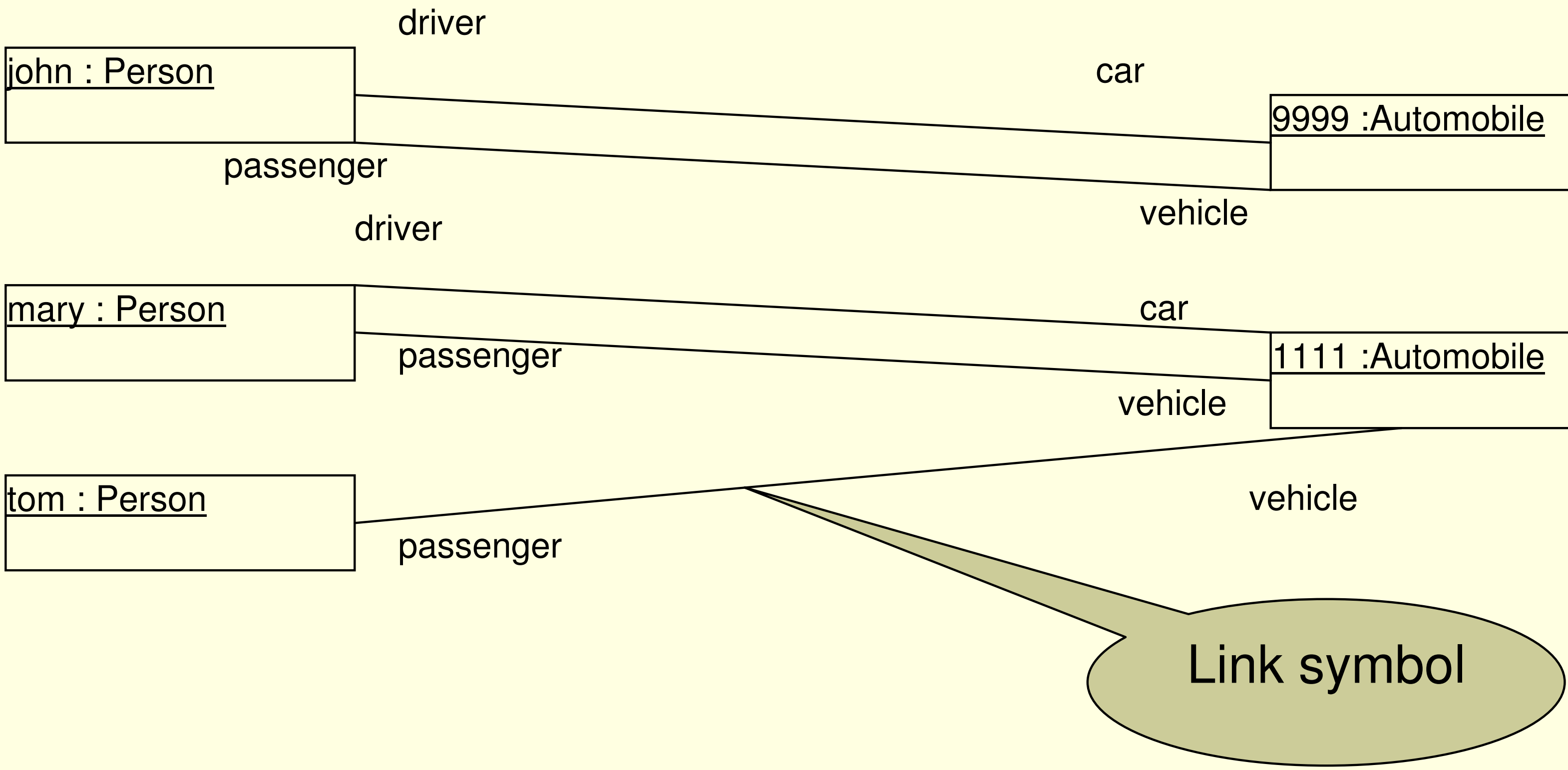
Association

- Concept with no direct equivalent in common programming languages
- Is defined as a semantic relationship between classes, that can materialize at runtime
- The *instance* of an association is a *set of tuples relating instances* of the classes
- It's actual nature may vary, in terms of code, they may correspond to
  - Attributes, pointers
  - Operations
  - Nothing (i.e. graphical comments)

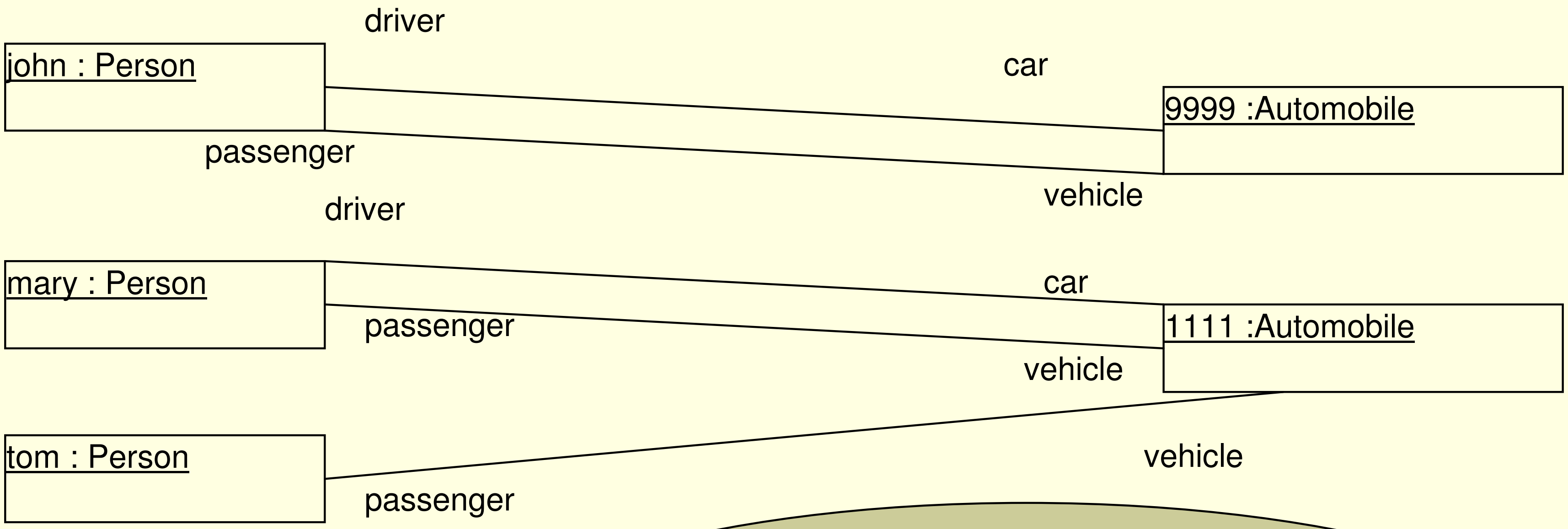
Example


Association
end
driver
car
Person
Automobile
0..1
0..1
+wheelsNO : integer
serialNo : integer
+fillTank(In volume:real):real
*
passenger
1
vehicle
UsesVehicle ▶
Association
symbol

Example – at instance level



Example – at instance level



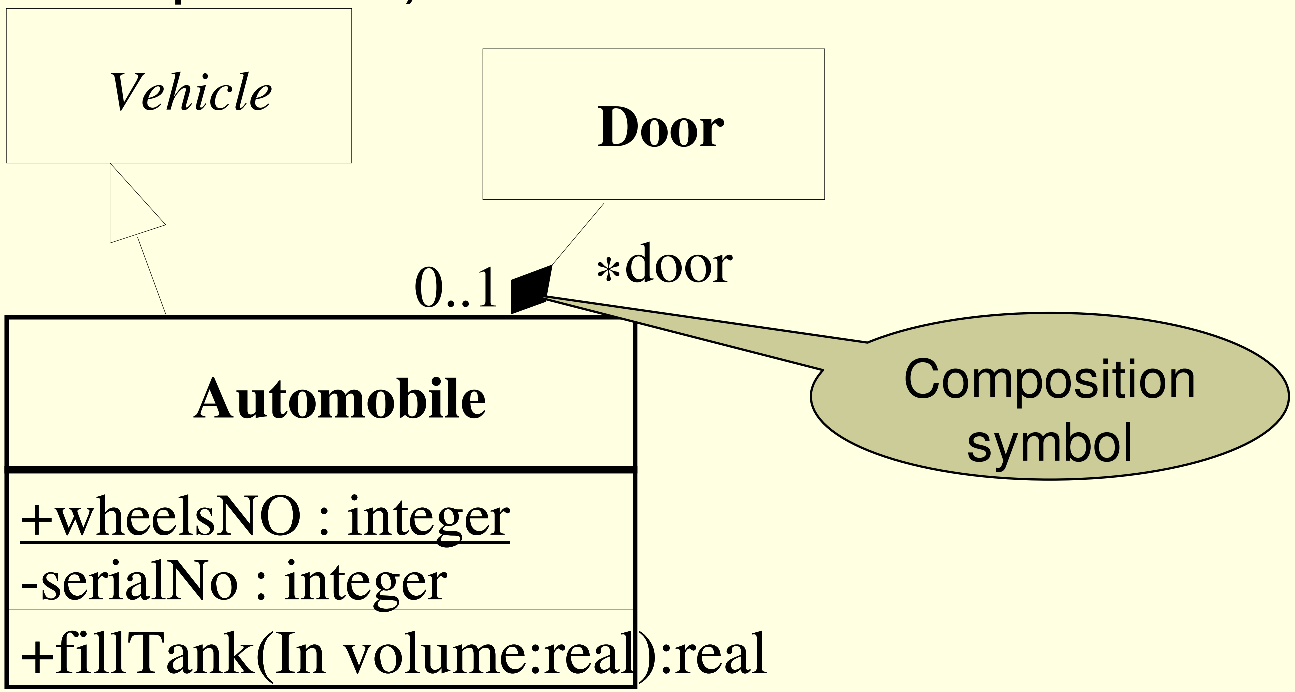
Note on style in UML diagrams:
Instance level names: lower case
Type level names: upper case

Association end

- Endpoint of an association
- Characterized by a set of properties contributing to the association definition
  - Multiplicity (ex: 1, 2..7, *, 4..*)
  - Ordering ordered/unordered
  - Visibility +,-,#, ~
  - Aggregation…

Various kinds of associations (1/2)

- Regular association
- Composition: one class *is owned (composed in)* the associated class
  Composition implies lifetime responsibility (based on association end multiplicities)

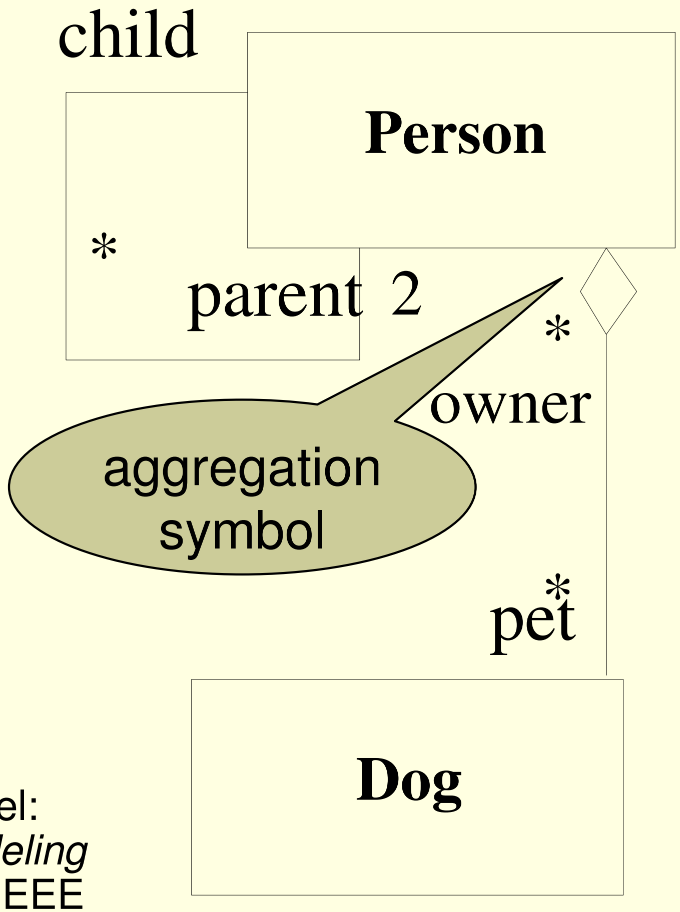


Various kinds of associations (2/2)

Aggregation

"light" composition, semantics left open, to be accommodated to user needs

As it is, it has no particular meaning…

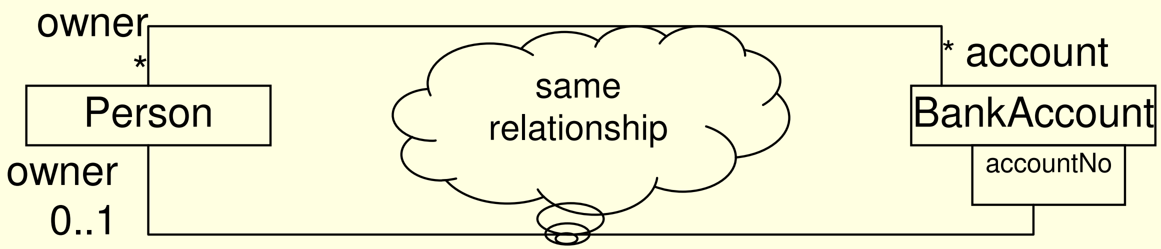


Further reading:

F.Barbier, B.Henderson-Sellers, A.Le Parc-Lacayrelle, J.-M.Bruel: *Formalization of the Whole-Part Relationship in the Unified Modeling Language*, IEEE Transactions on Software Engineering, 29(5), IEEE Computer Society Press, pp. 459-470, 2003

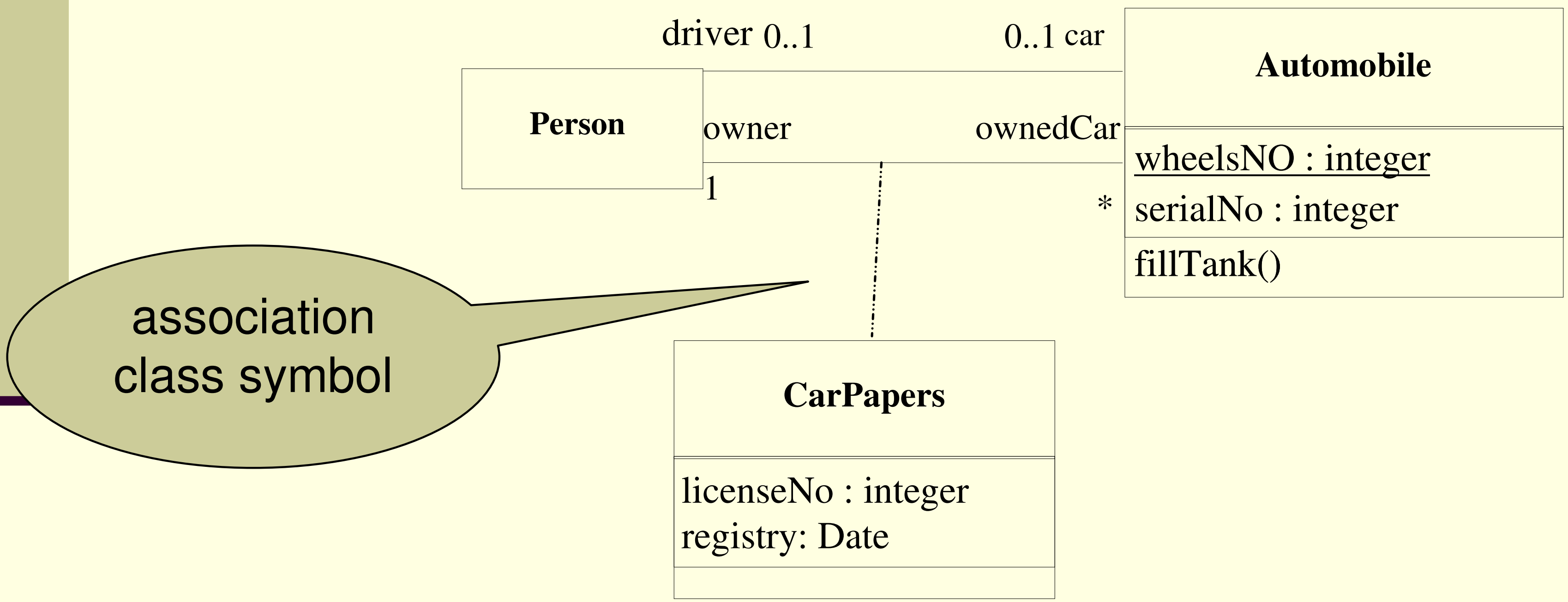
More on associations…

- Associations may be n-ary (n>2)
- *Qualifiers* – partition the set of objects that may participate in an association



Association class

- An association that is also a class.
- It defines a set of *features* that belong to the *relationship* itself and not any of the classifiers.



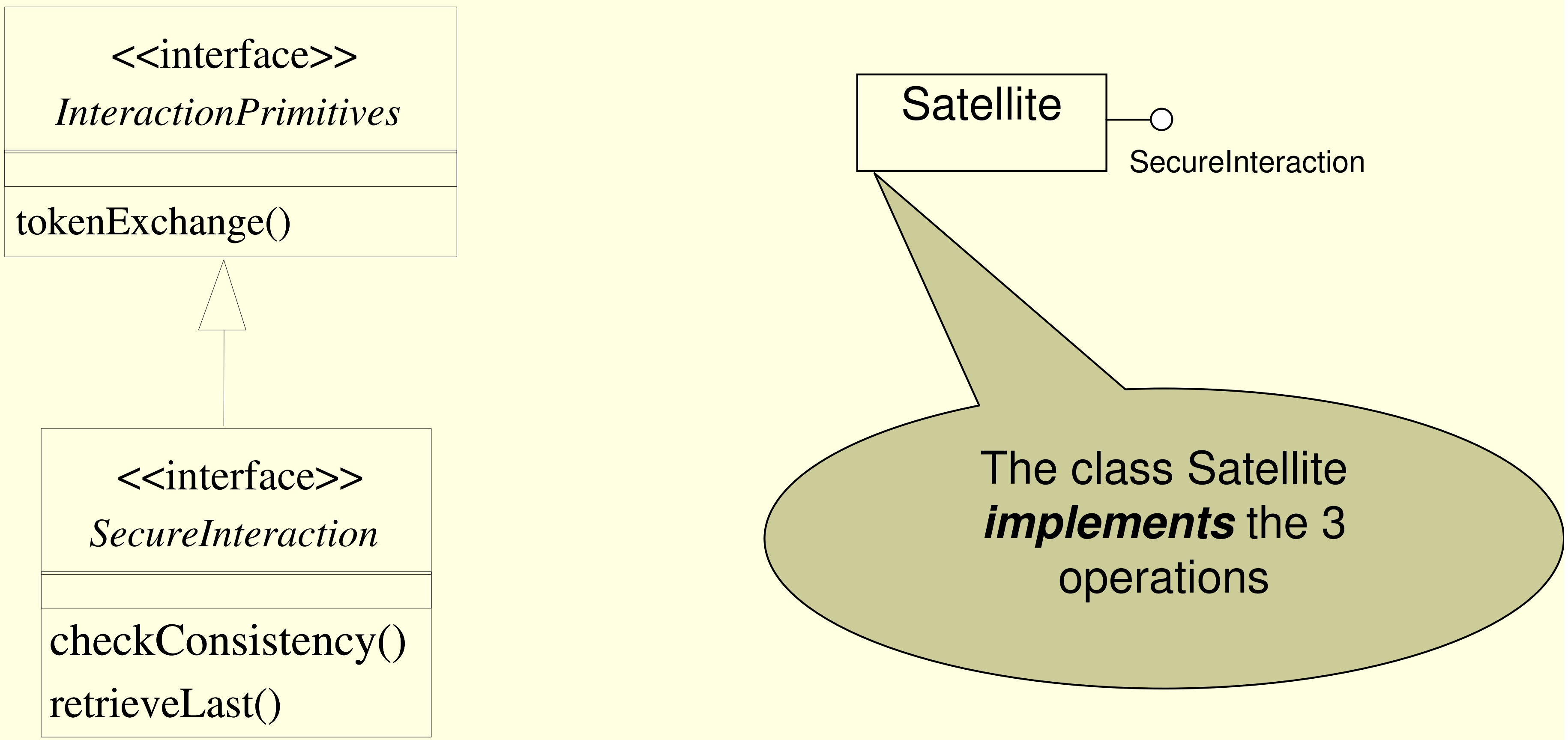
Other elements of class diagrams

- Interface (definition and use)
- Templates
- Comments

Interface

- Declares set of public features and obligations
- Specifies a contract, to be fulfilled by classes implementing the interface
- Not instantiable, required or provided by a class
- Its specification can be realized by 0, 1 or several classes
  - the class presents a public facade that conforms to the interface specification
    (e.g. interface having an attribute does not imply attribute present in the instance)
  - a class may implement several interfaces
- Interfaces hierarchies can be defined through inheritance relationships

Interface definition and use examples



Means to specify the interface contract

- Invariant conditions
- Pre and post conditions (e.g. on operations)
- Protocol specifications

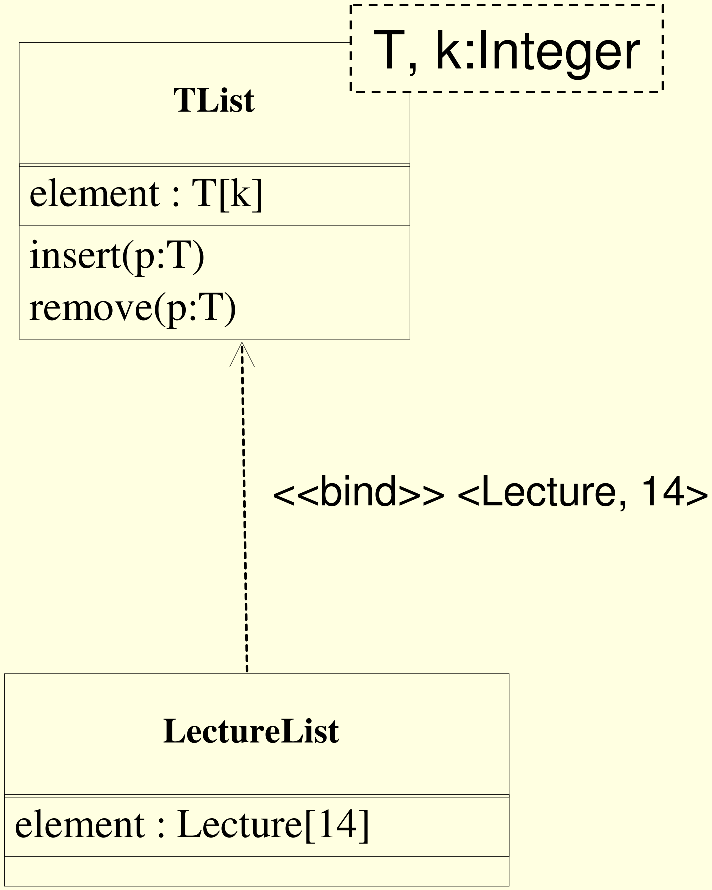
  which may impose ordering restrictions on interactions through the interface

  for this one may use *protocol state machines*

Templates

- Mechanism for defining patterns whose parameters represent types
- It applies to classifiers, packages, operations
- A template class is a template definition
  - Cannot be instantiated directly, since it is not a real type
  - Can be bound to an actual class by specifying its parameters
- A bound class is a real type, which can be instantiated

Template example



Class diagram summary

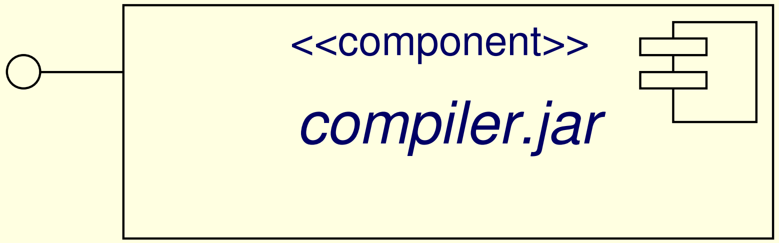
- The most used diagrams
- Describes the static structure of the system in terms of classes and their relationships (associations, inheritance)
- Offers connection points with the UML behavior description means

Overview

- What is UML?
- Structure description
  - Use case diagram
  - Class diagram
  - Components
  - Communication principles in UML
  - System initialization
- Behavior description
- OCL
- UML and tools

Component

- Its definition evolves from UML 1.x to UML 2.0
- In UML 1.x - deployment artifacts



- In UML 2.0 – structured classes

Component in UML 2.0

- Modular part of a system encapsulating its content
- Defines its behavior in terms of provided and required interfaces, and associated contracts
- Defines a type. Type conformance is defined on the basis of conformance to provided / required interfaces
- Main property: substitutability = ability to transparently replace the content (implementation) of a component, provided its interfaces and interface contracts are not modified

Component examples (1/2)

- Algorithmic calculus component
  - Interface:
    - Offered: provided mathematical calculus functions
    - Required : logarithm value calculus
  - Contract
    - Expected behavior
    - Constraints on unauthorized values

Component examples (2/2) : mobile phone logical network

Sample component: virtual cell manager

- Interface:
  - Manage reachable mobile phones
  - Forward message calls
  - …
- Contracts:
  - Functional
    - Fulfill expected behavior
    - Protocol describing authorized message exchange: (e.g. first identify)
  - Non-functional
    - Net load capacity, reactivity time, electromagnetic interference…

Component related concepts

- Class
- Package
- (Library)

The exact relationship between all these concepts is not completely clear (neither in UML, nor in the literature)

UML offers a unifying concept… classifier

- Generalization of the class concept
- Gives a type for a collection of instances sharing common properties
- Interfaces, classes, data types, components

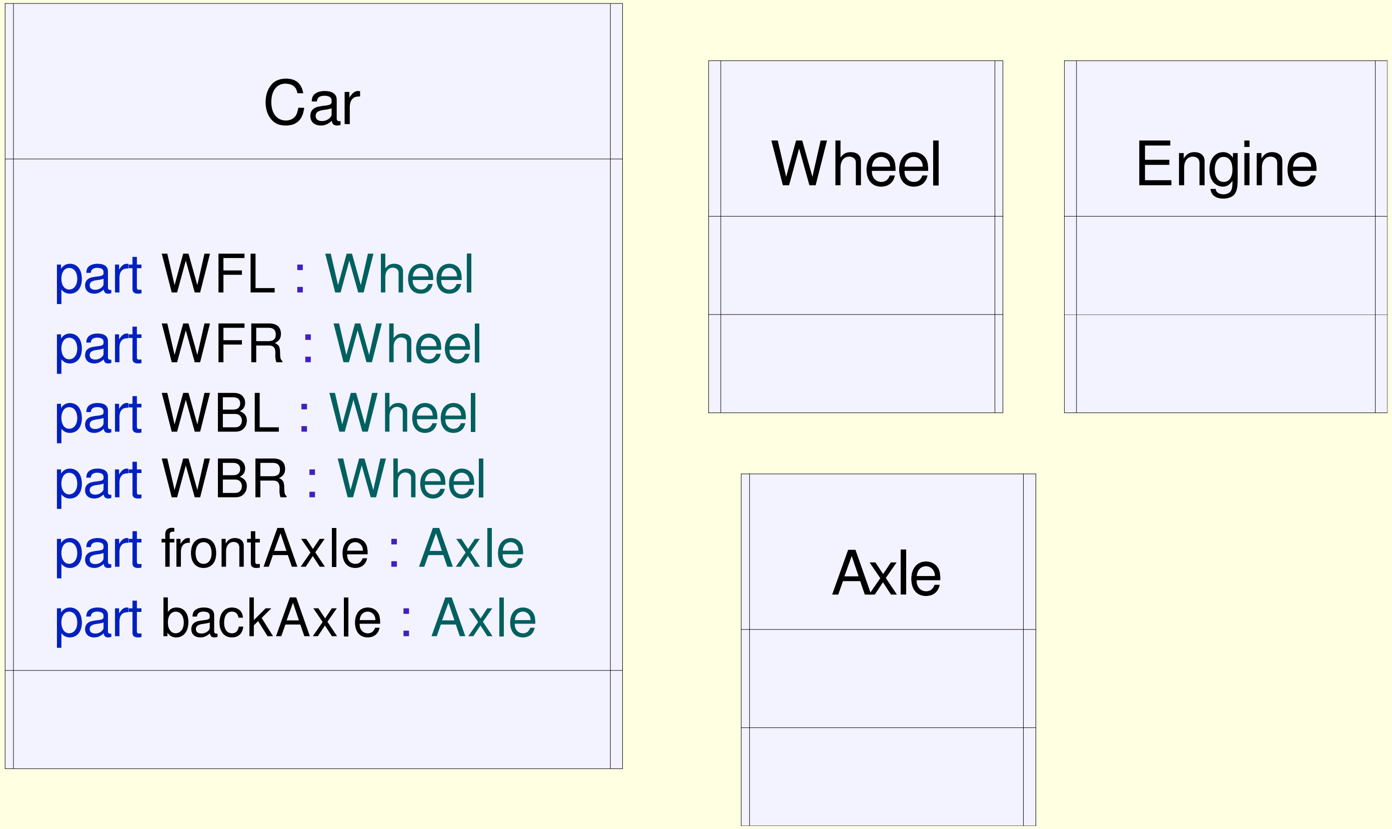
Composite structure diagram (a.k.a. architecture diagram)

- Added in UML 2.0
- Depicts
  - The internal structure of a classifier
  - Interaction points to other parts of the system
  - Configuration of parts that perform together the behavior of the containing classifier
- Concepts involved:
  - Classifier
  - Interface
  - Connection
  - Port
  - Part

Part

- Element representing a (set of) instance owned by a classifier
- Semantics close to the one of attributes or composed classes
  - May specify a multiplicity
  - At parent creation time, parts may need to be created also
  - When the parent is destroyed, parts may need to be destroyed also

Example


Car
part WFL : Wheel
part WFR : Wheel
part WBL : Wheel
part WBR : Wheel
part frontAxle : Axle
part backAxle : Axle
Wheel
Engine
Axle

Abstraction level for part

- Somewhere between instance and type…
- WFL characterizes the wheels front left, owned by Car instances
- *Given a Car class instance*, the part WFL is an instance of its front right wheel
- *If no Car class instance is fixed, the part WFL is an instance abstraction* generically characterizing front right wheels of Cars

Port

- feature of a classifier specifying a distinct interaction point
  - between that classifier and its environment (service port)
  - between the behavior of the classifier and its internal parts (behavior port)

- characterized by a list of required and provided interfaces
  - Required interfaces describe services the owning classifiers *expect* from environment and may access via this interaction point
  - Provided interfaces describe services the owning classifiers *offer* to its environment via this interaction point

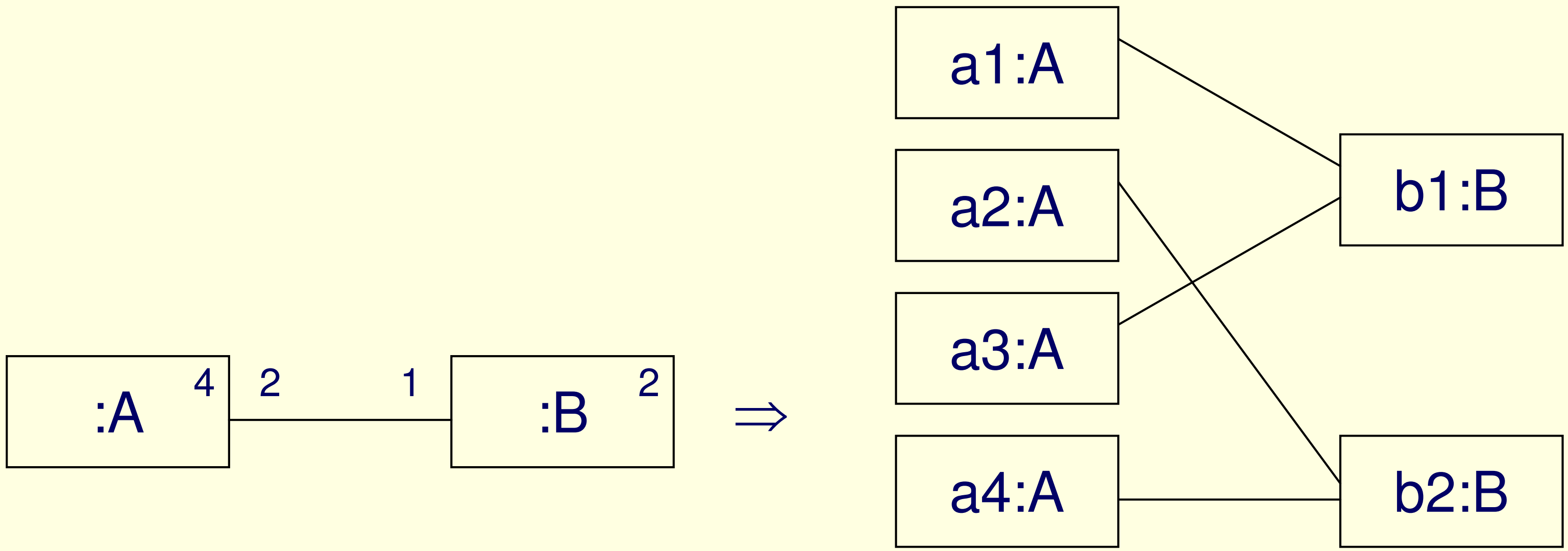
- *an instance may differentiate between invocations of a same operation received through different ports*

Connector

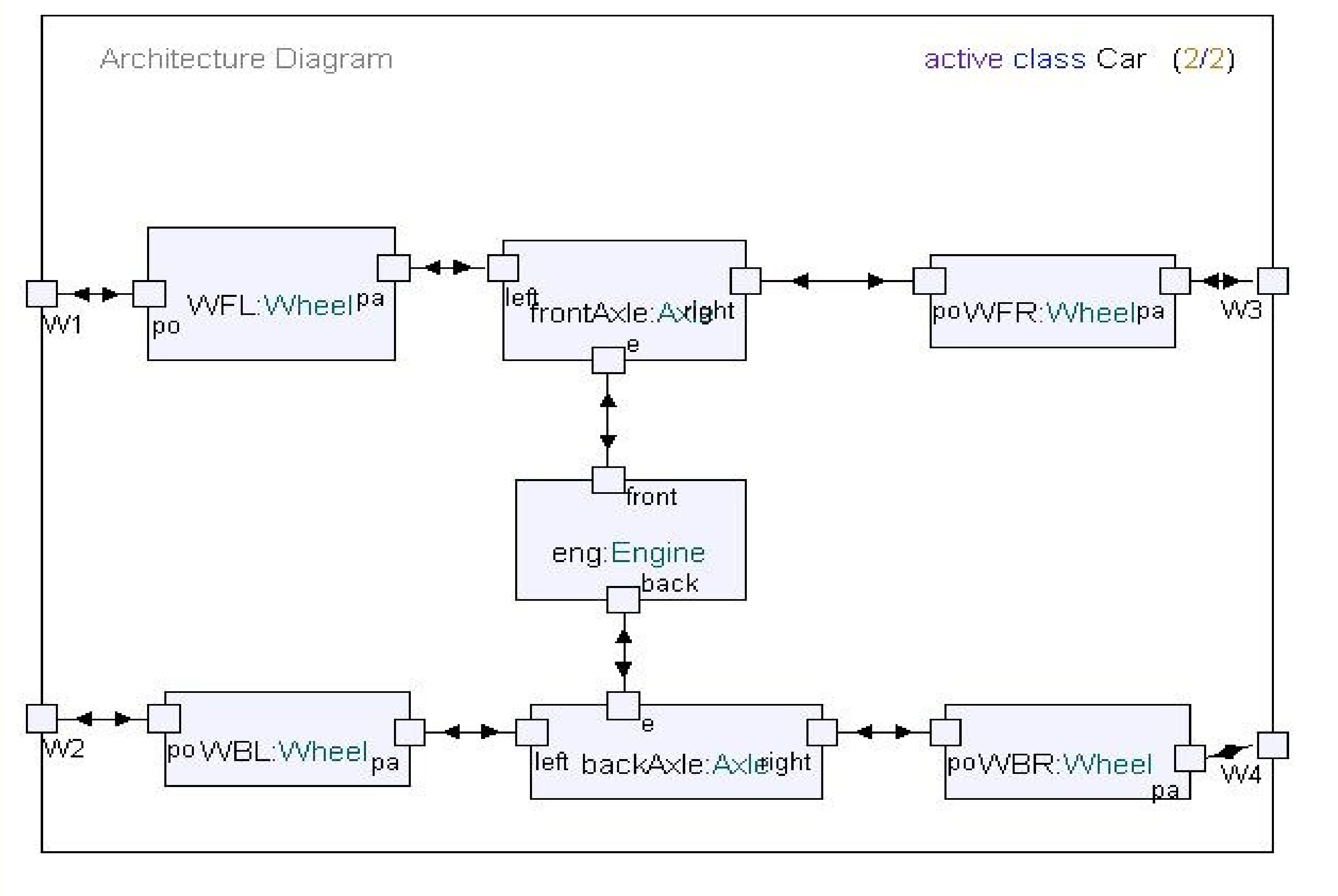
- Link enabling communication btw instances
- It's actual realization is not specified
  (simple pointer, network connection, …)
- It has two connector ends, each playing a distinct role
- The communication realized over a connector may be constrained
  (type constraint, logical constraint in OCL, etc)

Communication architecture

- complex multiplicity → need for initialization rules



Example: composite structure diagram



Port vs. interface

- Interface – signature
- Port – interaction point

- Interfaces describe what happens at a port
- The same interface may be attached to several ports of a component

Port constraints vs. interface constraints

- Constraints may be attached to both ports and interfaces
- For both, constraints can take the form of
  pre and post conditions, invariants, protocol constraints

- Nothing is stated on how constraints at various levels should be composed
- By default, constraint conjunction

- More elaborated constraint handling schemes may be imposed by the methodology

Connector vs. link

- Link = association instance
  - Data oriented
  - May be attached to any instance of the corresponding classifier
- Connector
  - Behavior (communication) oriented
  - Can only be connected to particular instances
  - Instance to which it applies are depicted in the composite structure diagram

Overview

- What is UML?
- Structure description
  - Use case diagram
  - Class diagram
  - Components
  - Communication principles in UML
  - System initialization
- Behavior description
- OCL
- UML and tools

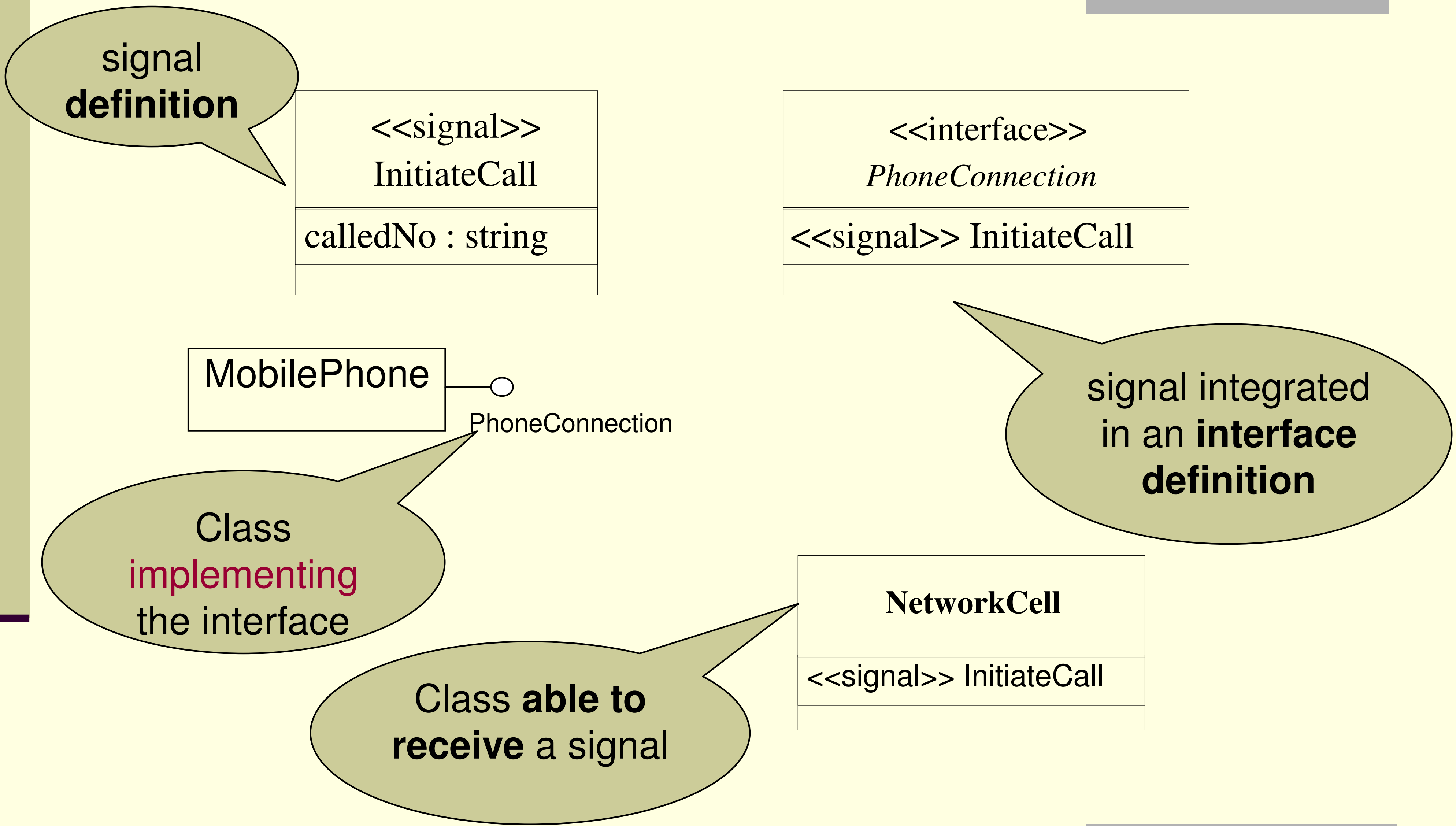
Communication

- Communication primitives
- Communication schema

Communication primitives

- **Signal**
  - One way
  - Asynchronous communication primitive
  - May carry data
  - It is defined independently of the classifiers handling it
- **Operation call**
  - Two-way communication primitive (call-reply)
  - The caller is blocked
  - May carry data
  - Typically, it has a target object
- **Queue**
  - Communication buffer
  - May be attached to instances
  - Management policy not constrained

Signal definition and use examples



Communication schema in UML

- If the *model says noting on communication* (i.e. no connectors exist)
  - Point to point: between objects knowing their ID (due to existing associations, passed as parameter in some operation, etc)
  - Broadcast: to listening and accessible objects

- If a *communication structure is stated* (architecture diagram) - the communication obeys its constraints

  communication paths, connectors chain, conveyed messages, port constraints etc…

Overview

- What is UML?
- Structure description
  - Use case diagram
  - Class diagram
  - Components
  - Communication principles in UML
  - System initialization
- Behavior description
- OCL
- UML and tools

System initialization

- What it is?
  - The mechanism that gives the initial status of the system
- How it can be done?
  - Using a ***God*** object that creates the whole system
  - Using an initialization script
  - Based on a particular object diagram giving the snapshot of the system at initialization time
- How it is in the standard?
  - No standard mechanism exists

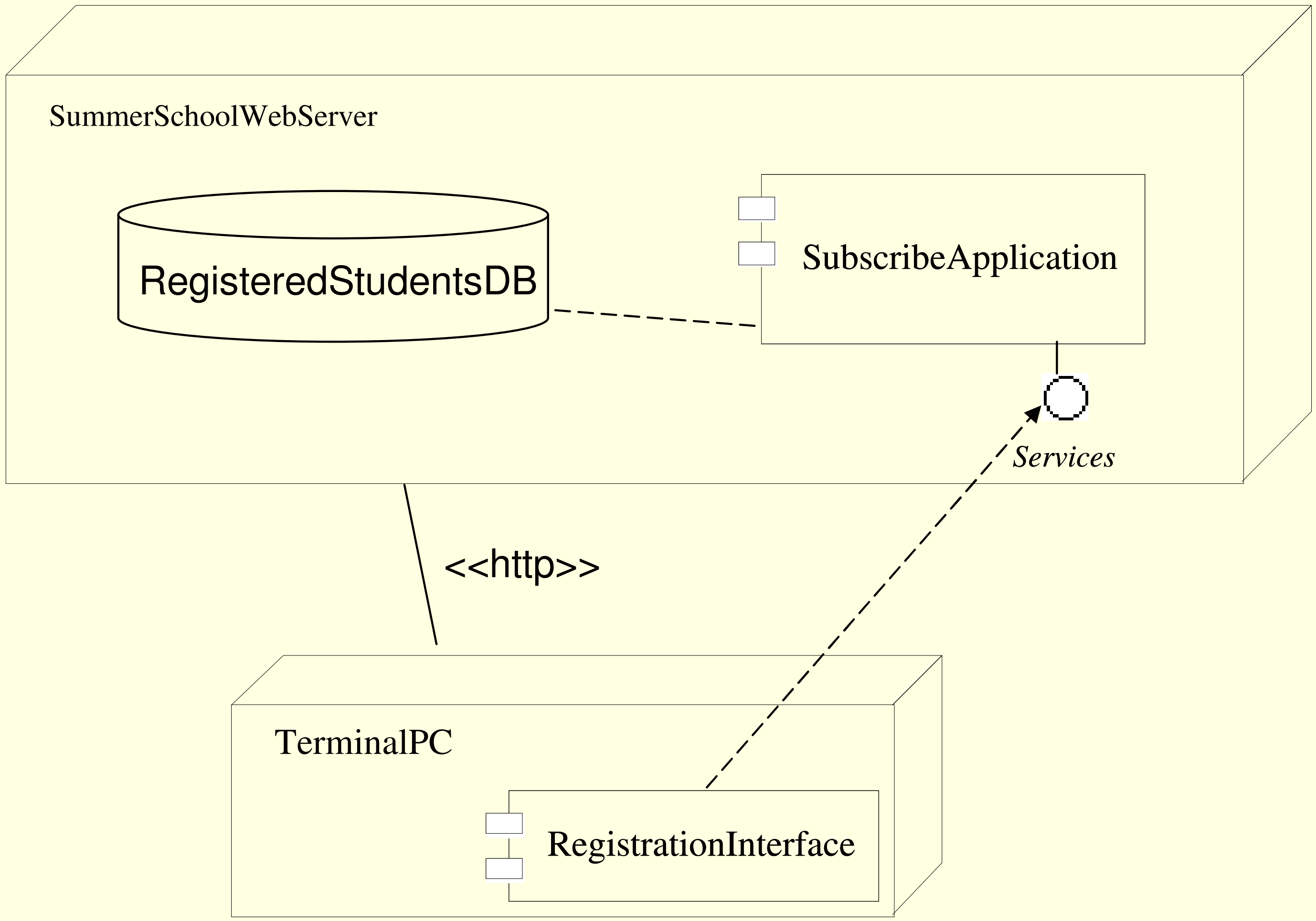
Going forward in component based modeling …

- The actual “wiring” of components is designed using *component* and *deployment diagrams*

- Component diagrams
  - Models business and technical software architecture
  - Uses components defined in the composite structure diagrams, in particular their ports and interfaces

- Deployment diagrams
  - Models the physical software architecture, including issues such as the hardware, the software installed on it and the middleware
  - Gives a static view of the run-time configuration of processing nodes and the components that run on those nodes

Deployment diagram example



Overview

- What is UML?
- Structure description
- Behavior description
  - State machine diagrams
  - Protocol state machine
  - Activity diagrams
  - Sequence diagrams
  - Timing diagrams
- OCL
- UML and tools

Specifying behavior in UML

	specification	description
System	Use case Sequence diagram Invariants	State machine
Class	Sequence diagram Invariants Protocol state machine	State machine
Operation	Pre-condition Post-condition Invariants Protocol state machine	State machine Actions

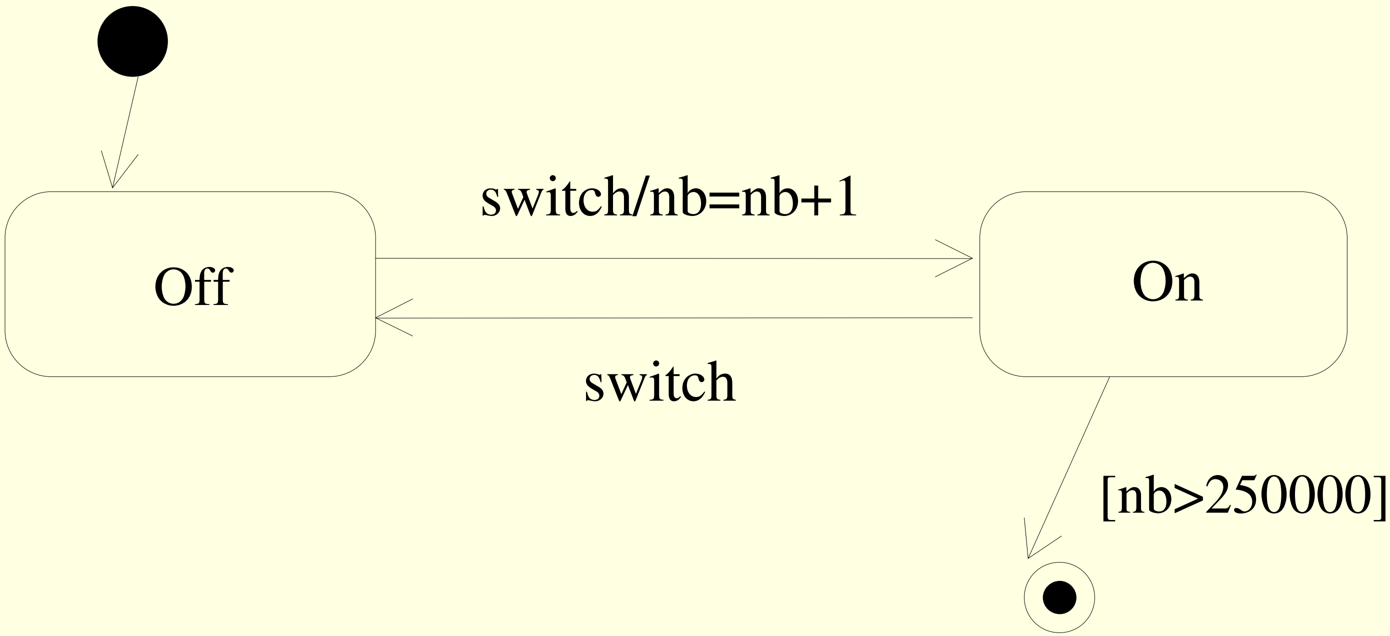
State machine

- UML finite state automaton
- Behavior description mechanism
- Describes the behavior for:
  - System
  - Class
  - Operation

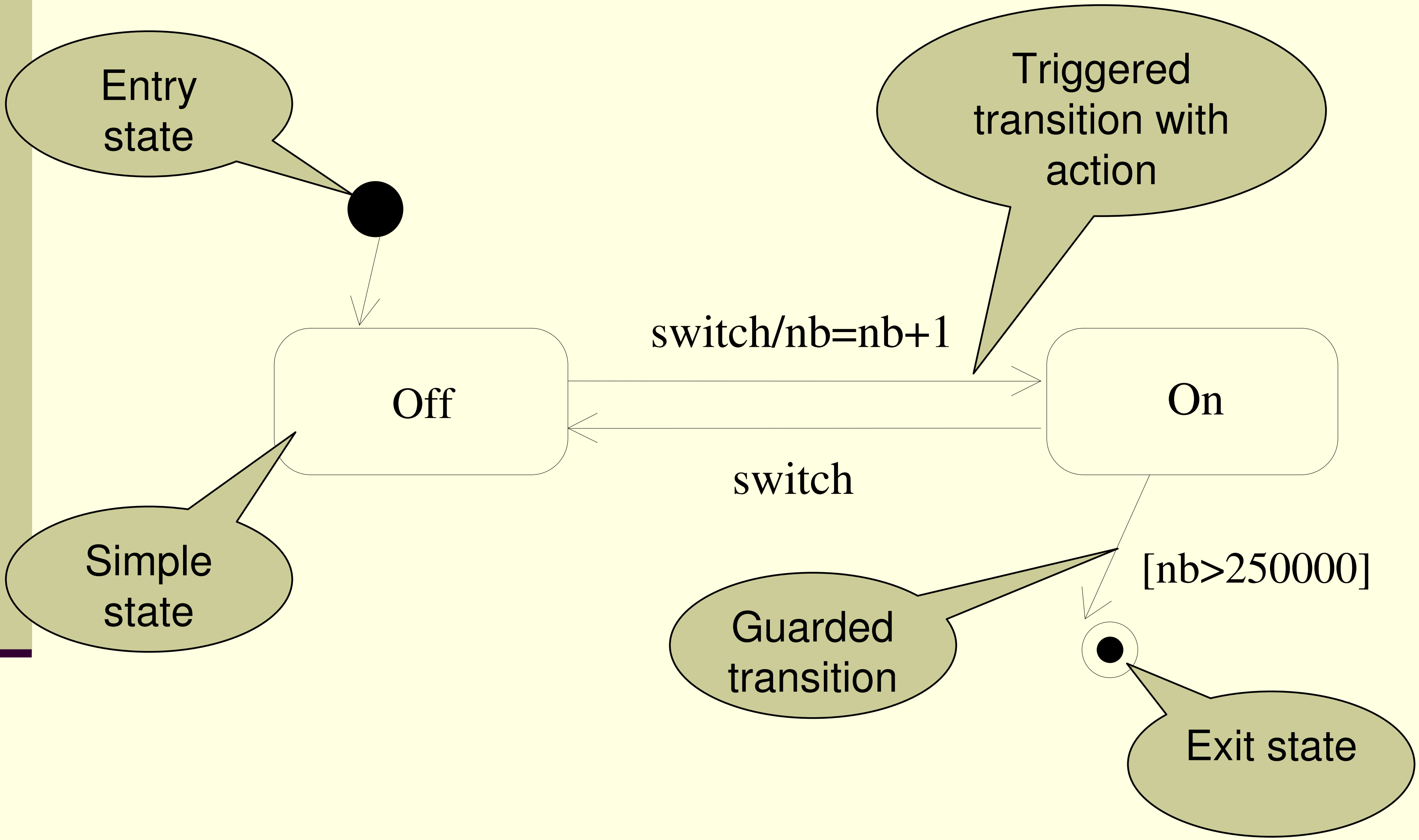
Main concepts

- State – stores information of the system (encodes the past)
  Particular states
  - Initial state (?)
  - Final state
- Transition – describes a state change
  - Can be triggered by an event
  - Can be guarded by a condition
- Actions – behavior performed at a given moment
  - Transition action : action performed at transition time
  - Entry action : action performed when entering a state
  - Exit Action : action performed when exiting a state
  - Do Action : action performed while staying in a state

Simple state machine example



Simple state machine example



Event

- “Specification of some occurrence that may potentially trigger effects by an object”

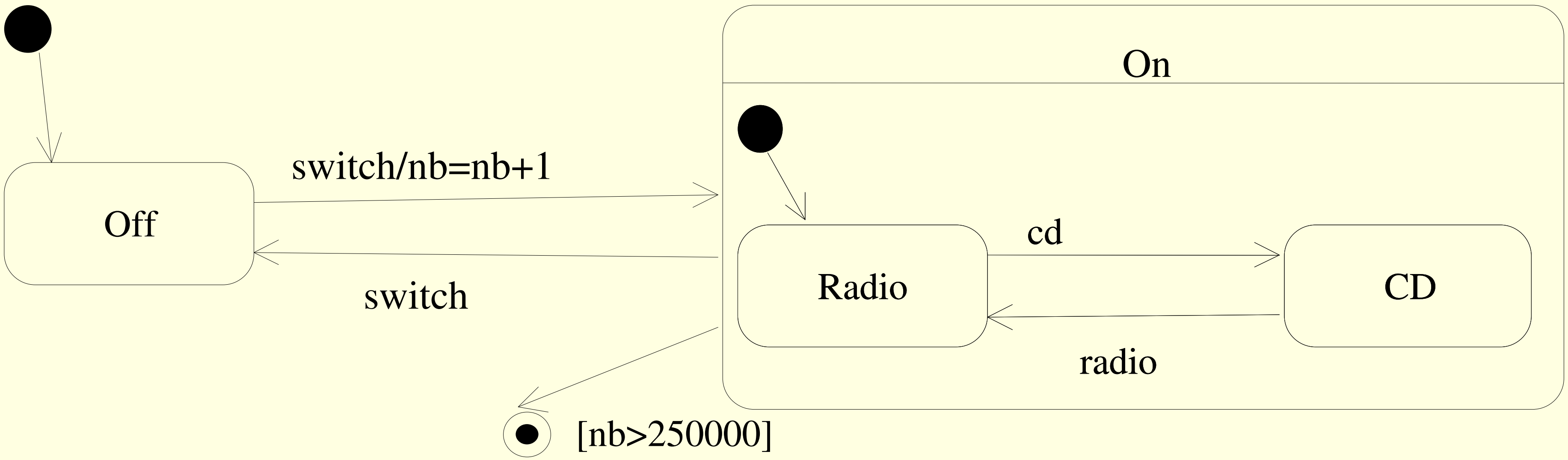
Typically used in StateMachines as triggers on transitions

- Examples (as defined in the standard): SignalEvent, CallEvent, ChangeEvent, TimeEvent, etc.
- Notion refined in the SPT profile

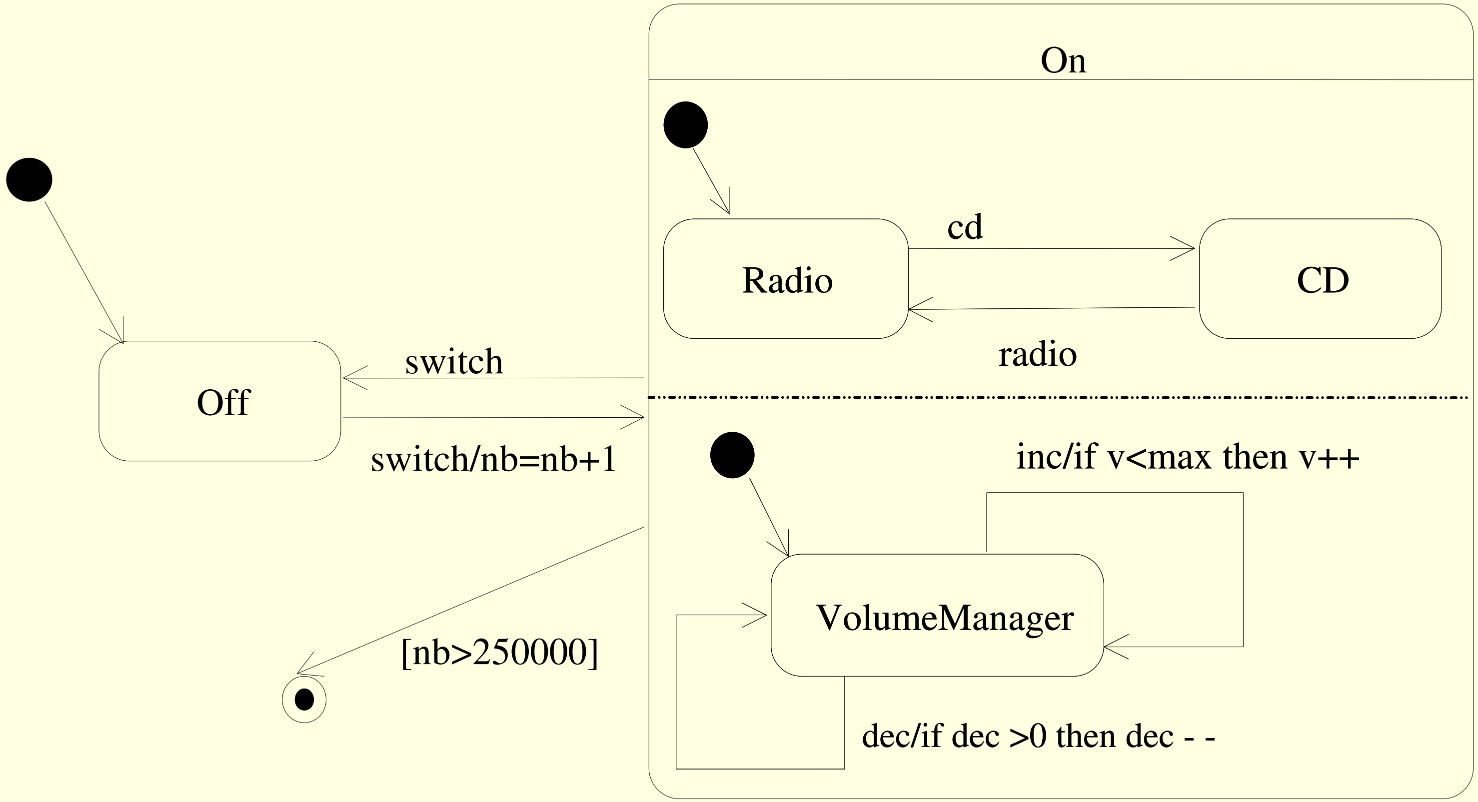
Hierarchical states

- All states are at the same level => the design does not capture the commonality that exists among states
- Solution: Hierarchical states – described by sub-state machine(s)
- Two kinds of hierarchical states:
  - And-states (the contained sub-states execute in parallel)
  - Or-states (the contained sub-states execute sequentially)

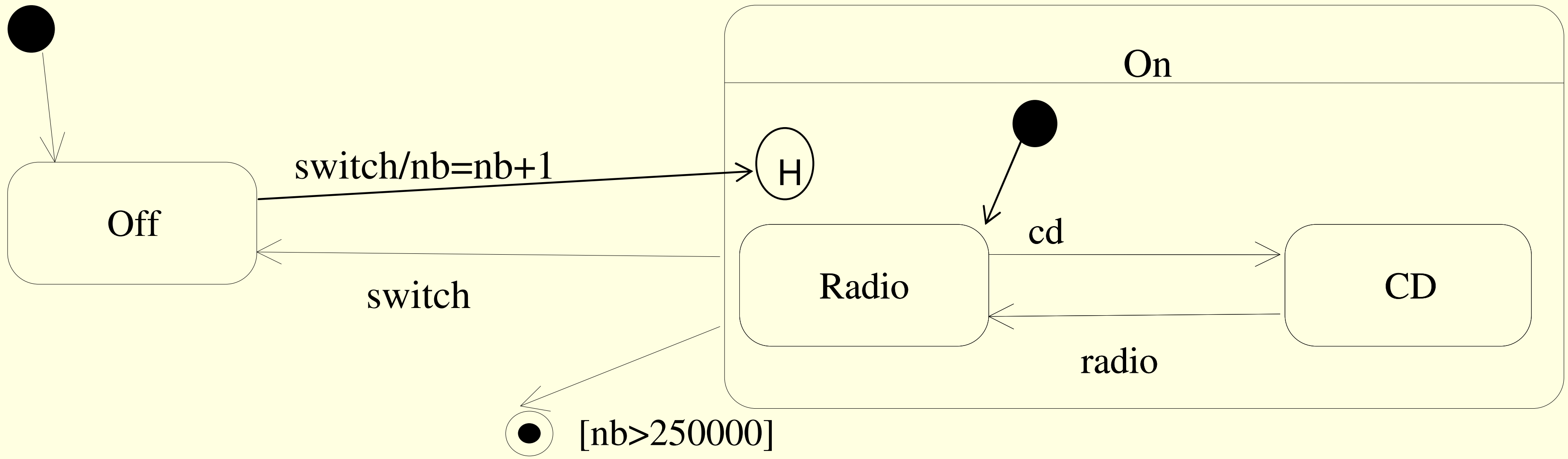
Hierarchical OR-state machine example



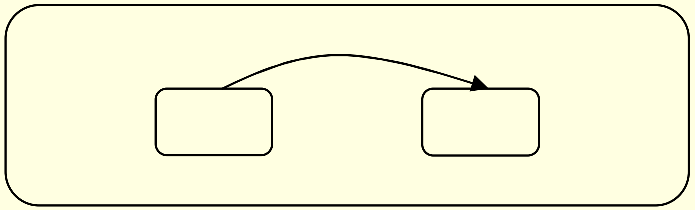
Hierarchical AND-state machine example



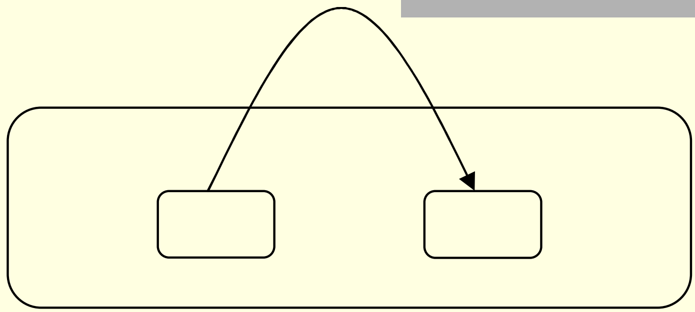
History sub-states

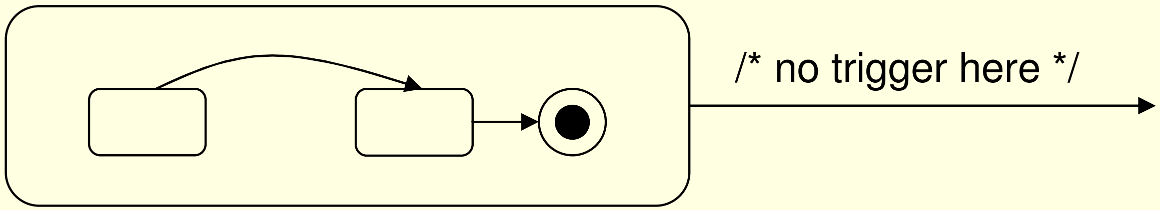


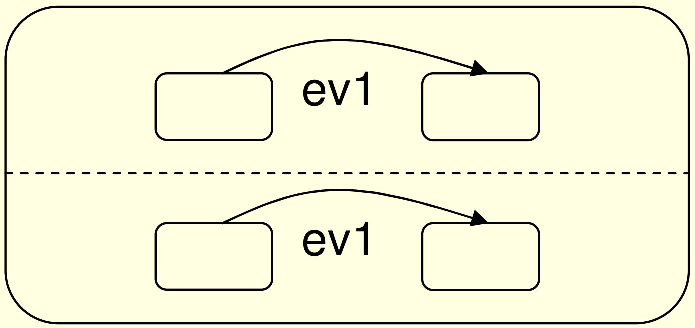
Semantic nuances in state machine diagrams

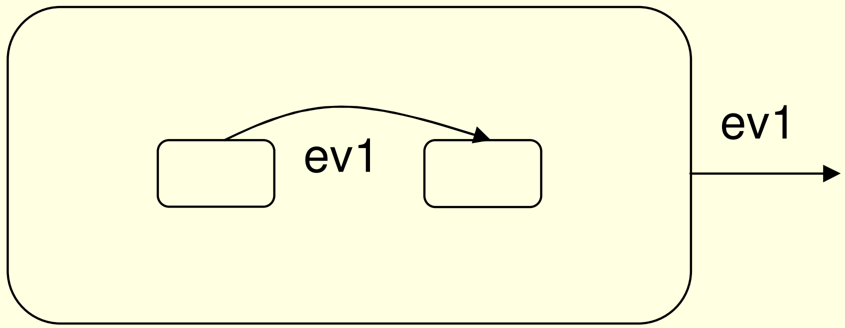


vs.









When to use state machines?

- For reactive systems

- Why use them?
  If properly used
  - easy to read
  - nice verification results
  - the tools can generate code more efficient than if hand-written

- Open questions:
  - state machine inheritance…
  - consensual semantics

Further reading:

Harel, David and Eran Gery, "Executable Object Modeling with Statecharts", *IEEE Computer*, July 1997, pp. 31-42.

Harel, David, "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming*, 8, 1987, pp. 231-274.

Overview

- What is UML?
- Structure description
- Behavior description
  - State machine diagrams
  - Protocol state machine
  - Activity diagrams
  - Sequence diagrams
  - Timing diagrams
- OCL
- UML and tools

Protocol state machines

- Particular state machines used to impose sequencing constraints
- Can be attached to interfaces, components, ports, classes

- Express
  - Usage protocols
  - Lifecycles for objects
  - Constrain the order of invocation for its operations

- Do not preclude any specific behavior description

- Protocol conformance must apply between the protocol state machine and the actual implementation

- A classifier may own several state machines (ex. due to inheritance)

Syntactic constraints on protocol state machines

- No entry, exit, do action on states
- No action on its transitions
- If a transition is triggered by an operation call, then that operation should apply to the context classifier

Protocol state machine interpretations

- Declarative
  - Specifies legal transitions for each operation
  - The actual legal transitions for operations are not specified
  - Defines the contract for the user of the context classifier
- Executable
  - Specifies all events that an object may receive and handle, plus the implied transitions
  - Legal transitions for operations are the triggered transitions

Protocol state machine example

- Notation: {protocol} mark should be placed close to the state machine name

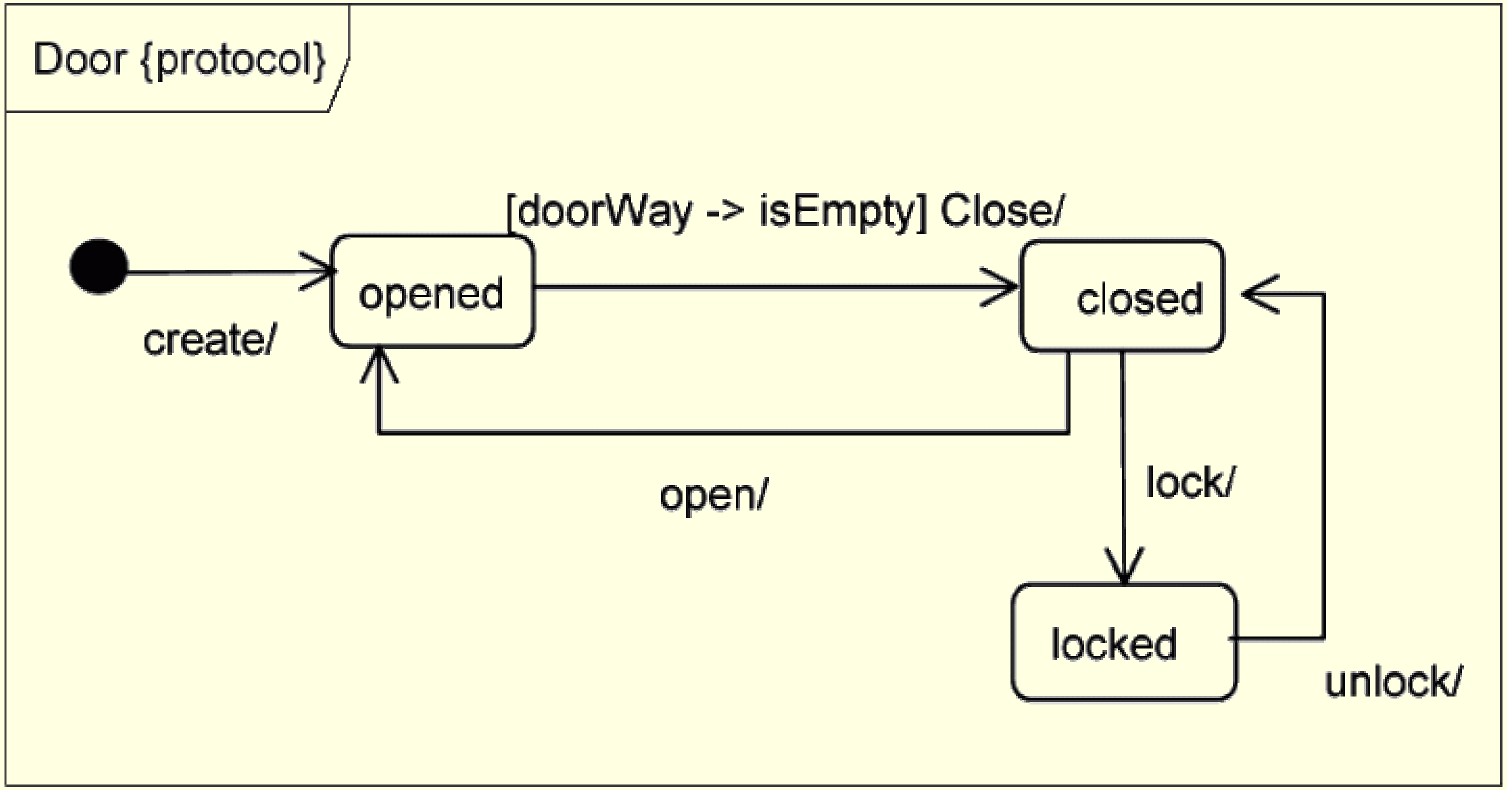


Figure 364 - Protocol state machine

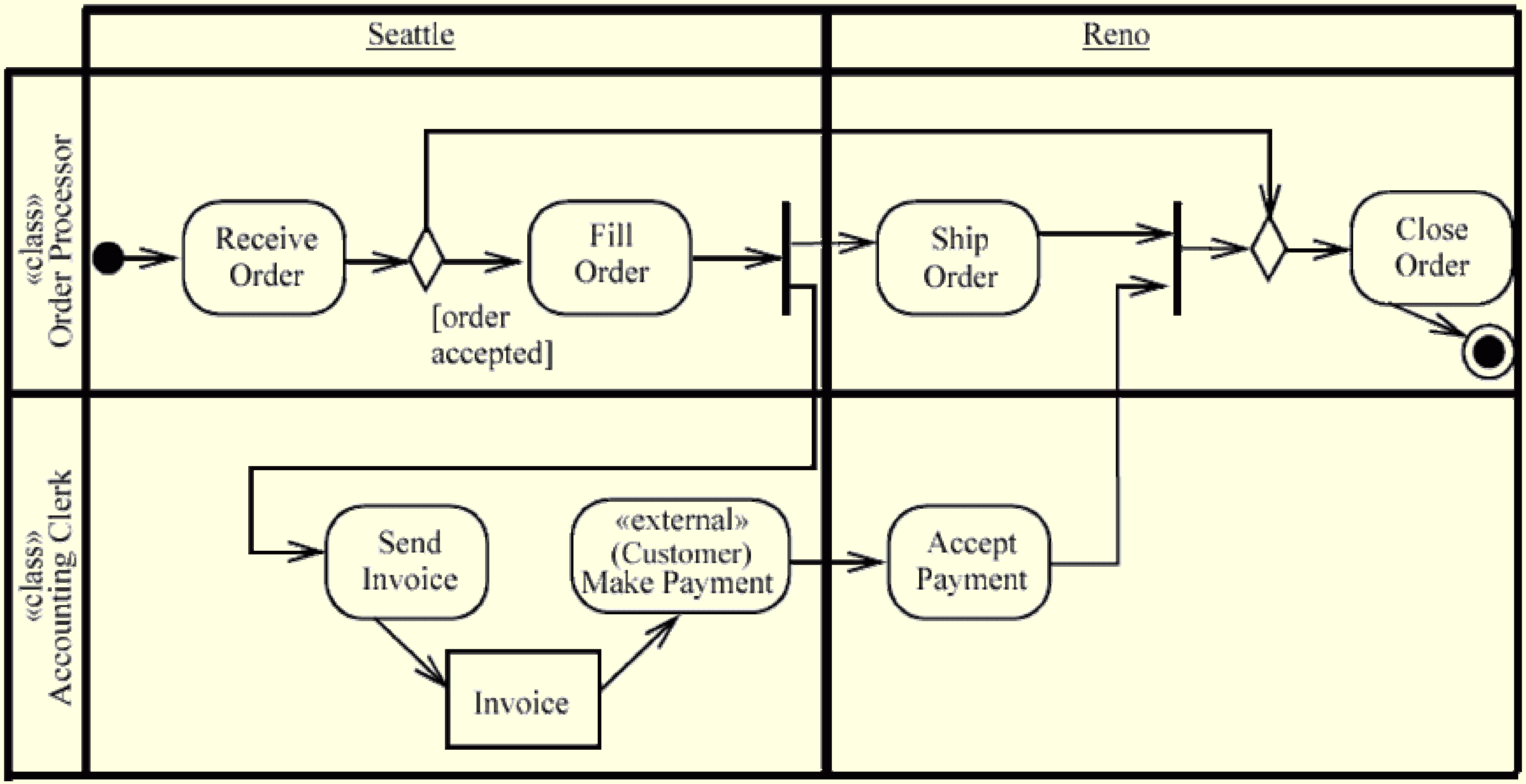
Overview

- What is UML?
- Structure description
- Behavior description
  - State machine diagrams
  - Protocol state machine
  - Activity diagrams
  - Sequence diagrams
  - Timing diagrams
- OCL
- UML and tools

Activity diagrams

- Related to state machine diagrams
  - State diagrams – focus on the execution of a single object
  - Activity diagram – focus on the behavior of a set of objects
- Purpose
  - Models high-level business processes, including data flow,
  - Models the logic of complex logic within a system
- Concurrency model based on Petri Nets

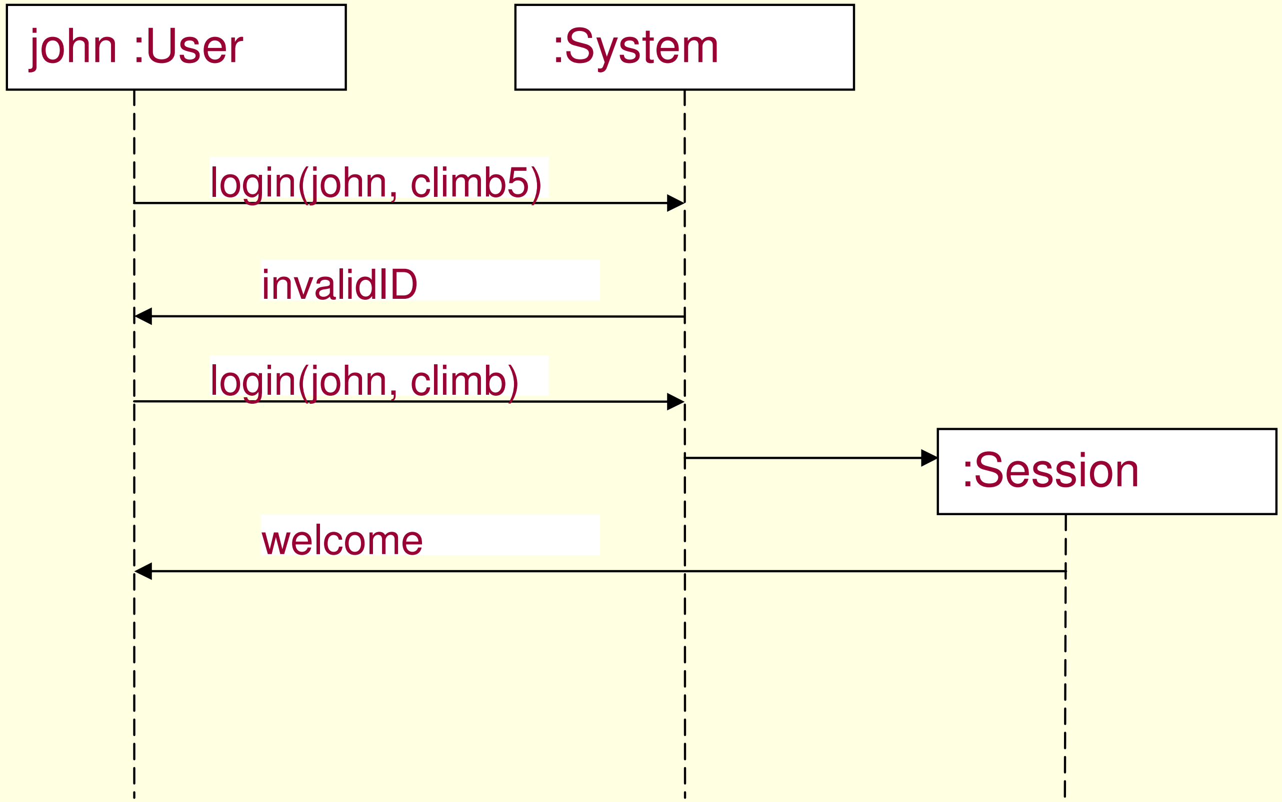
Activity diagram example



Sequence diagram

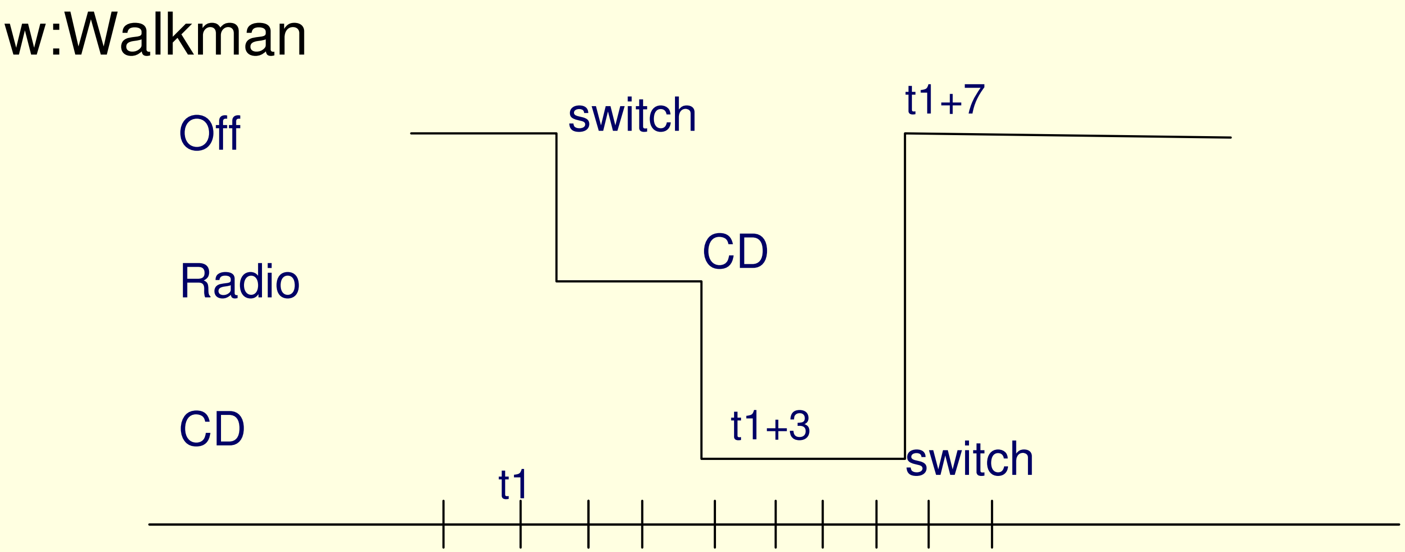
- Shows a concrete execution scenario, involving: objects, actors, generic system
- Highlights the lifelines of the participating instances
- Focuses on *interaction*, *exchanged messages* and their *ordering*
- Give *instances* of (cooperating) *state machine executions*
- Can address various levels of abstraction:
  - System level
  - Object sets level
  - Object level
  - Method level

Example


john :User
:System
login(john, climb5)
invalidID
login(john, climb)
:Session
welcome

Timing diagram

- used to explore the behaviors of one or more objects throughout a given time interval
- relevant for systems with time sensitive behavior



Although universal, UML can't contain everything…

- Extension mechanisms
  - Stereotype
    - mechanism allowing to specialize particular UML concepts
    - allows to use platform or domain specific terminology

    e.g. Class stereotyped *reactive* if it has a state machine

  - Tagged values – allows to attach information to UML model elements

- Profile - a stereotyped package containing model elements that have been customized (e.g. for a specific domain) using stereotypes, tagged definitions and constraints

  e.g. SPT, UML profile for EDOC, …

Overview

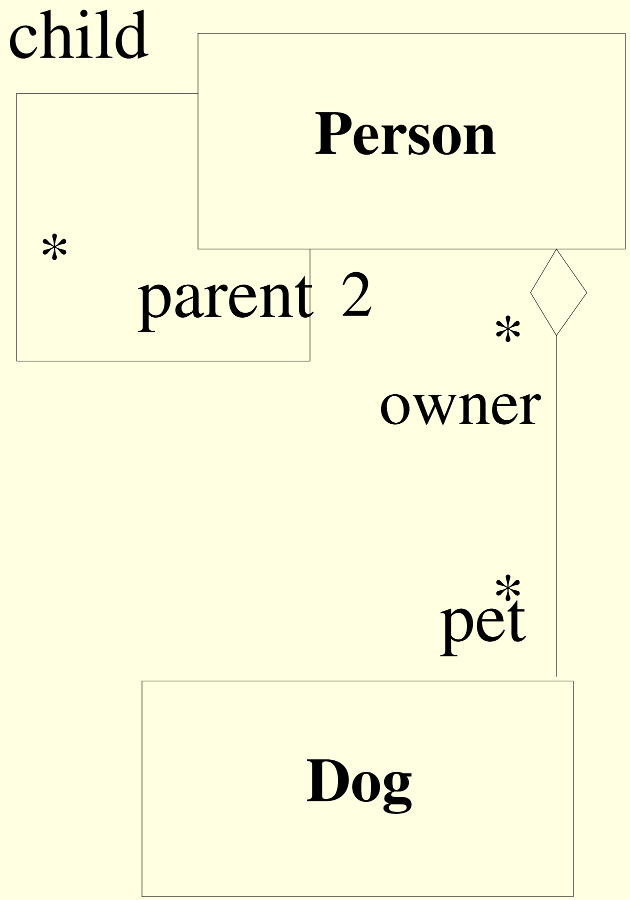
- What is UML?
- Structure description
- Behavior description
  - State machine diagrams
  - Protocol state machine
  - Activity diagrams
  - Sequence diagrams
  - Timing diagrams
- OCL
- UML and tools

OCL – Object Constraint Language

- Constraint language integrated in the UML standard
- Aims to fill the gap between mathematical rigor and business modeling
- Recommended for:
  - Constraints: pre and post conditions, invariants
  - Boolean expressions: guards, query body specification
  - Defining initial and derived values of features
- UML meta-model WFRs written in OCL

Example 1 – (all kinds of) invariants

No grandchild may not have more than 2 pet dogs:



contex Person

inv: self.child.child.pet -> size()<2

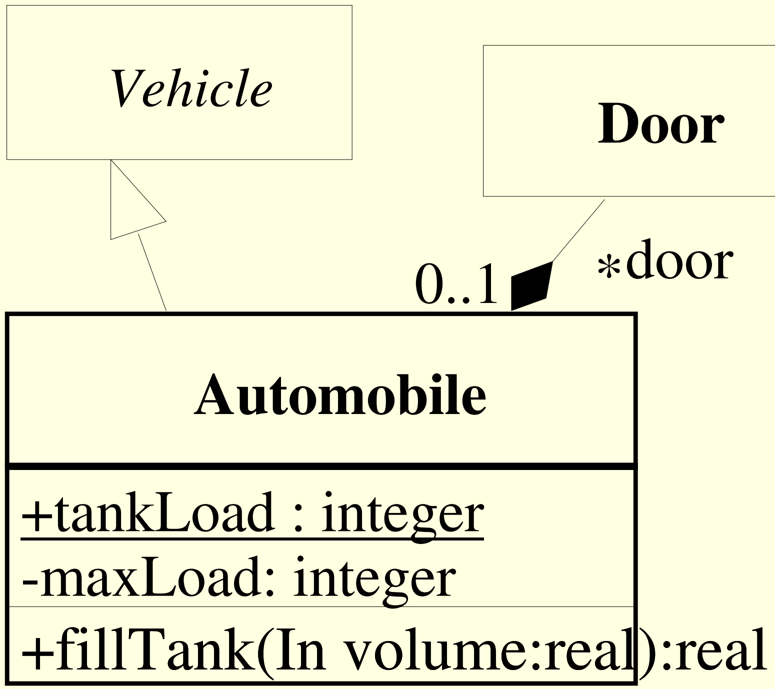
Example 2 – pre and post conditions

contex Automobile::fillTank (in volume:real):real

pre: volume>0

pre: tankLoad + volume < maxLoad

post: tankLoad = tankLoad@pre + volume



Overview

- What is UML?
- Structure description
- Behavior description
- OCL
- UML and tools

Tool support for UML

- UML can only live if tool builders support it
  Just think of a programming language with no compiler…
- Tool builders are de facto deciders of live and dead parts of the languages
- There is no UML tool that offers all the functionalities one can think of
- This part is not a presentation of tools, rather a list a functionalities offered by various tools

Functionalities

- Editing support
- Documentation generation
- Syntax check
- Static semantic check

- Code generation
- Symbolic execution / simulation
- Formal verification

- Support for tests on model
- Test case generation

- Reverse engineering
- Model transformation and translations to other formalisms
- …

Model interchange

- The need
  - A single tool does not offer all the functionalities
  - Avoid user kidnapping
- The solution
  - XMI: standardized model interchange format
  - Offers an XML DTD schema of the metamodel, to be used by tools
- The reality
  - Commercial tools offer limited support (why?)
  - The complexity of the UML metamodel often leaves place to interpretations => incompatibilities
  - Until UML 2.0 no diagram interchange

Conclusions – UML summary

- UML – modeling language to be used throughout the entire software lifecycle
- Capture requirements
  - Use cases
  - Sequence diagrams
- Structure aspects
  - OO inspired definition
  - Component support
- Behavior aspects
  - State machines – for reactive behavior
  - Actions – in general
- Deployment aspects
  - Component/deployment diagrams

UML summary (2/2)

- To be as flexible as possible
  - UML offers extension mechanisms, profiles
  - Using profile UML can be transformed in a DSL
- Tool support
  - Lots of commercial/non-commercial tools exist
  - Various functionalities offered
  - Tool interchange exists, but lots are still to be done

Impact on research activity

Researchers attitude evolved:

- Hostility: received with skepticism, and (violent) critics
- Resign: very used in research papers, projects, books
- Pragmatism: taken as it is, used as a bridge with the industrial world
- Often the main focus of conferences, workshops, basic research, more as a means than as a goal

The bad news is that …

- The various notations within UML are not perfectly coordinated
- Often, different tools interpret the UML standard differently
- The unique modeling language is in fact a set of dialects

The good news is that …

- We have a language allowing to design and model various aspects of systems
- This language is standardized and supported by various tools

- The tool support and interoperability improves in time, as UML, OCL, and XMI are still relatively young standards