

Analysis, Design, and Software Architecture (BDSA)
Paolo Tell

Introduction to UML

Recap

What did we see so far ... I

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software engineering is a collection of techniques, methodologies, and tools that help with the production of a high quality software system developed with a given budget before a given deadline while change occurs.
- Requirements engineering is the process of developing a software specification.
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

What did we see so far ... II

- Functional requirements statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations.
- Non-functional requirements constrain the system being developed and the development process being used. Often relate to the emergent properties of the system and therefore apply to the system as a whole—they concern the architecture.
- Domain requirements are constraints on the system from the domain of operation—the application domain.
- The software requirements document is an agreed statement of the system requirements that should be organized so that both system customers and software developers can use it.
- The requirements engineering process an iterative process that includes requirements elicitation, specification, and validation.
- Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities—requirements discovery, requirements classification and organization, requirements negotiation, and requirements documentation.

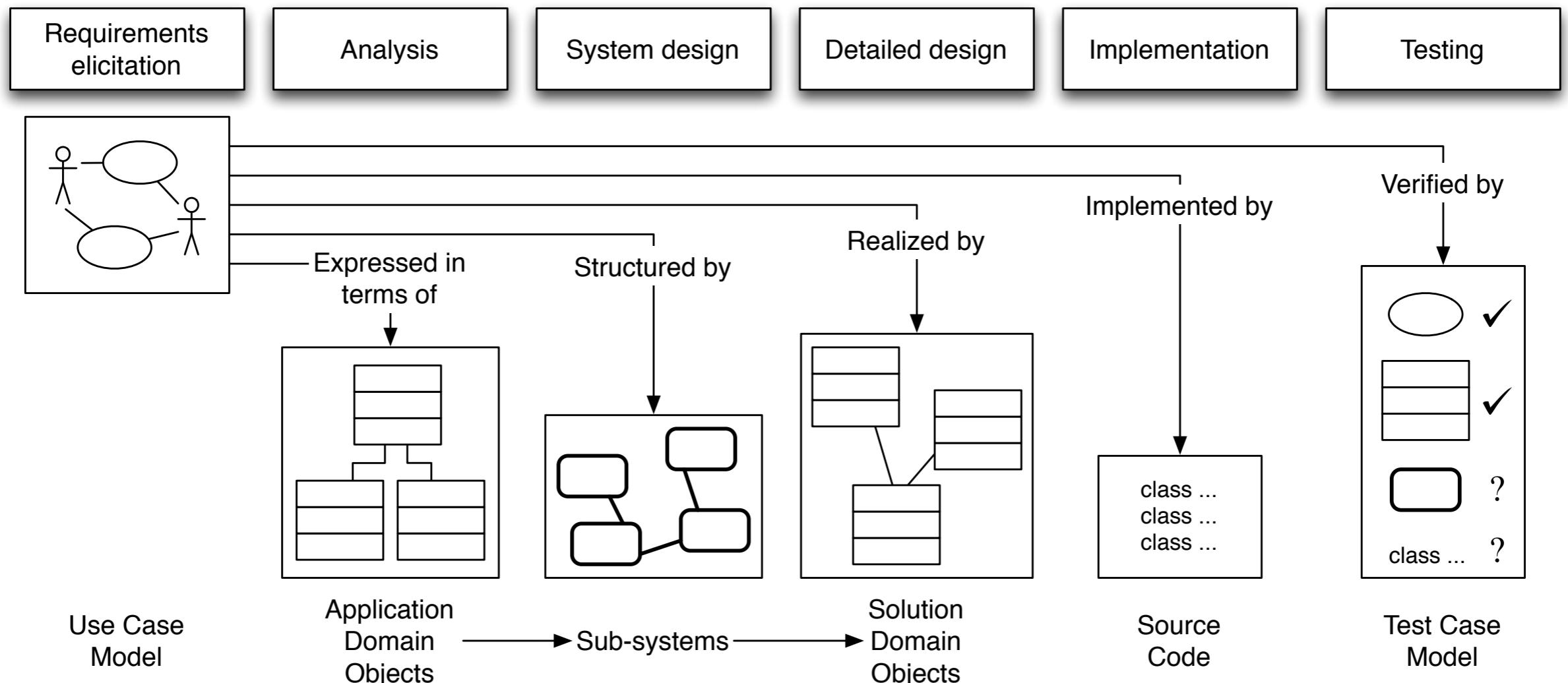
What did we see so far ... III

- Requirement elicitation concepts
 - Completeness, Consistency, Clarity, and Correctness
 - Realism, Verifiability, and Traceability
 - Greenfield Engineering, Reengineering, and Interface Engineering
- Requirements elicitation activities
 - Identify Actors, Scenarios, and Use Cases
 - Identify Relationships between Actors and Use Cases
 - Identify Initial Analysis Objects
 - Identify Non-functional Requirements
- Documenting requirements elicitation
 - Requirements analysis document (RAD)

Requirements Analysis Document

1. Introduction
 - 1.1 Purpose of the system
 - 1.2 Scope of the system
 - 1.3 Objectives and success criteria of the project
 - 1.4 Definitions, acronyms, and abbreviations
 - 1.5 References
 - 1.6 Overview
2. Current system
3. Proposed system
 - 3.1 Overview
 - 3.2 Functional requirements
 - 3.3 Nonfunctional requirements
 - 3.3.1 Usability
 - 3.3.2 Reliability
 - 3.3.3 Performance
 - 3.3.4 Supportability
 - 3.3.5 Implementation
 - 3.3.6 Interface
 - 3.3.7 Packaging
 - 3.3.8 Legal
 - 3.4 System models
 - 3.4.1 Scenarios
 - 3.4.2 Use case model
 - 3.4.3 Object model
 - 3.4.4 Dynamic model
 - 3.4.5 User interface—navigational paths and screen mock-ups
 4. Glossary

What did we see so far ... IV



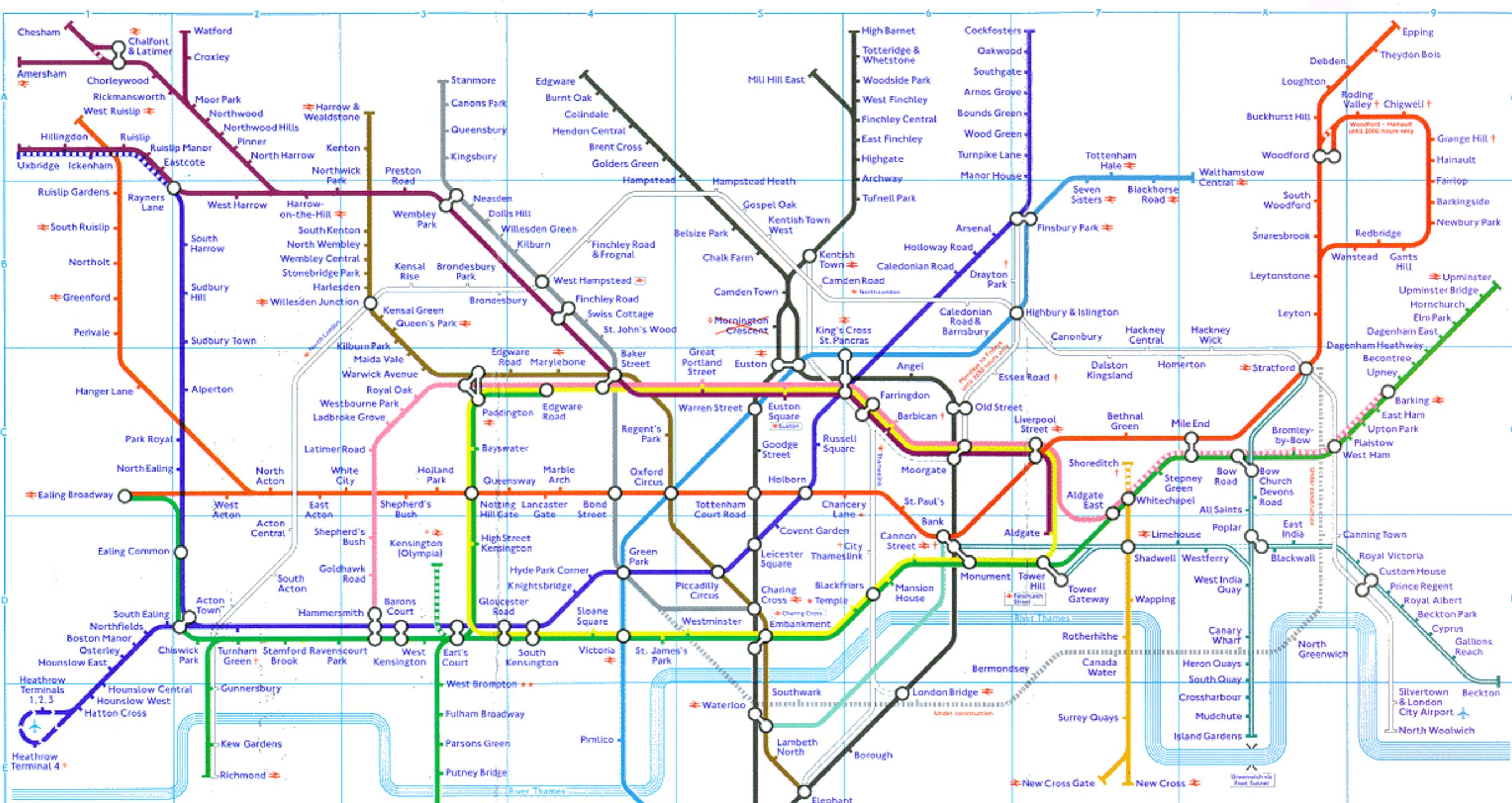
- Software lifecycle

Outline

- Literature
 - [OOSE] ch. 2
- Topics covered:
 - Modeling
 - UML and its history
 - UML diagrams
 - Brief description
 - Example(s)
 - Comment (or another example)
 - Template from [\[uml-diagrams.org\]](http://uml-diagrams.org)
 - A deeper dive into class diagrams

Modeling





Key to Lines

- | | | | |
|--|--------------------------------|--|---------------------------|
| | Bakerloo | | Metropolitan |
| | Central | | peak hours only |
| | peak hours only | | Northern |
| | Circle | | Piccadilly |
| | District | | peak hours only |
| | restricted service | | Victoria |
| | East London | | Waterloo & City † |
| | peak hours and Sunday mornings | | Docklands Light Railway † |
| | Hammersmith & City | | under construction |
| | peak hours only | | British Rail |
| | Jubilee | | restricted service |
| | under construction | | |



Key to symbols

- | | | |
|--|--|--|
| Interchange stations | These stations are open at the following times | |
| Connections with British Rail | Barbican | All day Mondays to Fridays
0715 to 2345 Saturdays, 0800 to 2345 Sundays. |
| Connections with British Rail within walking distance | Cannon Street | Until 2045 Mondays to Fridays
Closed Saturdays and Sundays |
| Airport Interchange | Chigwell | Until 2000 daily |
| Closed Sundays | City Thamestlink | 0600 to 2045 Mondays to Fridays
Closed Saturdays and Sundays |
| Closed Saturdays and Sundays | Docklands Light Railway | Poplar – Beckton until 2150 Mondays to Fridays only
All stations between Tower Gateway, Bank, Stratford and Island Gardens. Buses replace trains all day Saturdays and Sundays. |
| Mornington Crescent closed for rebuilding | Drayton Park | Until 2030 Mondays to Fridays.
Closed Saturdays and Sundays |
| Certain stations are closed on public holidays. | | Essex Road Until 2030 Mondays to Fridays
Closed Saturdays and Sundays |
| | | Grange Hill Until 2000 daily |
| | | Heathrow Terminal 4 Until 1545 Mondays to Saturdays and 2315 Sundays. |
| | | Kensington (Olympia) 0700 to 2045 Mondays to Fridays. Saturdays and Sundays during exhibitions |
| | | Roding Valley Until 2000 daily |
| | | Shoreditch Hoxton to Friday peak hours
Closed Saturdays 0800 to 1415 Sundays |
| | | Turnham Green Served by Piccadilly Line trains early morning and late evening Mondays to Saturdays and all day Sundays |
| | | Waterloo & City Line 0630 to 2130 Mondays to Fridays 0800 to 1745 Saturdays. Closed Sundays. |

Modeling concepts

- **System**
 - an organized set of communicating parts.
 - a system is composed of a set of sub-systems.
 - models are used to handle complex systems.
 - often several models are used to describe a system.
- **Model**
 - an abstraction describing a system or sub-system (divide-et-impera).
- **Modeling**
 - Modeling means constructing an abstraction of a system that focuses on interesting aspects and ignores irrelevant details.
 - A rule of thumb is that each entity should contain at most 7 ± 2 parts [Miller, 1956].
- **View**
 - depicts selected aspects of a model.
- **Notation**
 - is a set of graphical or textual rules for depicting models and views.

Systems, Models, and Views

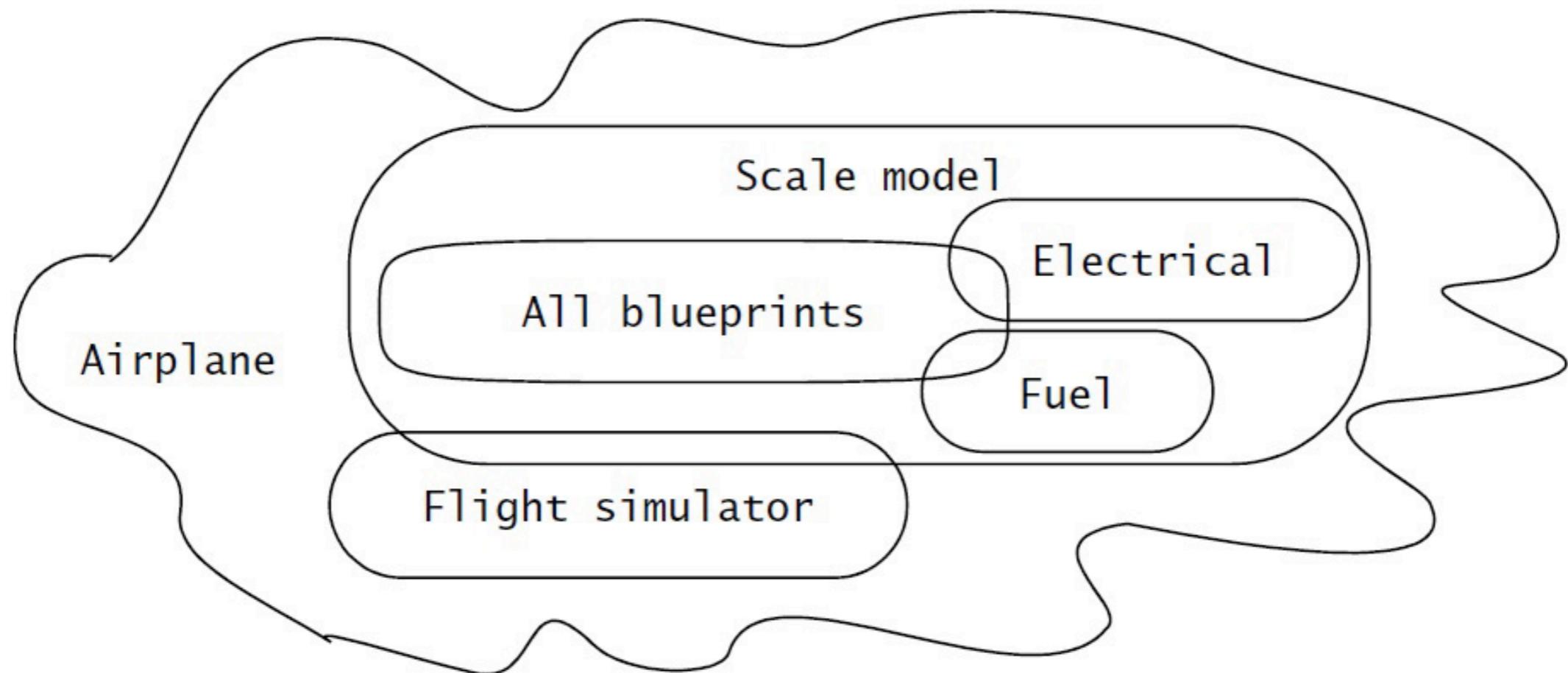
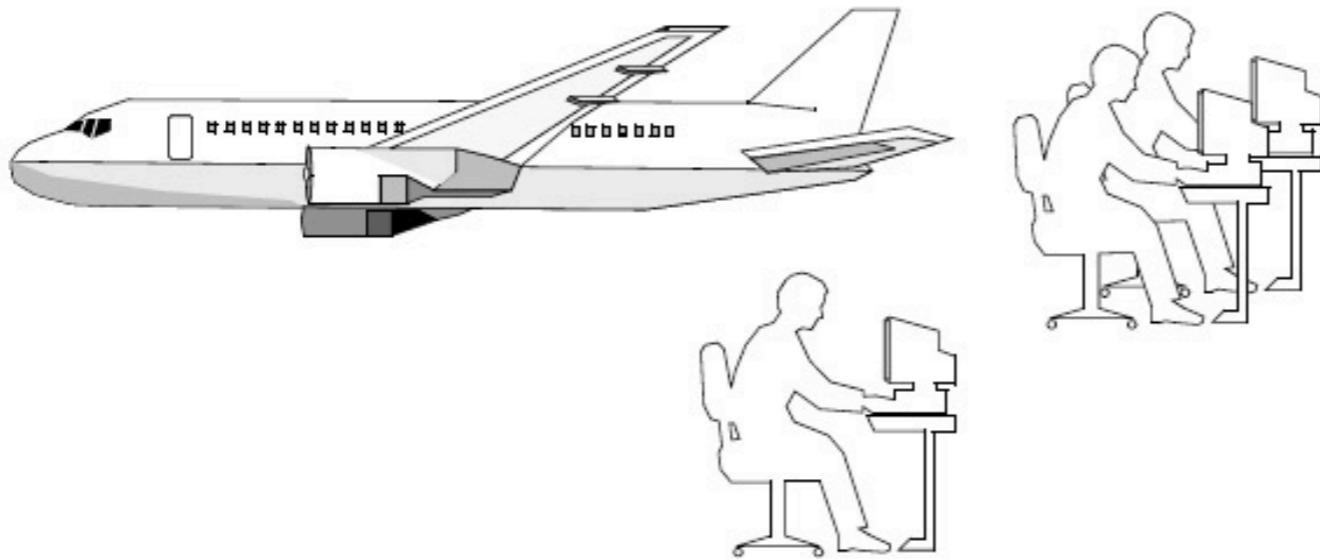


Figure 2-6 A model is an abstraction describing a subset of a system. A view depicts selected aspects of a model. Views and models of a single system may overlap each other.

Object-oriented modeling

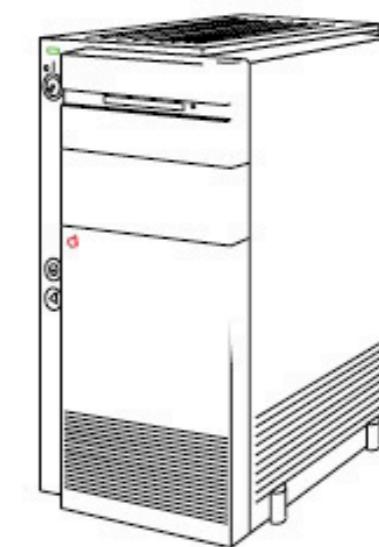
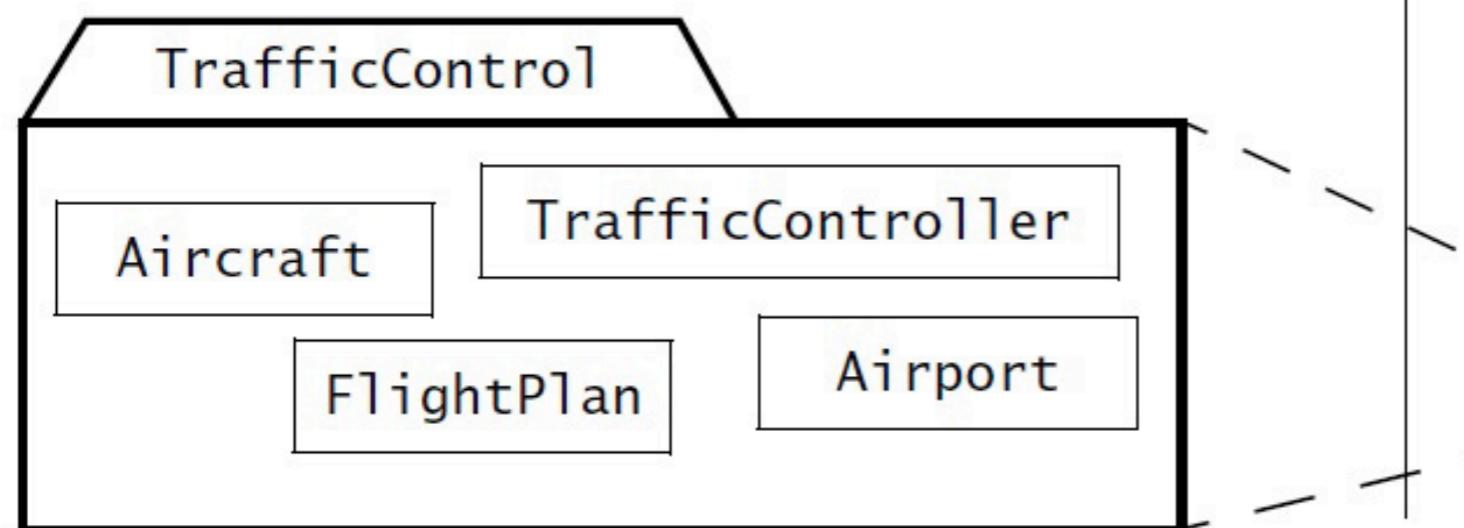
- Application Domain (Analysis)
 - The environment in which the system is operating.
 - The user's problem.
 - In OO we model this domain through object-oriented analysis (OOA).
- Solution Domain (Design, Implementation)
 - The technologies used to build the system.
 - The software engineer's problem.
 - In OO we model this domain through object-oriented design (OOD).
- Both domains contain abstractions that we can use for the construction of the system model.

Object-oriented modeling



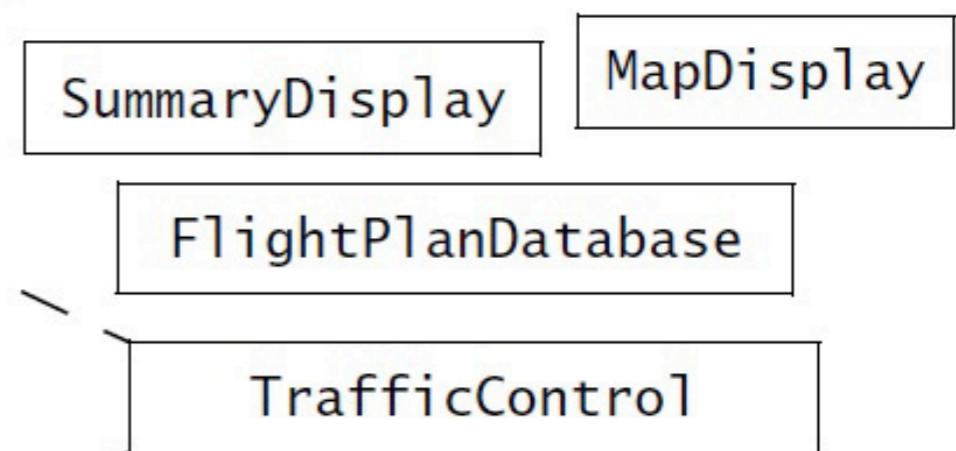
Application Domain

Application Domain Model



Solution Domain

System Model





The Unified Modeling Language (UML) and its history

Unified Modeling Language (UML)

- Object Management Group (OMG) definition
 - “The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems” [OMG].
- Adopted in 1997 as a standard by OMG.
- Published in 2005 by the International Organization for Standardization (ISO) as an approved ISO standard.

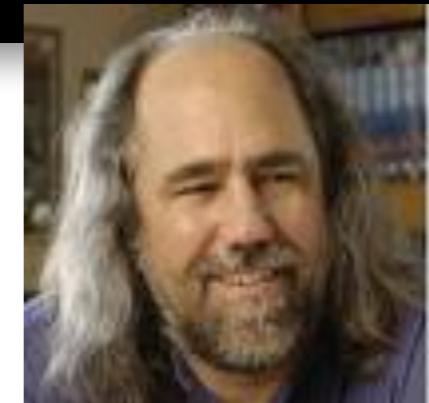
The 3 amigos



25 years at General Electric Research, where he developed OMT, joined (IBM) Rational in 1994, CASE tool OMTool

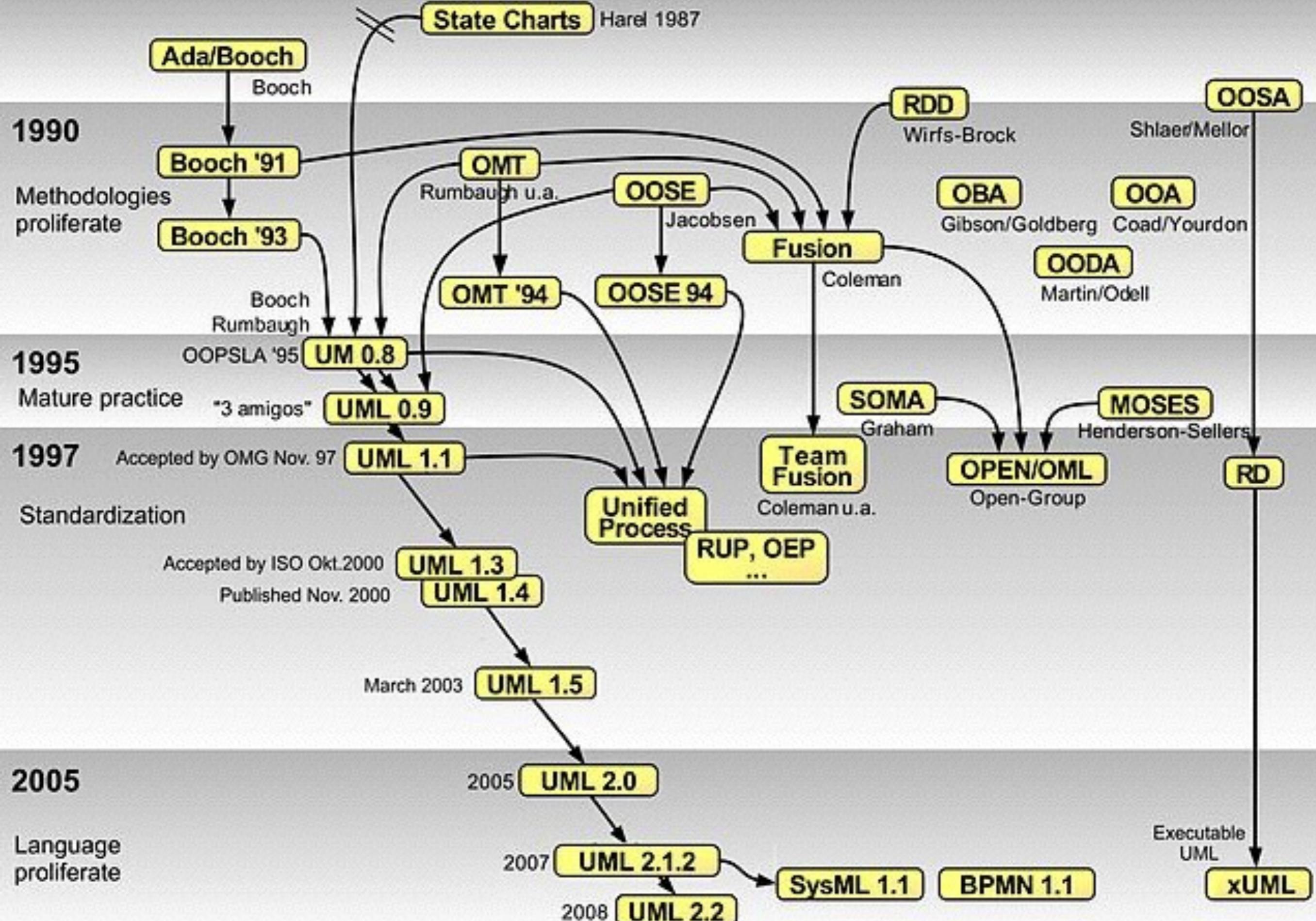


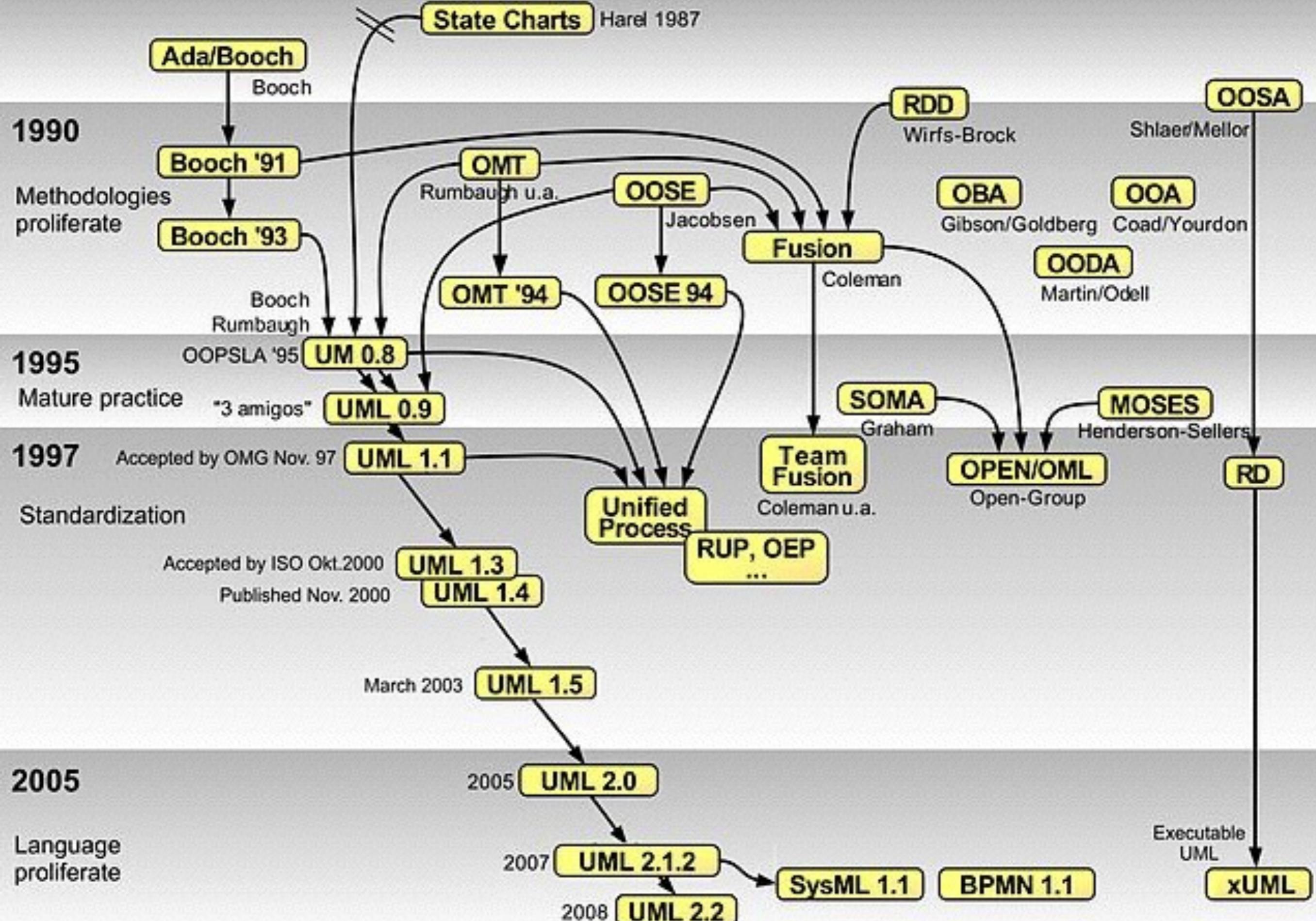
At Ericsson until 1994, developed use cases and the CASE tool Objectory, at IBM Rational since 1995



Developed the Booch method ("clouds"), ACM Fellow 1995, and IBM Fellow 2003

- Convergence of different notations used in object-oriented methods, mainly
 - OMT (James Rumbaugh and colleagues), OOSE (Ivar Jacobson), Booch (Grady Booch)
- They also developed the Rational Unified Process, which became the Unified Process in 1999
 - Iterative process (analysis, design, implementation, and testing at each iteration)





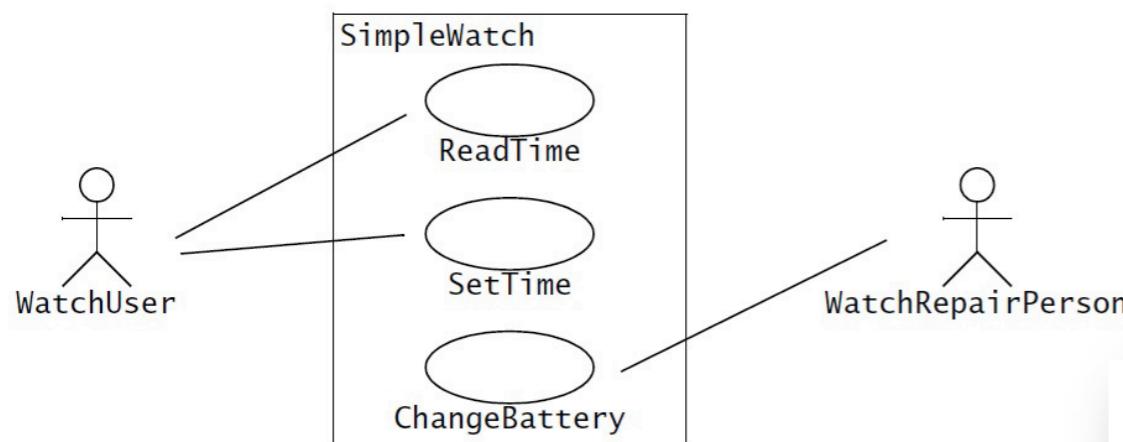
Why UML?

- Provide users a expressive visual modeling language.
- Facilitate communication among developers. UML became the lingua franca in modeling.
- Incorporate the object-oriented community's consensus on core modeling concepts.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Support the creation of object-oriented CASE tools.
- Support higher-level development concepts such as collaborations, frameworks, patterns, and components.
- Win the ‘war of notations’.

UML

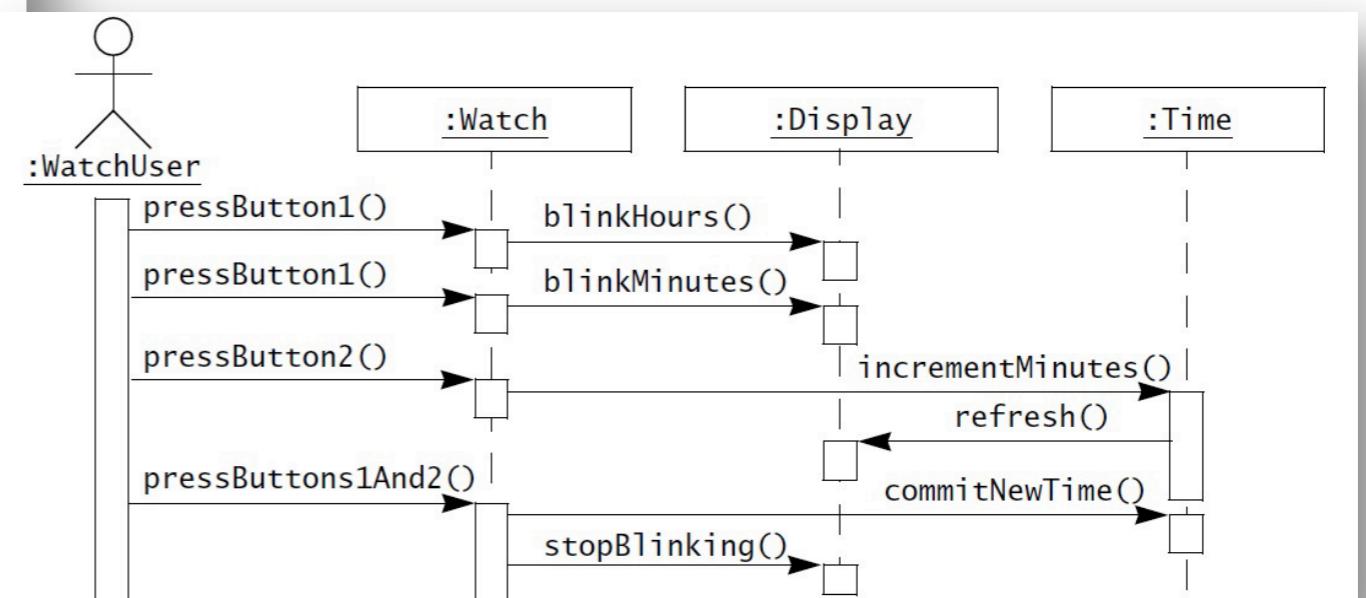
- 3 views on UML
 - UML as sketch.
 - UML as blueprint.
 - Before ~ specification.
 - After ~ documentation.
 - UML as a programming language.
- In the OOSE book
 - “Just” a diagramming notation standard.
 - Trivial and relatively unimportant.
 - Not a method, process, or design guide.
- In this course
 - You need to be able to draw syntactically correct UML diagrams – important part of the exam.

Examples

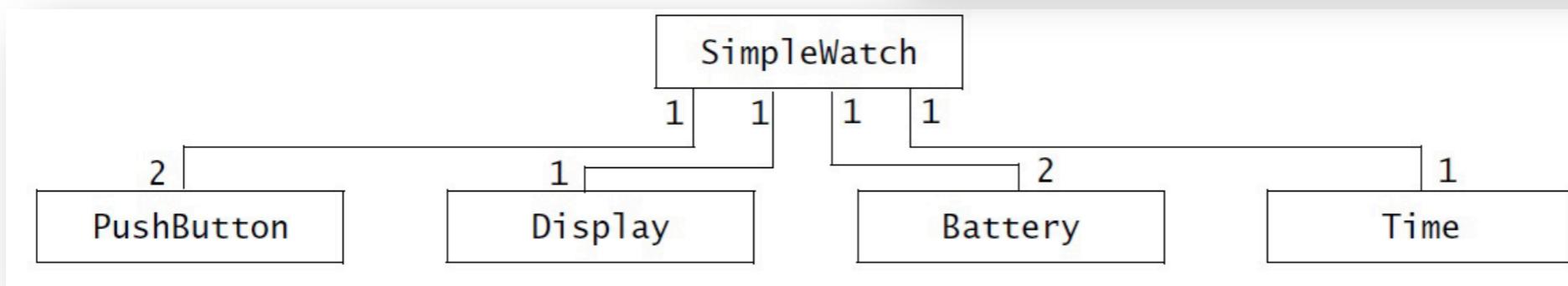


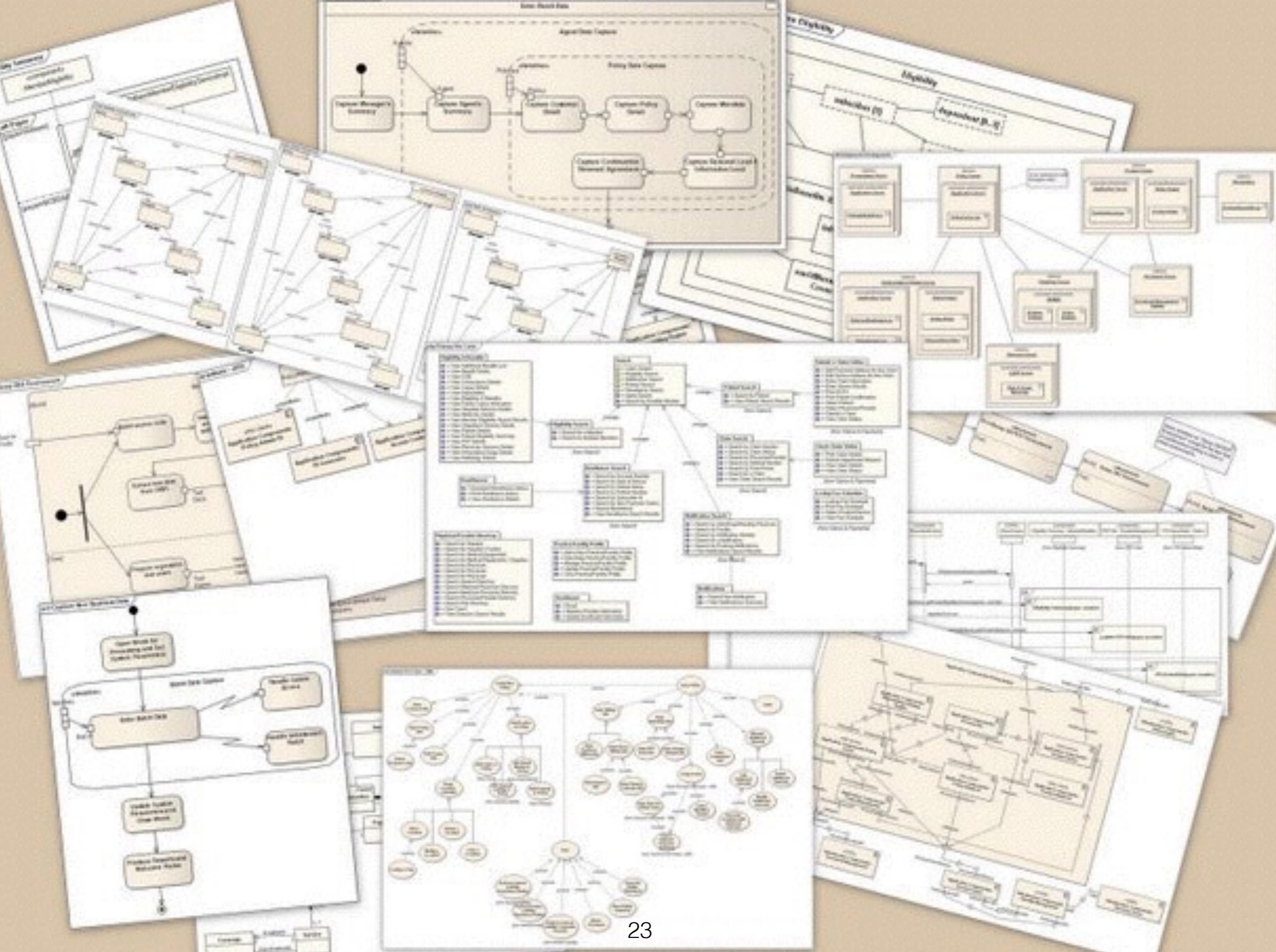
Use Case Diagram

Interaction Diagram

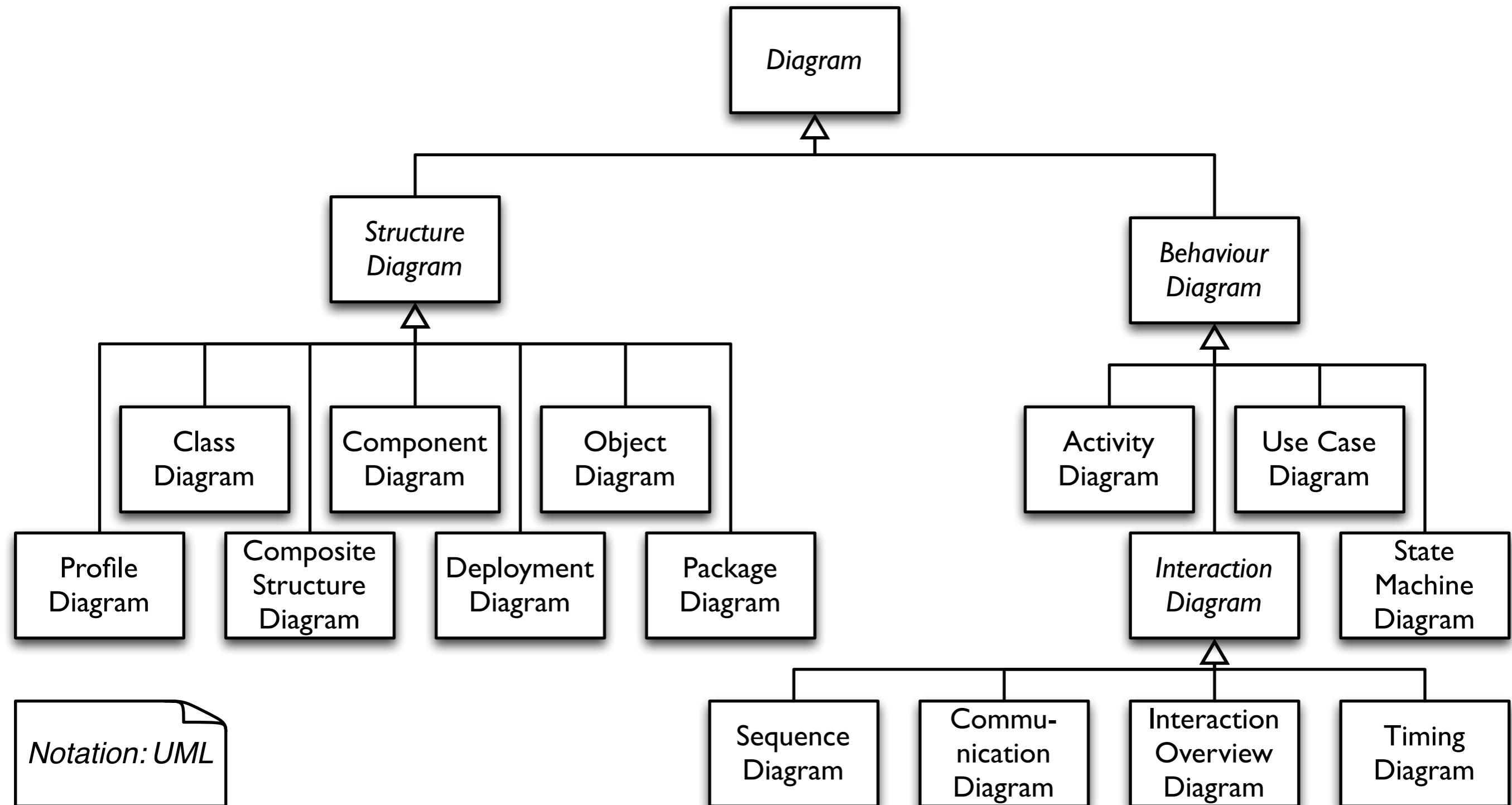


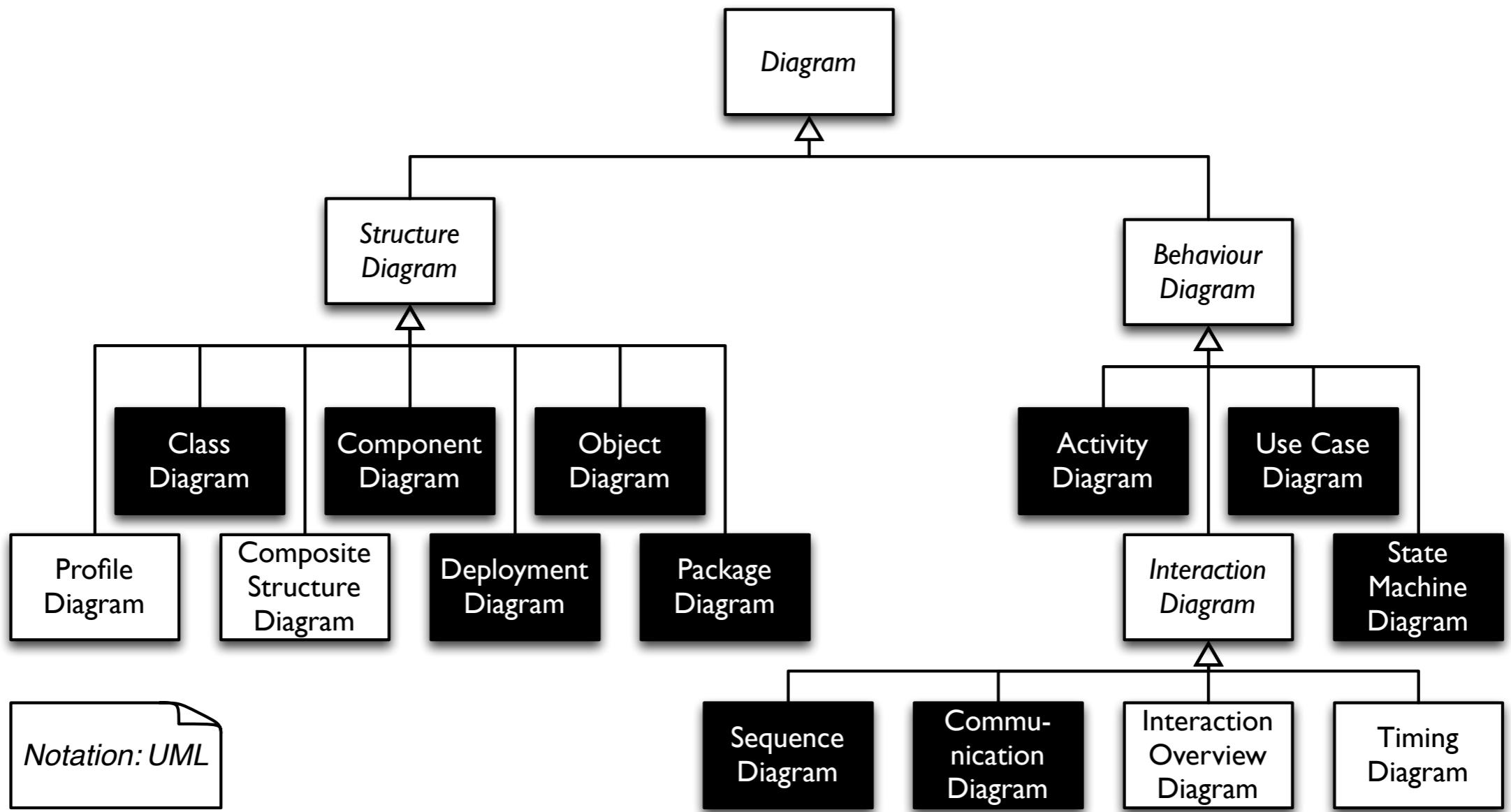
Object Diagram





Hierarchy of diagrams in UML 2.x

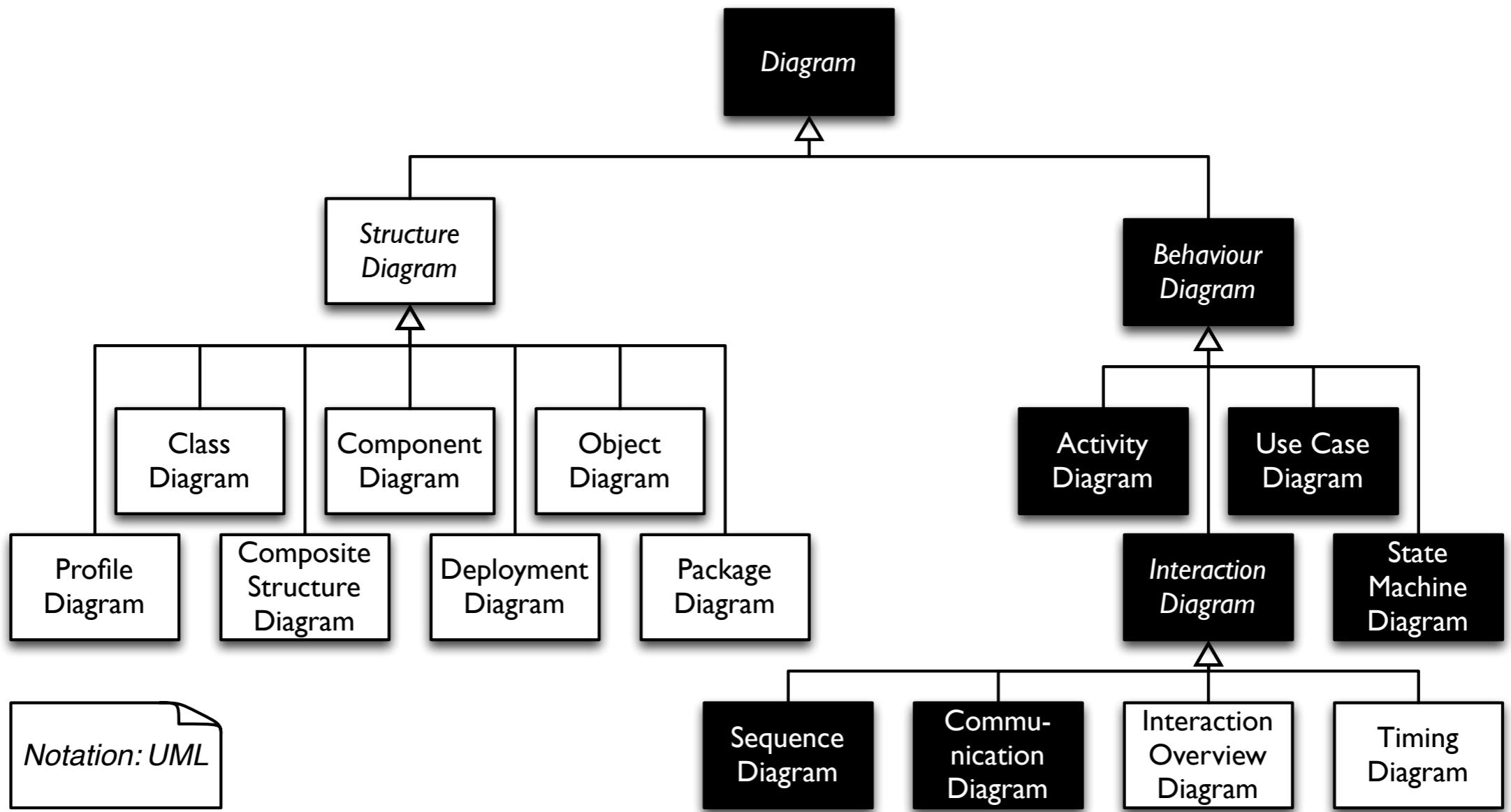




Diving into UML

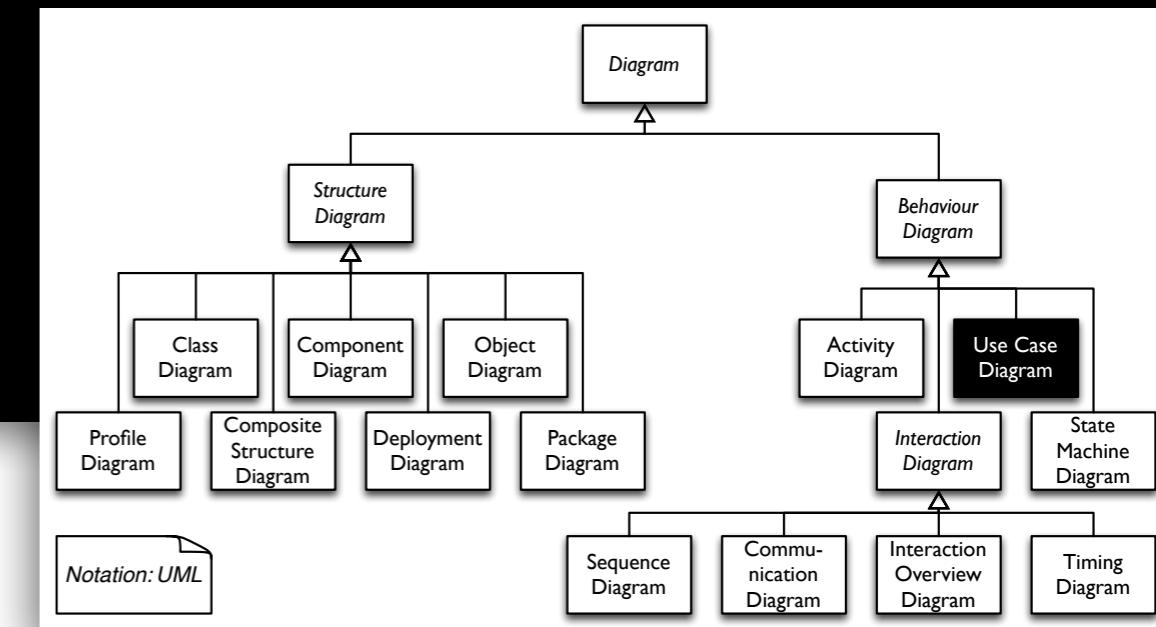
Core conventions

- All UML Diagrams denote graphs of nodes and edges
 - Nodes are entities and drawn as rectangles or ovals
 - Rectangles denote classes or instances
 - Ovals denote functions
- Names of Classes are not underlined
 - SimpleWatch
 - Firefighter
- Names of instances (object:class) are underlined
 - myWatch:SimpleWatch
 - Joe:Firefighter
- An edge between two nodes denotes a relationship between the corresponding entities

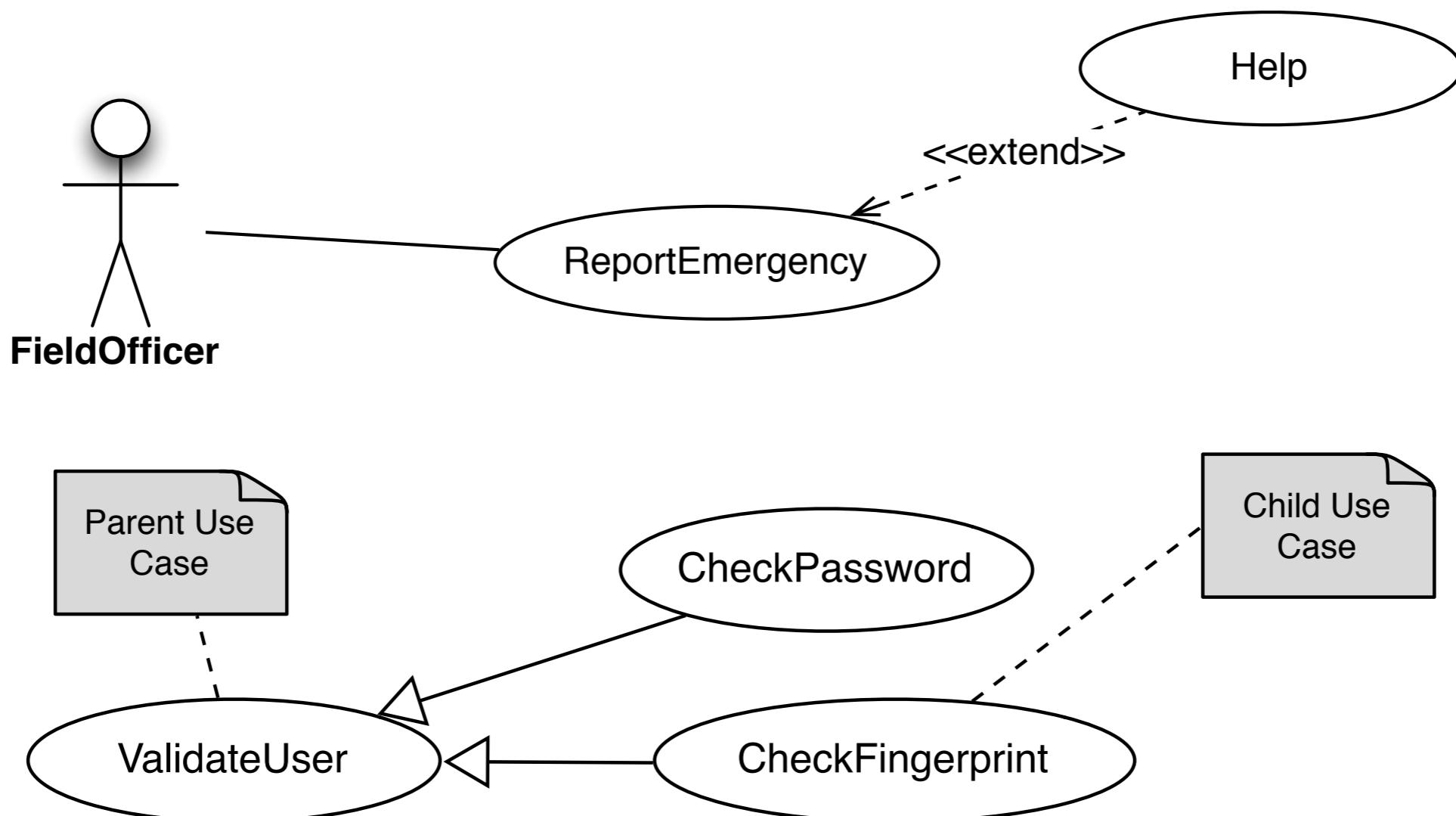
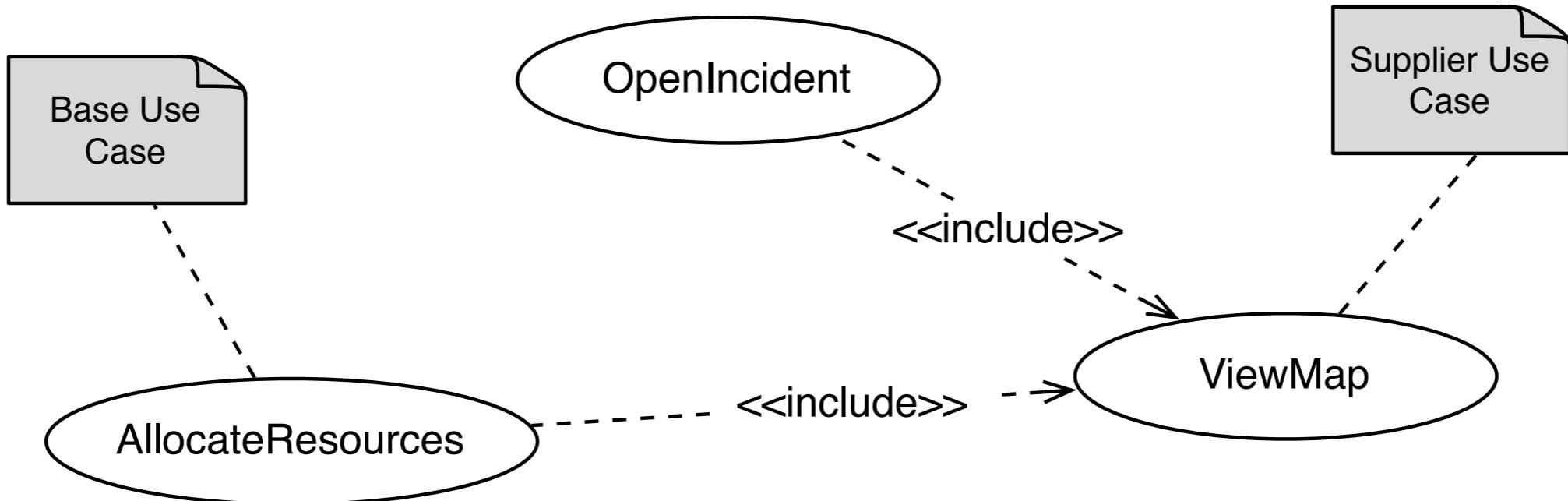


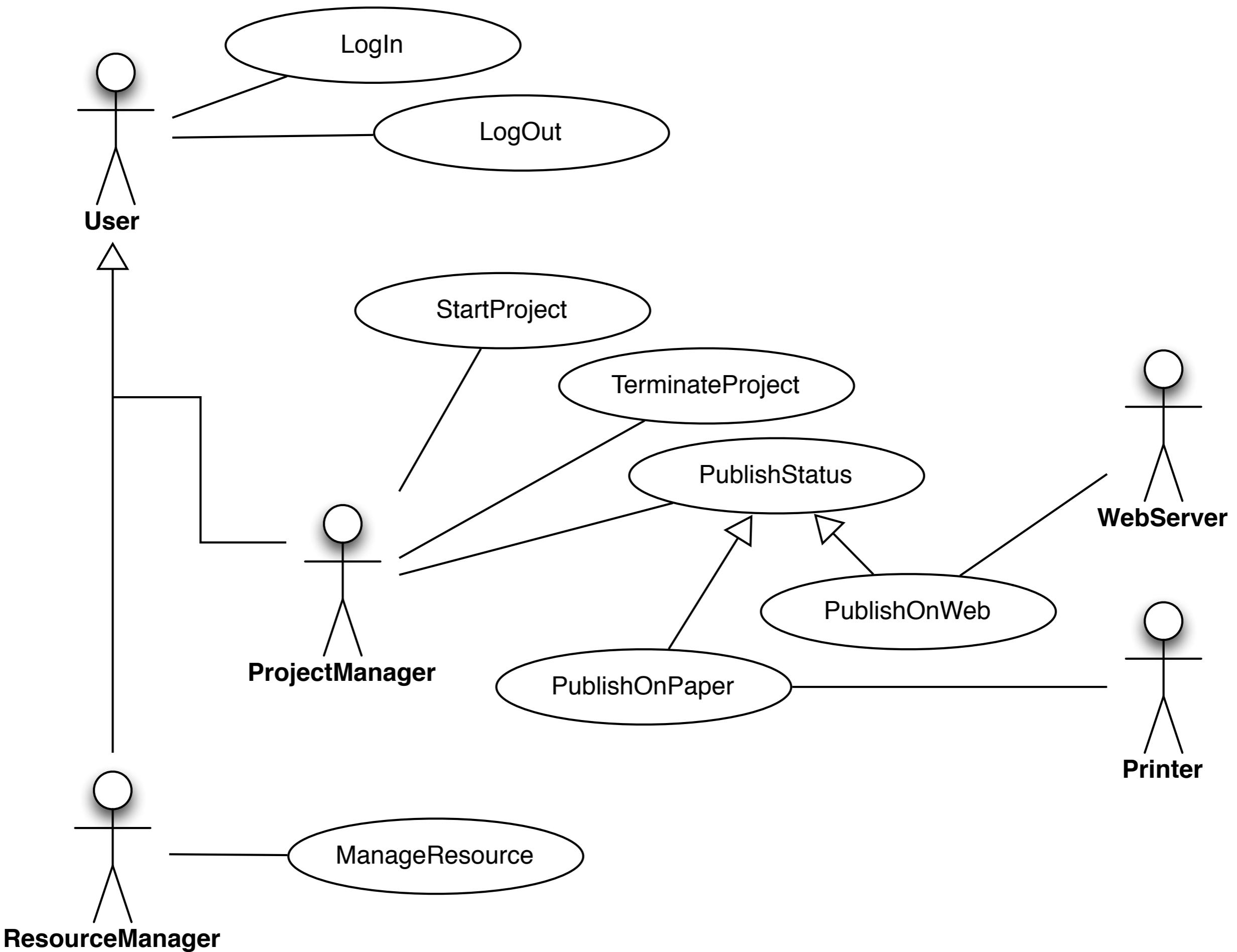
Diving into UML (1st part)

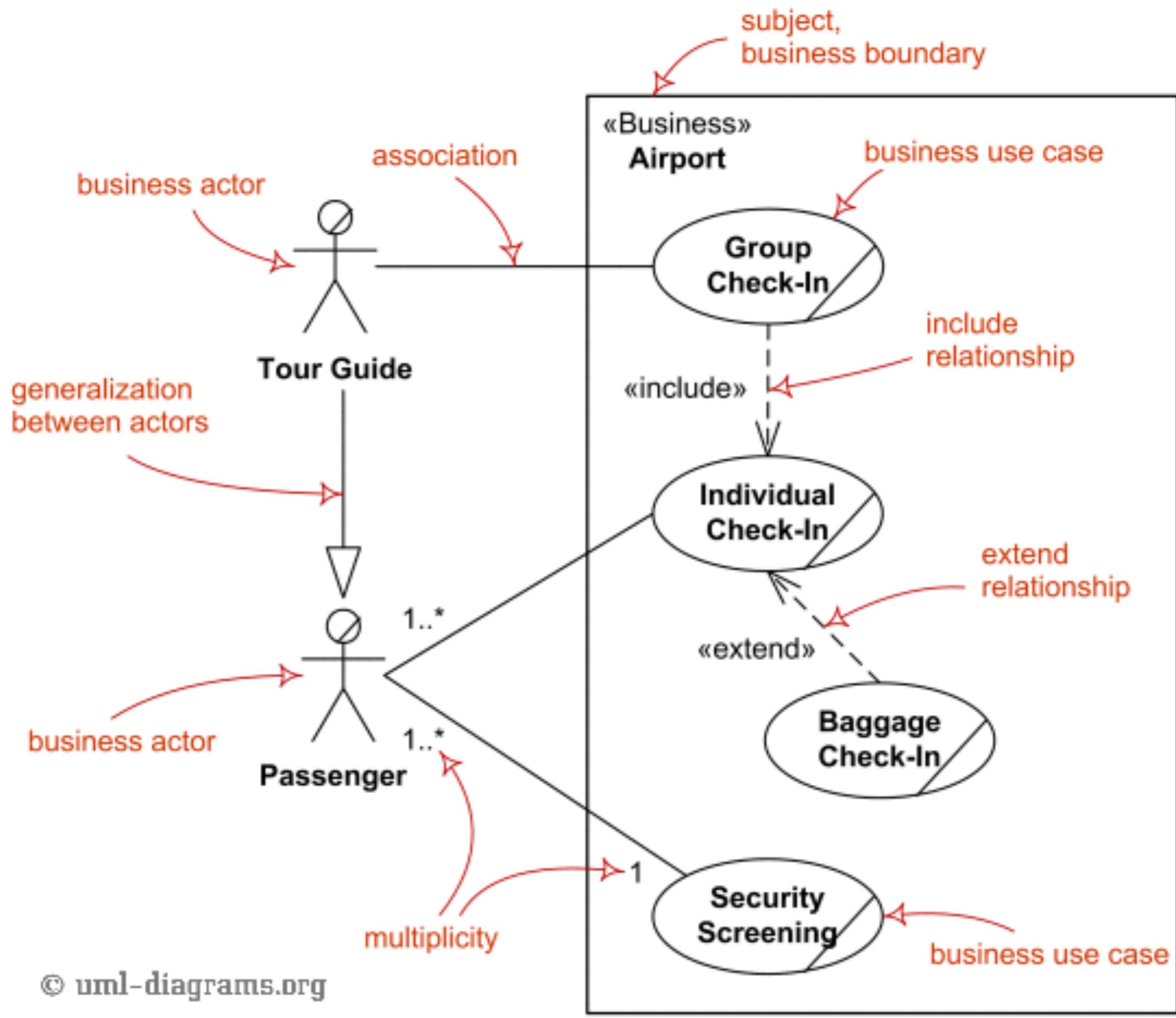
Use case diagram



- Behaviour diagrams.
- They capture requirements.
- A use case captures a specific functionality offered by the system.
- They abstract scenarios.
- They comprise of two parts:
 - the use case diagram (the “sticky man” diagram).
 - actors (roles, not only humans), use cases, and associations (includes, extends, generalization).
 - the use case text.







<i>Use case name</i>	ReportEmergency
<i>Participating actors</i>	Initiated by FieldOfficer Communicates with Dispatcher
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. The FieldOfficer activates the “Report Emergency” function of her terminal. 2. FRIEND responds by presenting a form to the officer. <i>The form includes an emergency type menu (general emergency, fire, transportation) and location, incident description, resource request, and hazardous material fields.</i> 3. The FieldOfficer completes the form by <i>specifying minimally the emergency type and description fields</i>. The FieldOfficer may also describe possible responses to the emergency situation <i>and request specific resources</i>. Once the form is completed, the FieldOfficer submits the form. 4. FRIEND receives the form and notifies the Dispatcher <i>by a pop-up dialog</i>. 5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. <i>All the information contained in the FieldOfficer’s form is automatically included in the Incident. The Dispatcher selects a response by allocating resources to the Incident (with the AllocateResources use case) and acknowledges the emergency report by sending a short message to the FieldOfficer.</i> 6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.
<i>Entry condition</i>	• ...

Figure 4-10 Refined description for the ReportEmergency use case. Additions emphasized in *italics*.

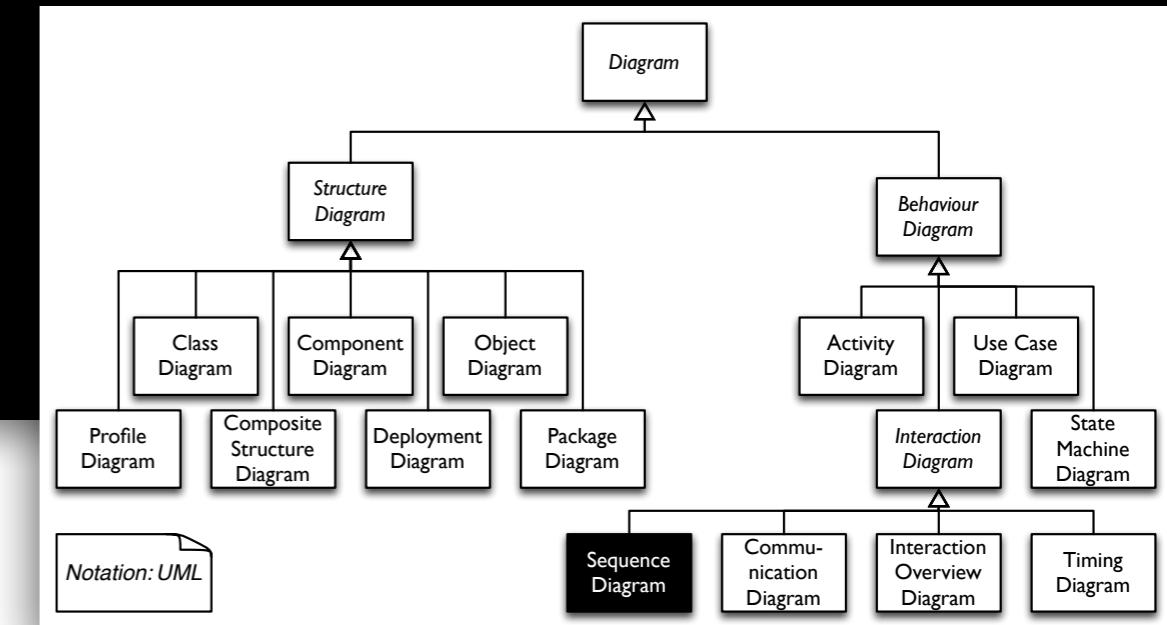
Beware!

- Use case diagrams are dangerous, they are a double-edged sword as the functional decomposition they facilitate can be endless.
- Use cases must include also the text:
 - Name of Use Case
 - Actors: description of Actors involved in use case.
 - Entry condition: “This use case starts when...”.
 - Flow of Events: free form, informal natural language.
 - Exit condition: “This use cases terminates when...”.
 - Exceptions: describe what happens if things go wrong.
 - Quality Requirements: nonfunctional requirements, constraints.

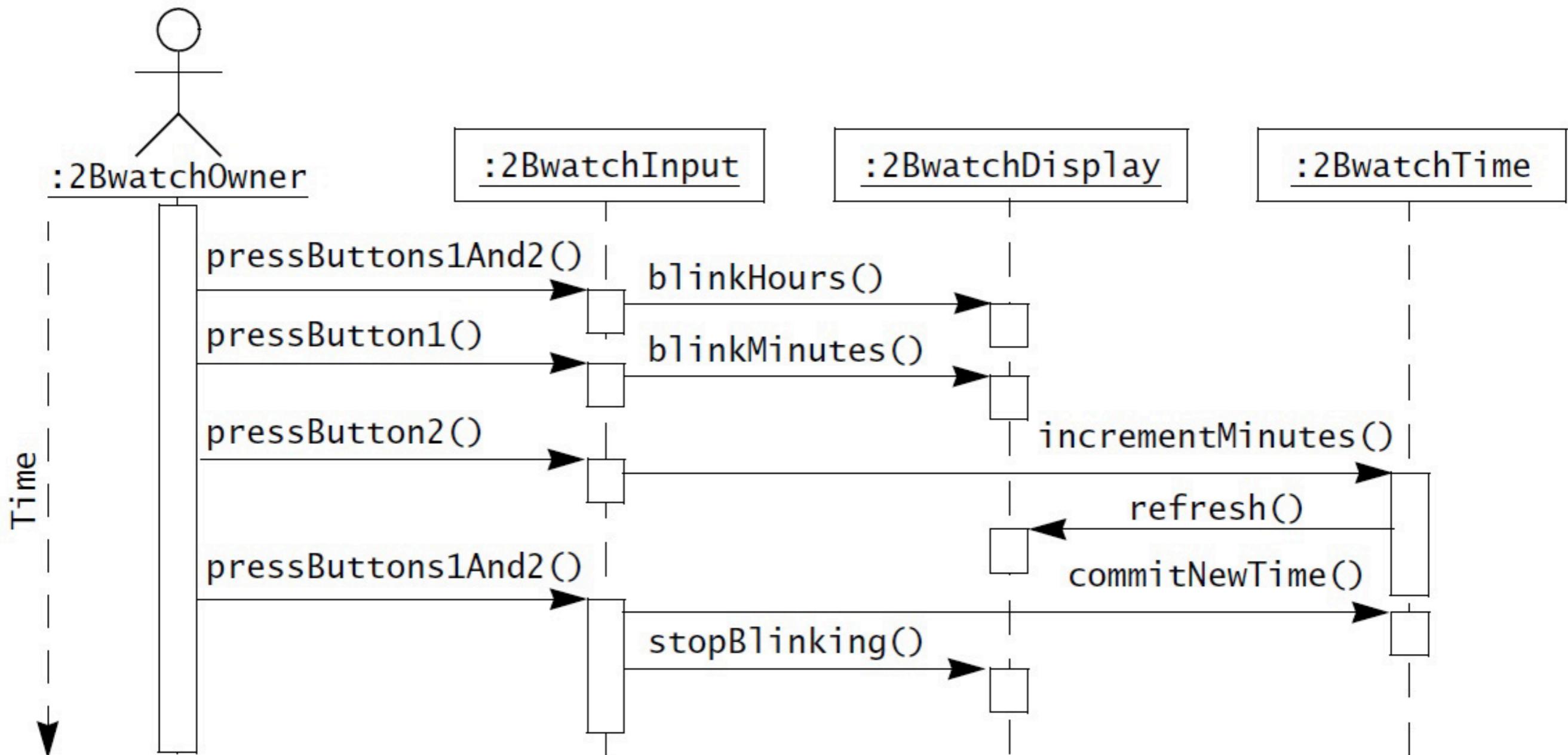
Exercise

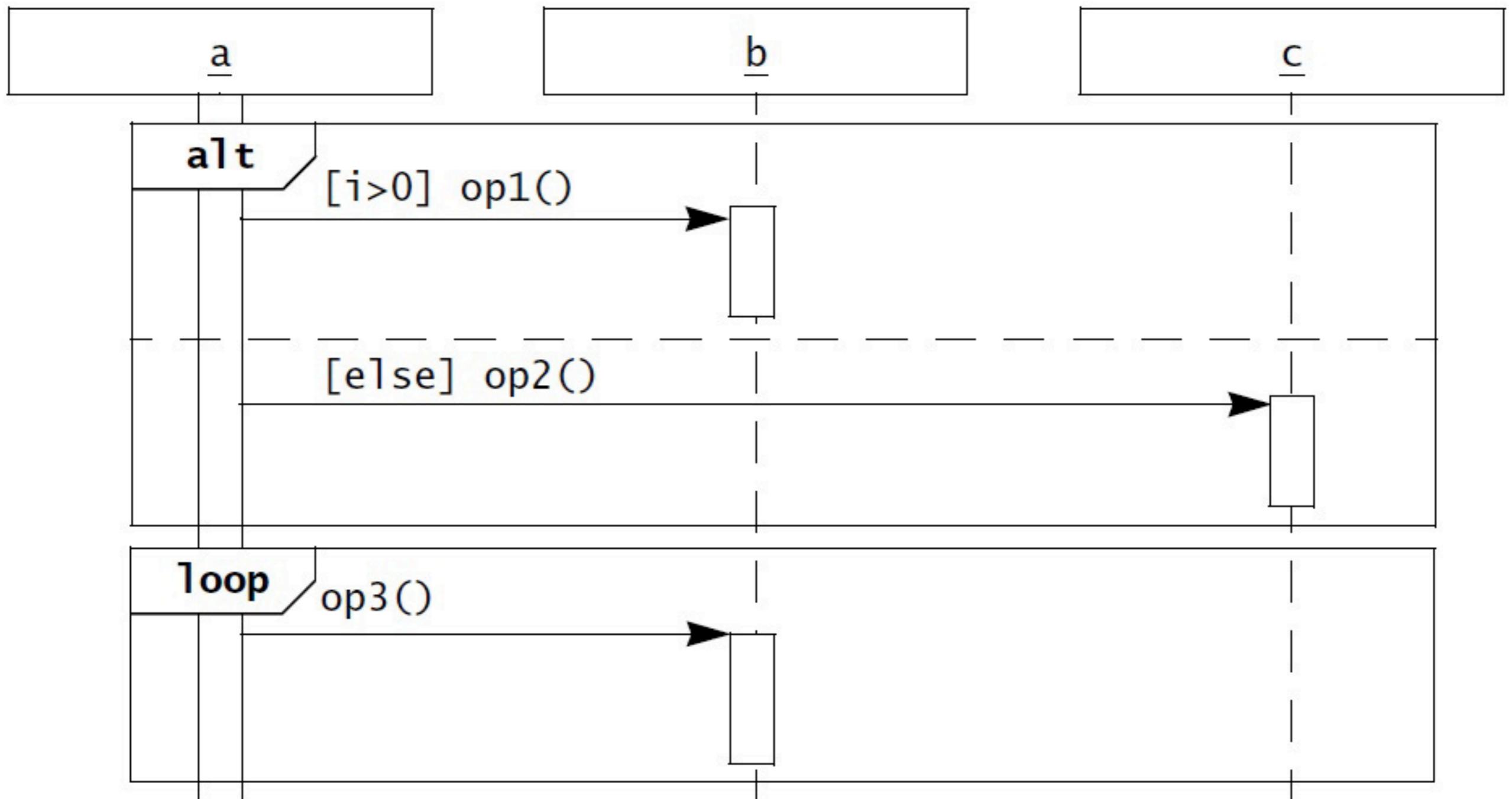
- Based on the 10 functional requirements you identified about the Project for today, draw the sticky-man part of the use case diagram.

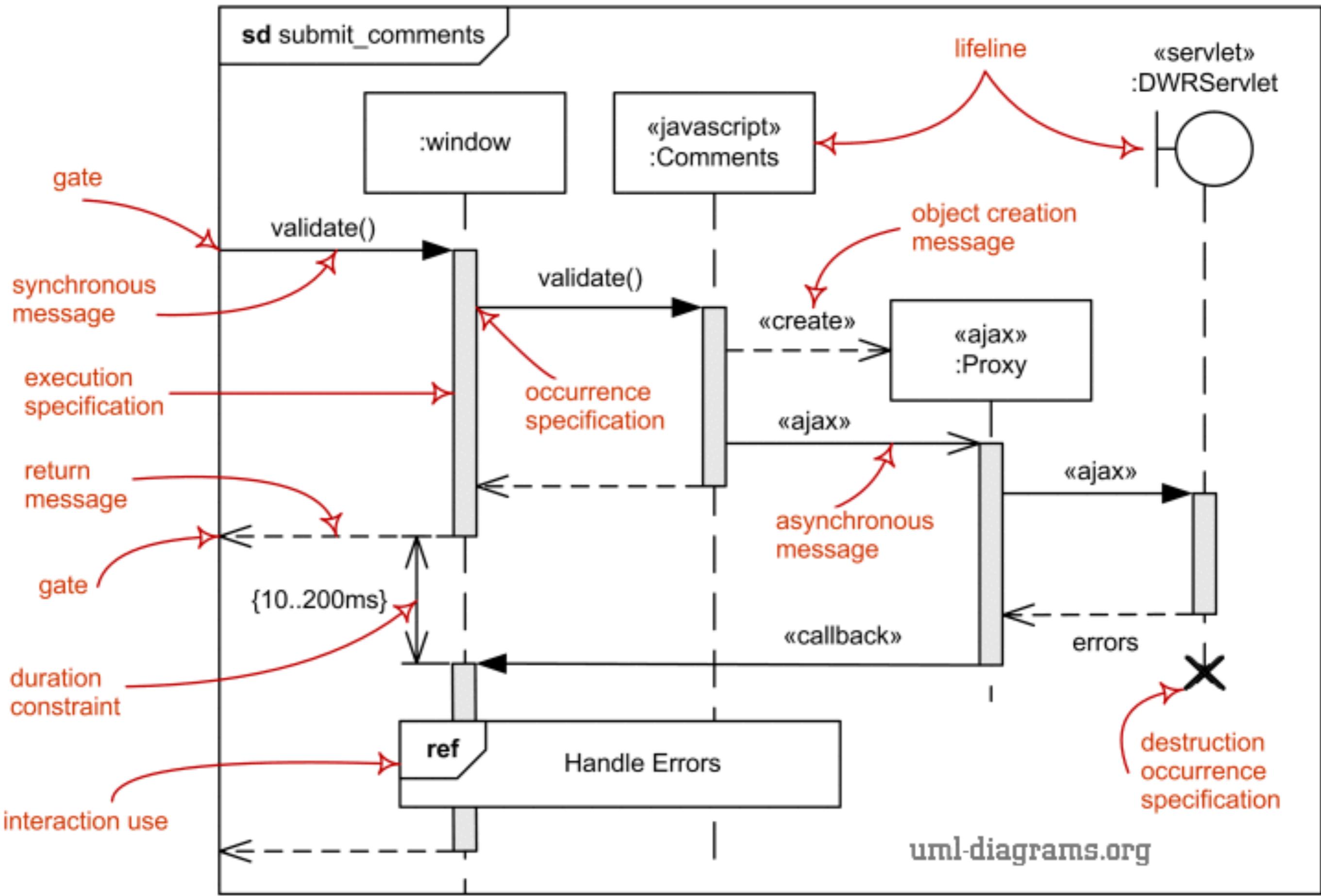
Sequence diagram



- Behaviour interaction diagrams.
- They specify the dynamic behaviour of a system as interactions between different objects.
- They capture the events that happen at runtime.
- They are “formal” and are used for specification and/or documentation.
- They focus on time and flow (e.g., concurrency) between objects.

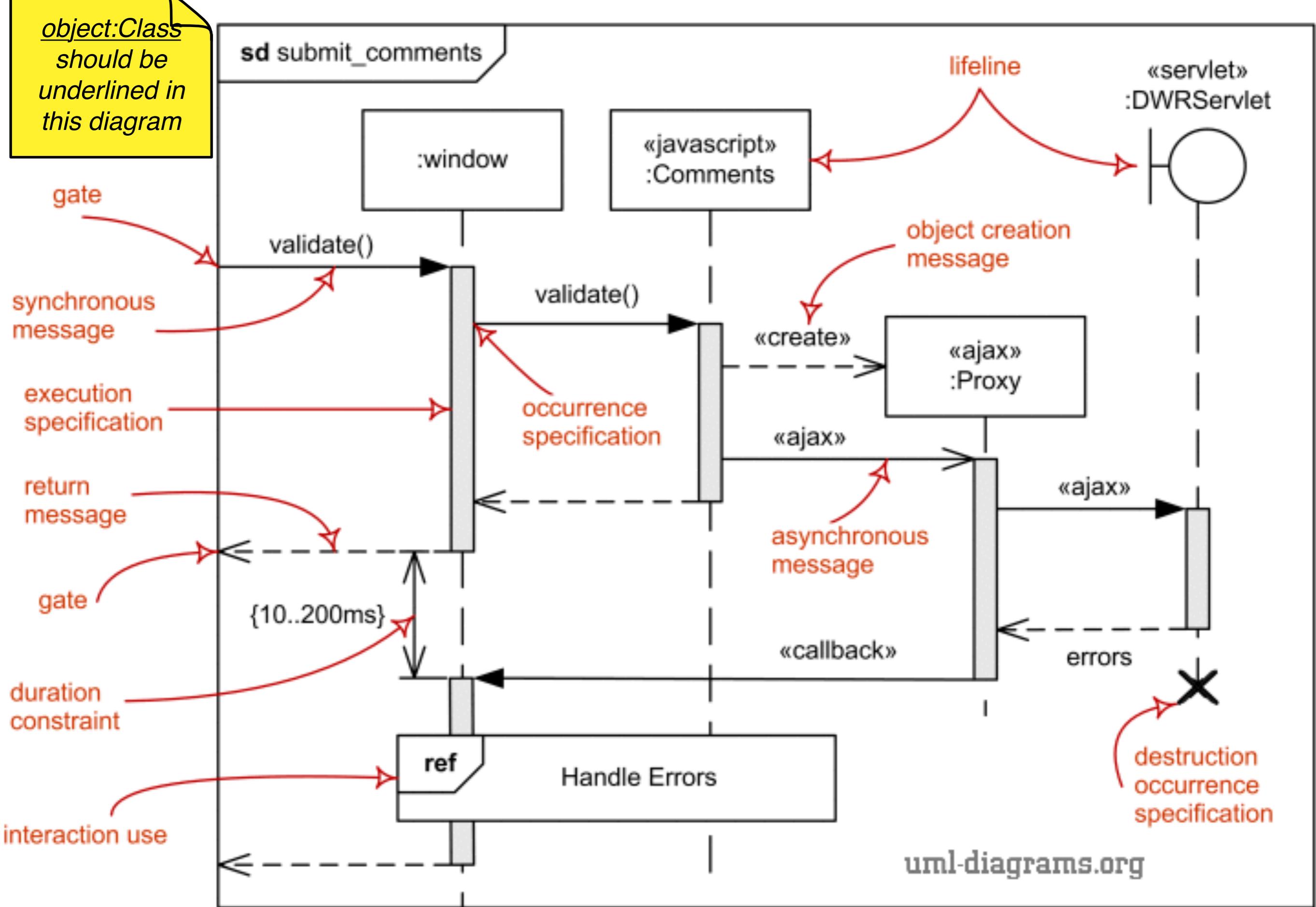






uml-diagrams.org

object:Class
should be
underlined in
this diagram

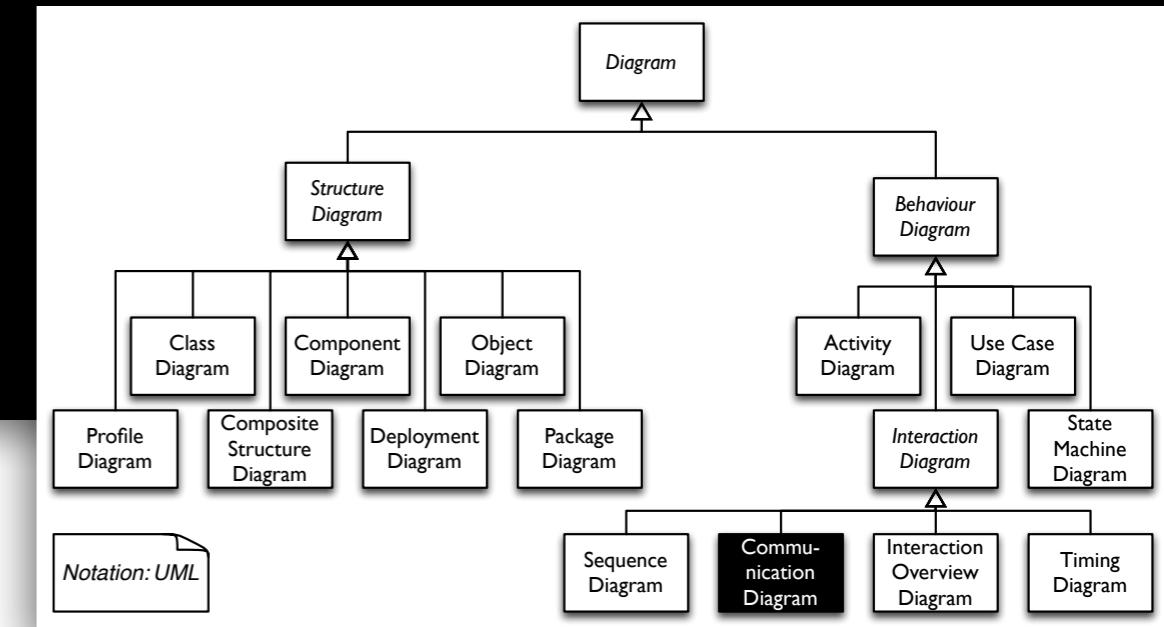


uml-diagrams.org

Other details

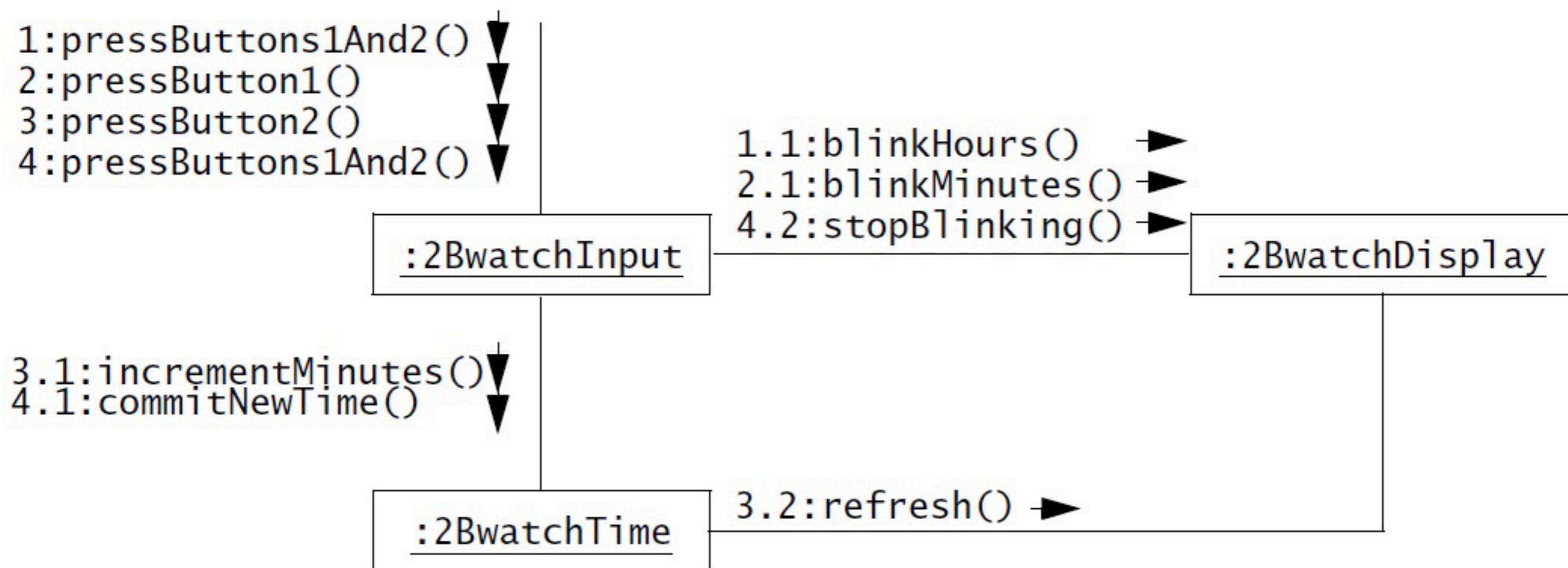
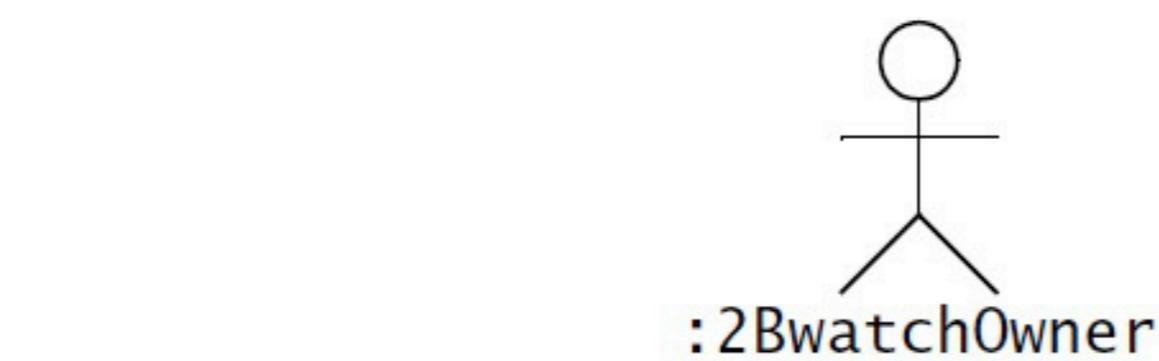
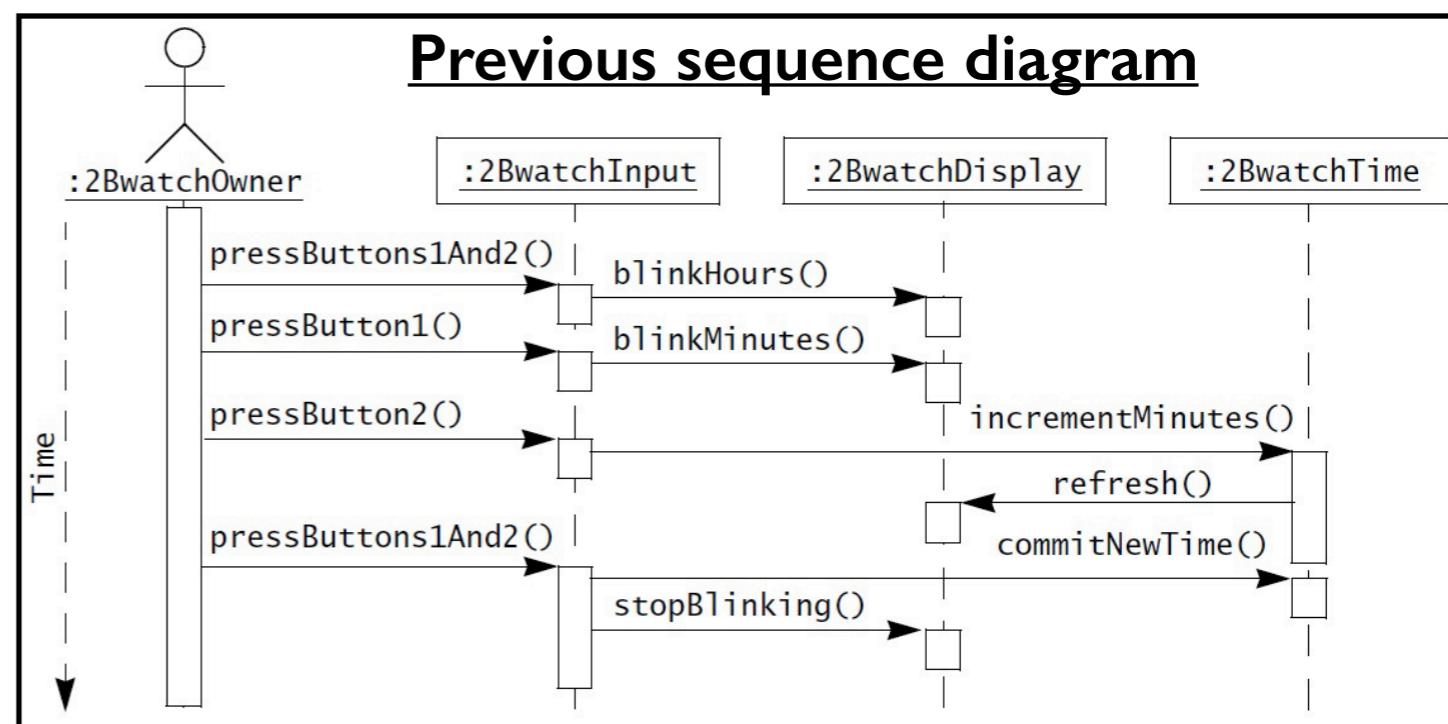
- object:Class
- Life line = the object has been created (new).
- Execution specification or activation box = a method of the object has its activation record on the stack.
- The “return” is not always necessary. If it is obvious, it can be omitted.
- The sequence diagrams provide an immediate overview of where the methods are distributed across the object/classes.

Communication diagram

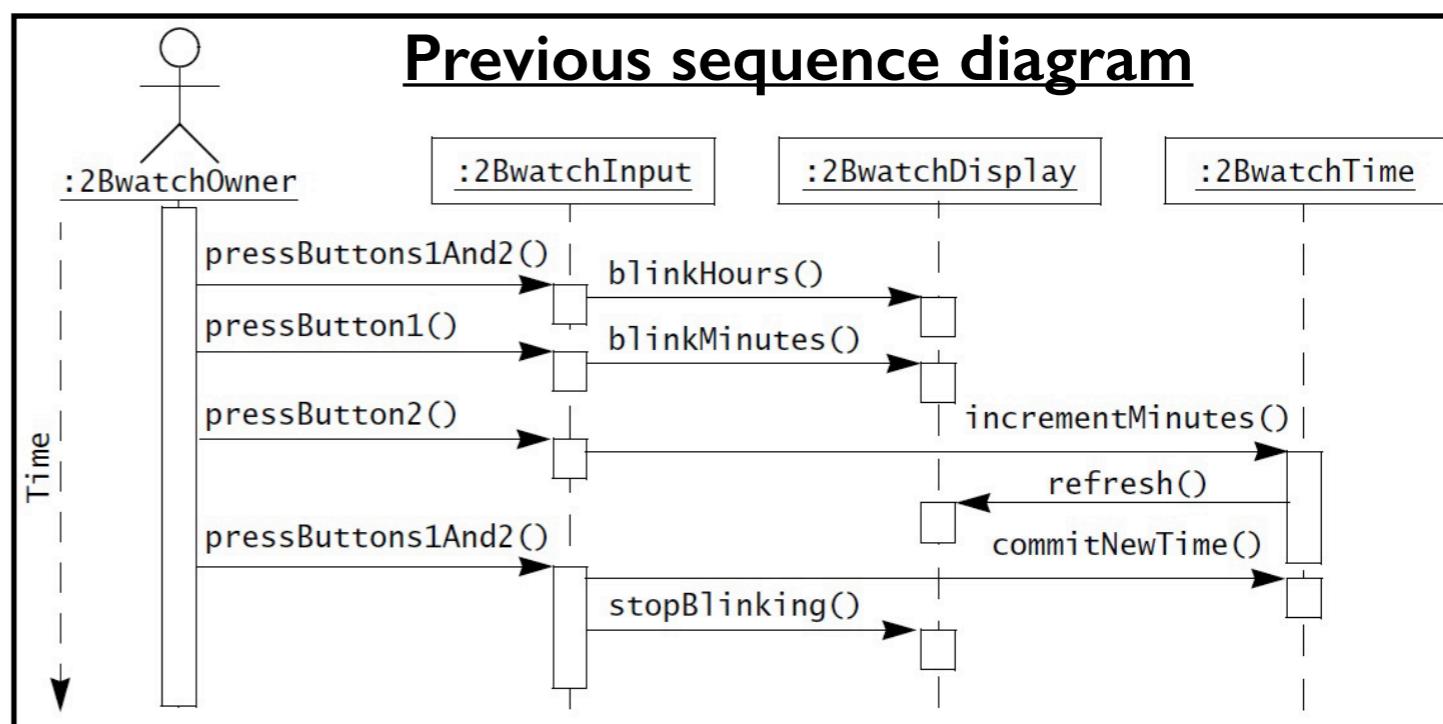


- Behaviour interaction diagrams.
- They specify the dynamic behaviour of a system as interactions between different objects.
- They contain the same information as the sequence diagrams.
- They are object diagrams + informations on the method invocations.
- They focus on the communication rather than the temporal order.
- They are more “informal” and used for sketching.

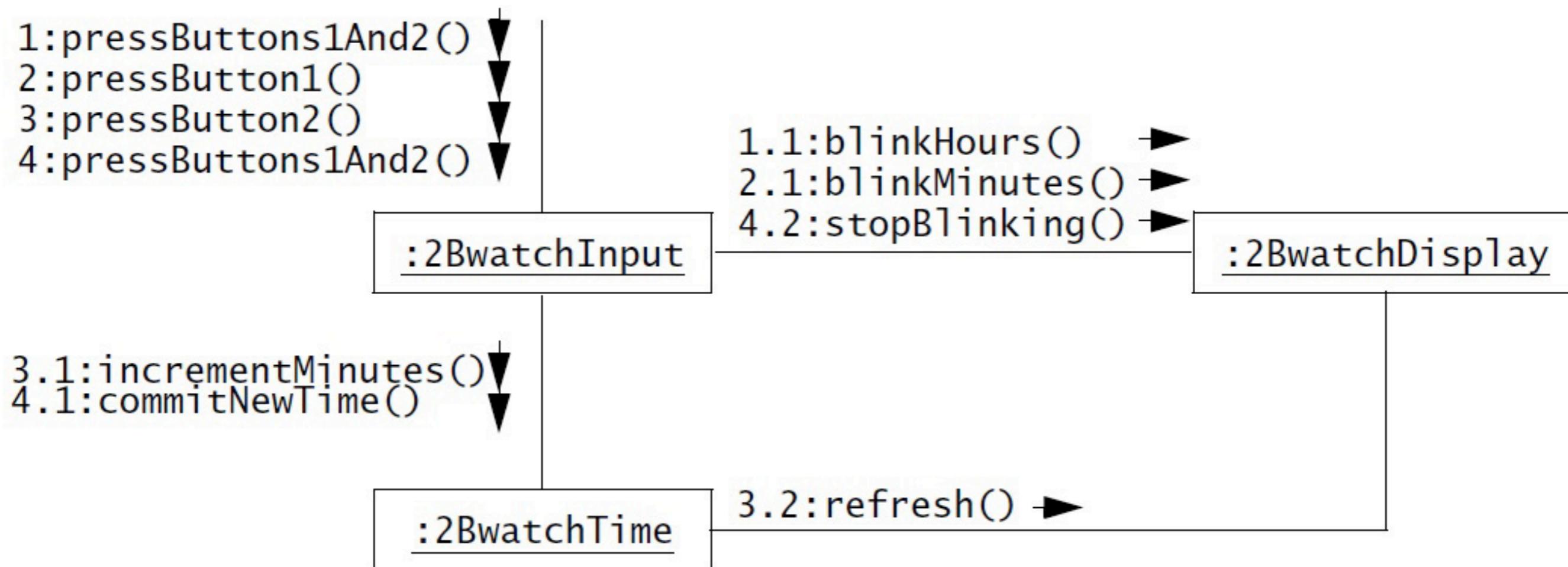
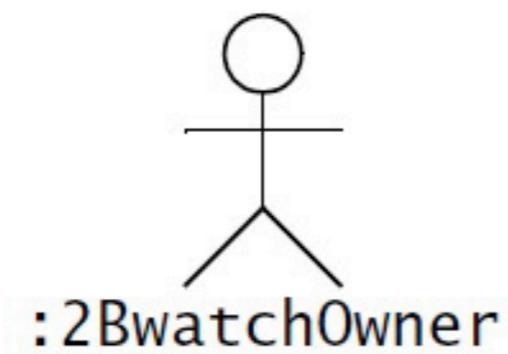
Previous sequence diagram

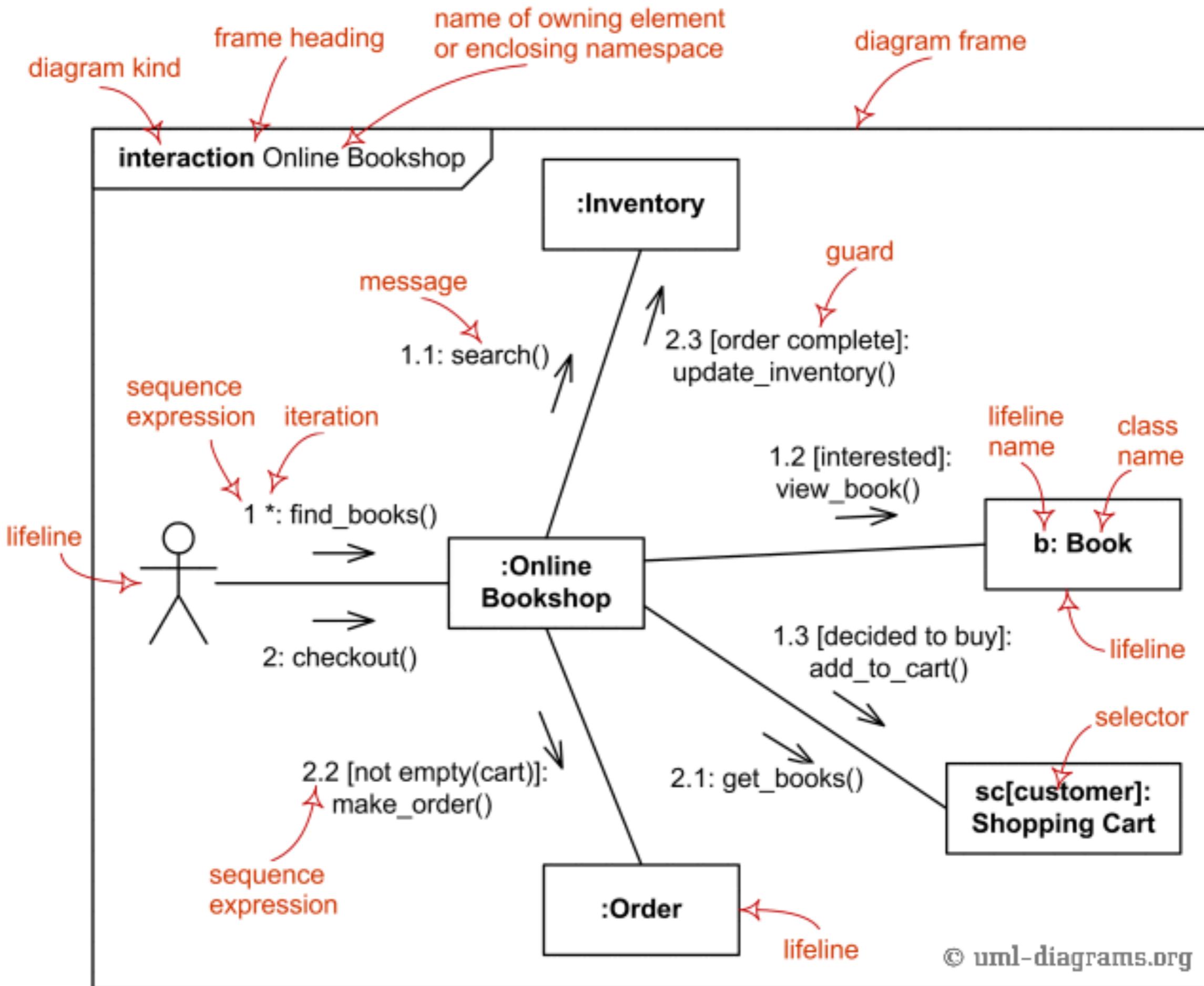


Previous sequence diagram

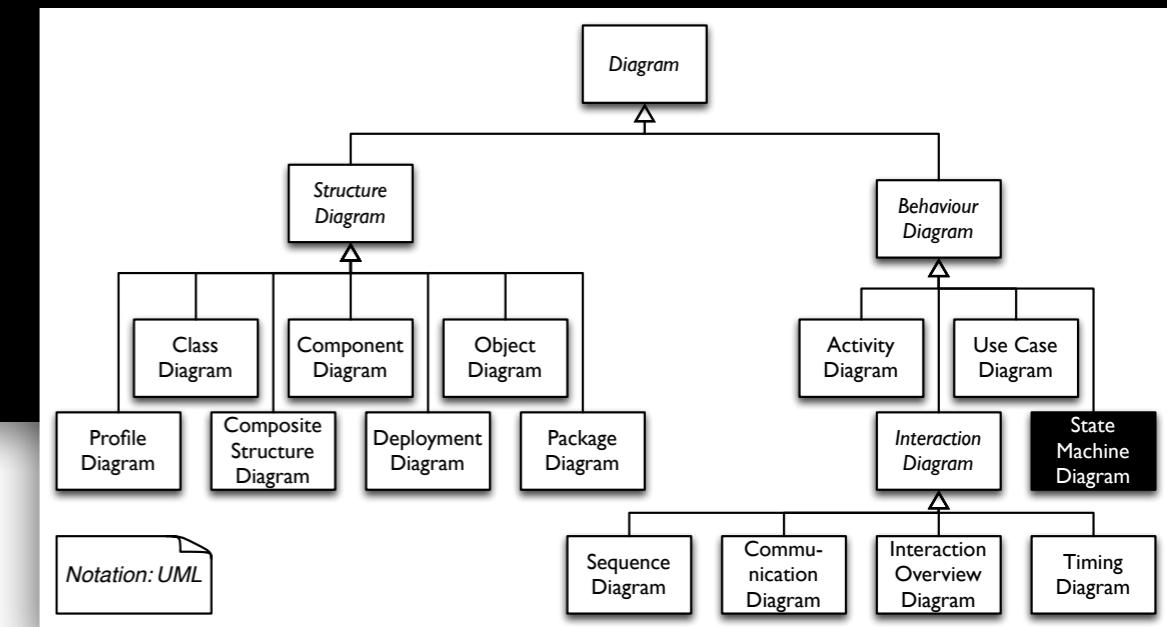


*The arrows
should be
open here*

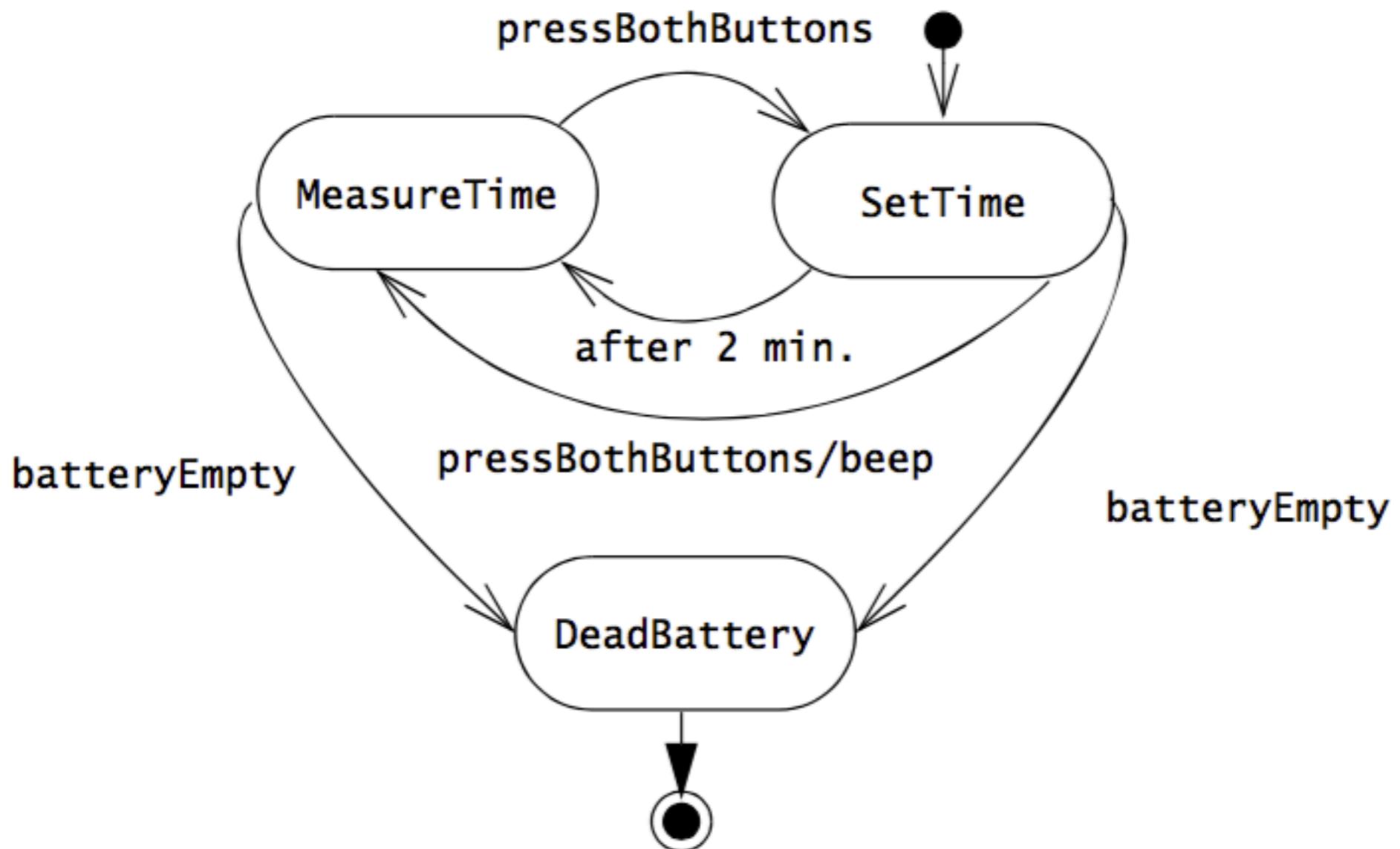
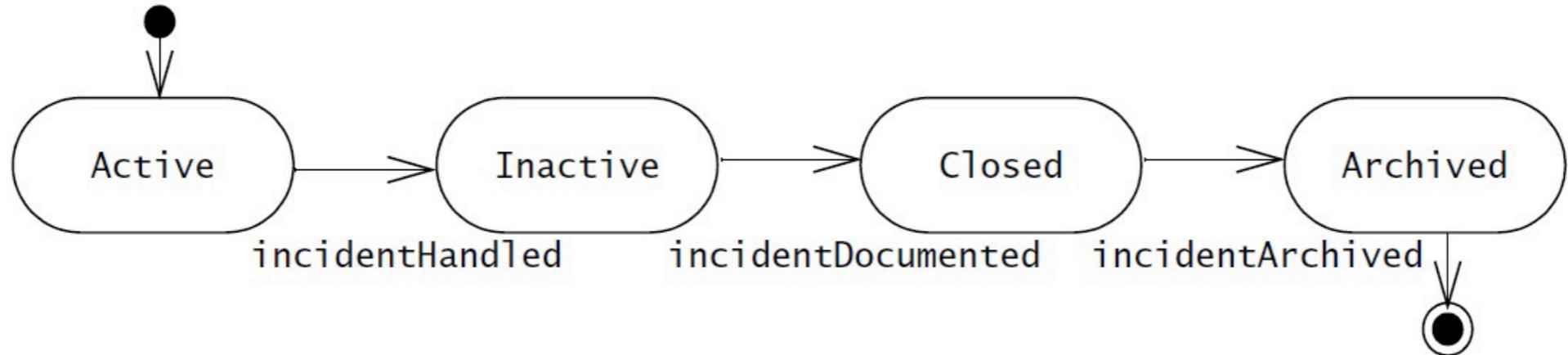


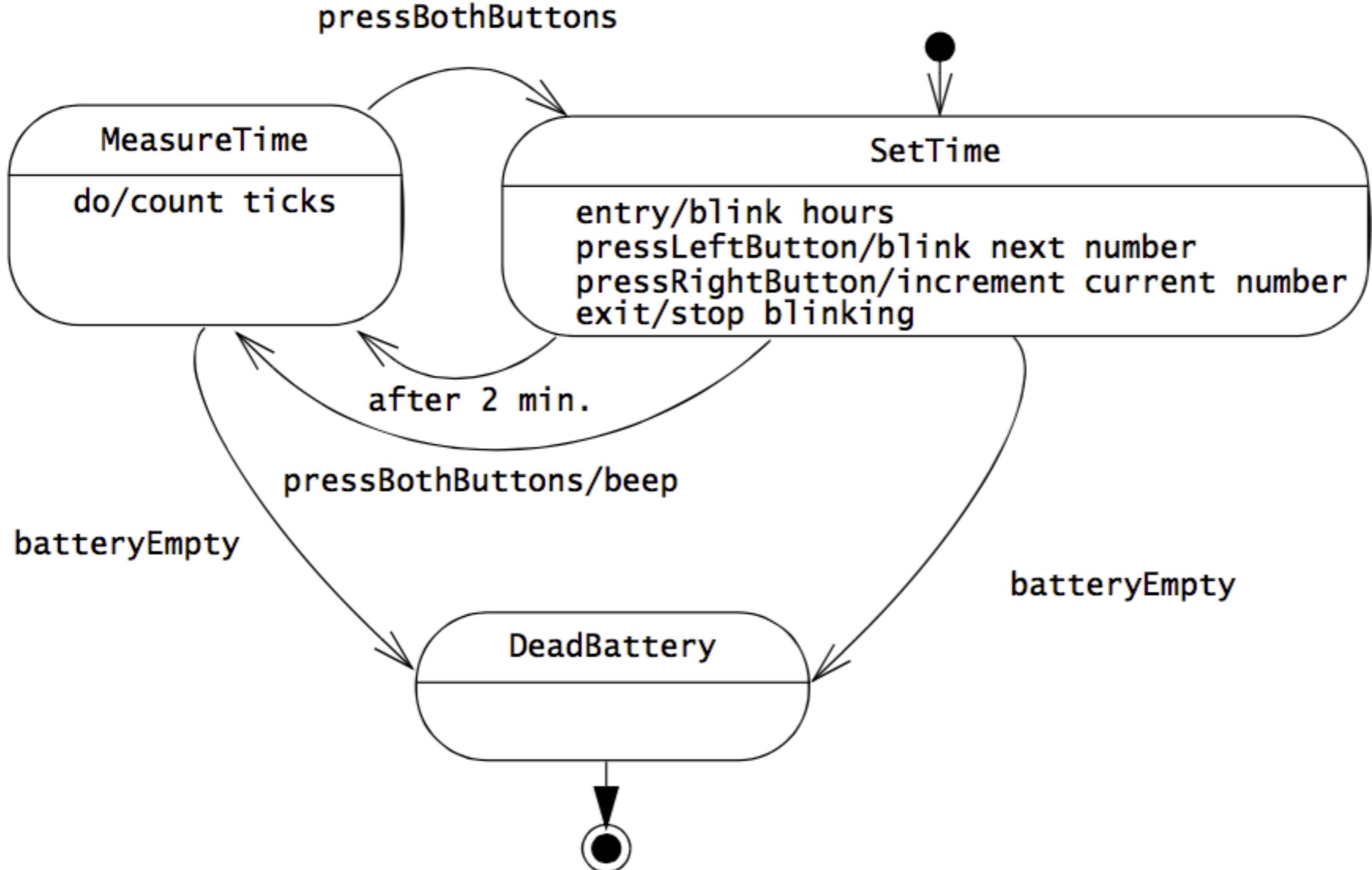


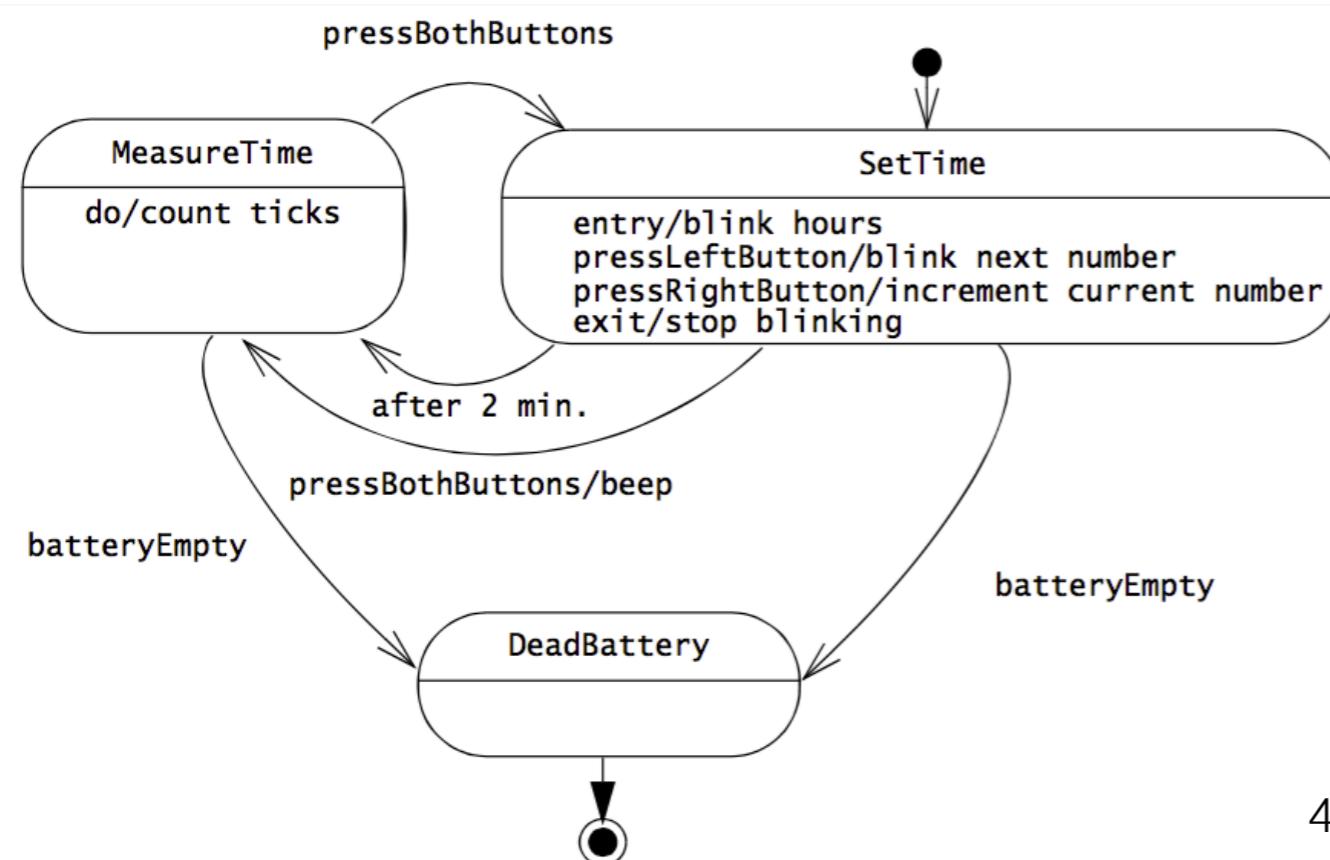
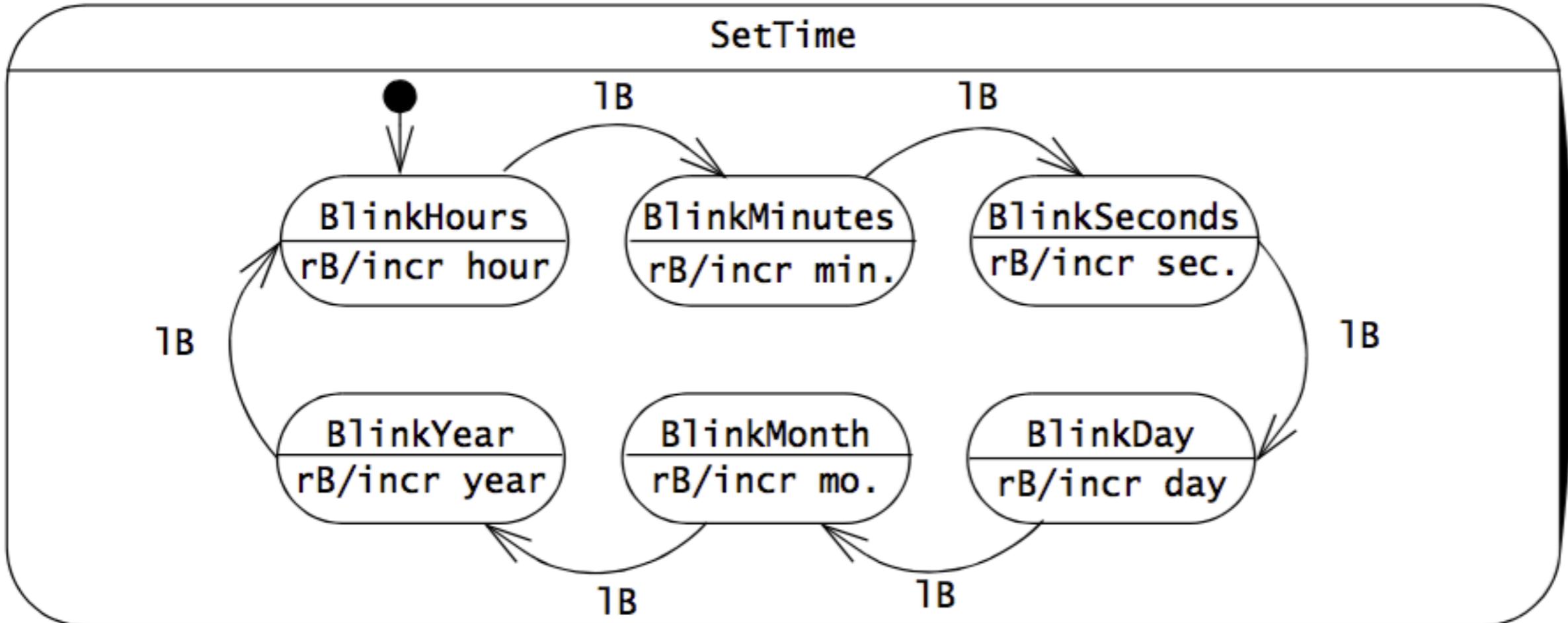
State machine diagram

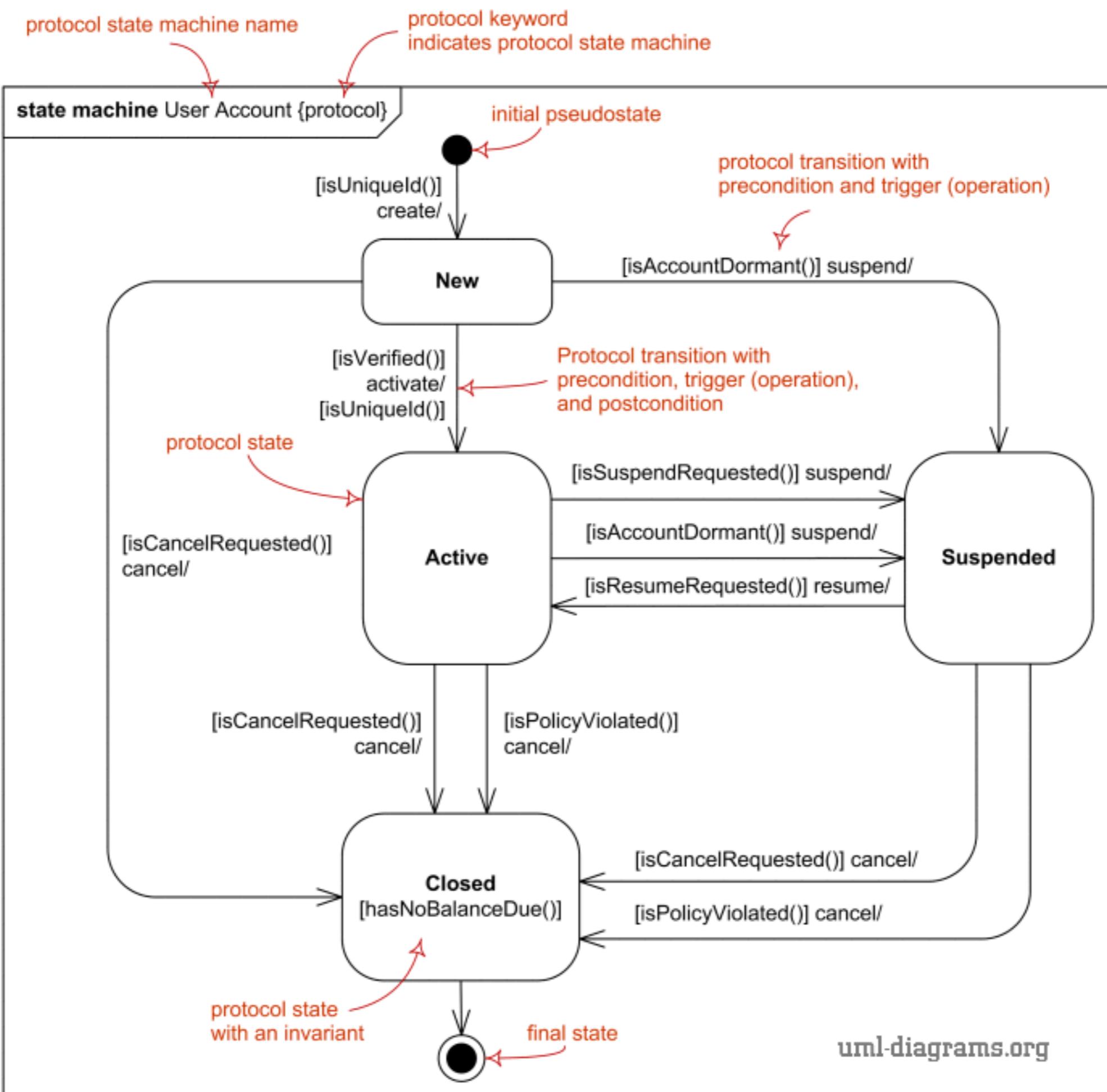


- Behaviour diagrams.
- They specify the dynamic behaviour of a single object.
- They model the sequence of states an object goes through at runtime in reaction to external events.
- They include an initial and final state.
- They show states and transactions.
- Substates are permitted.
- Transaction are labeled with events, guards, etc.

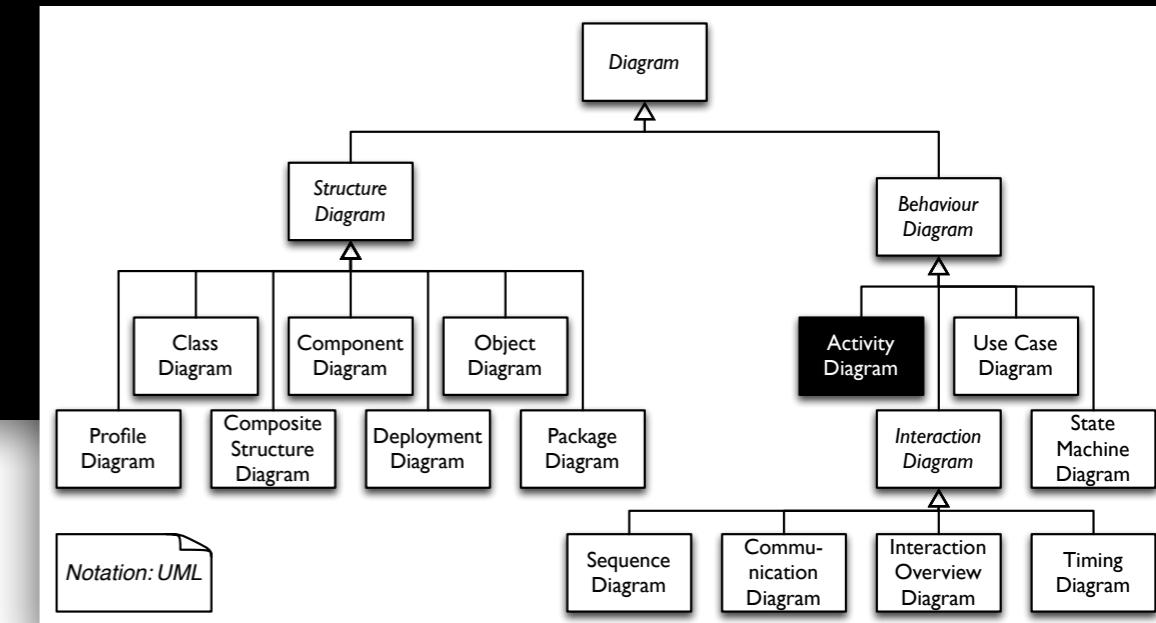




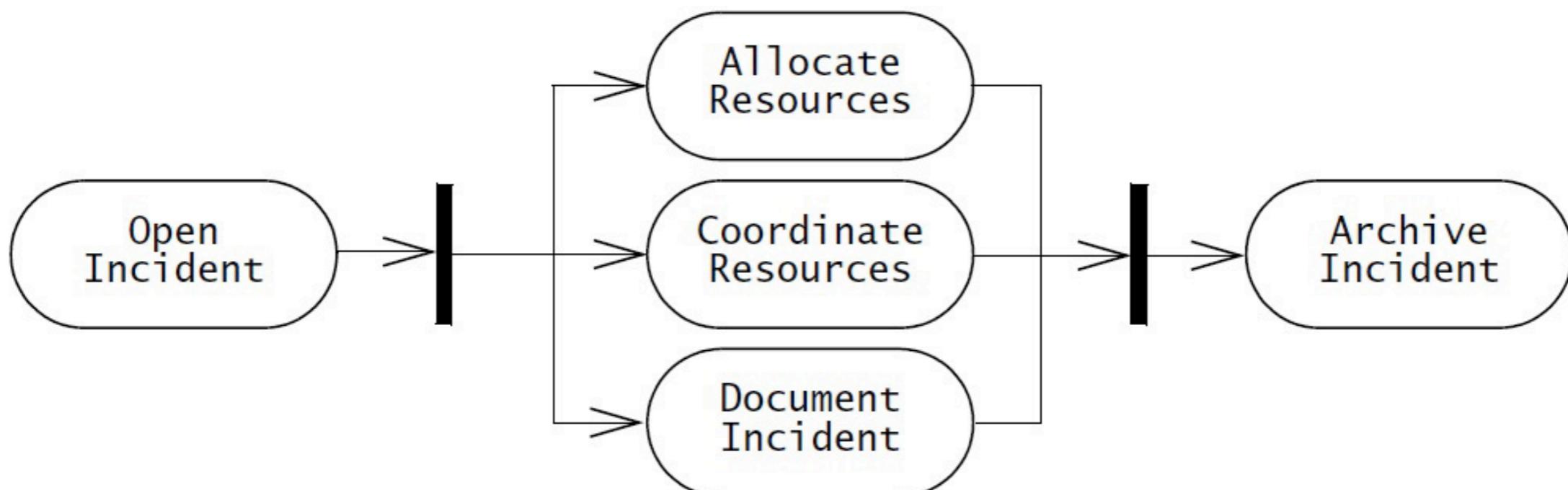
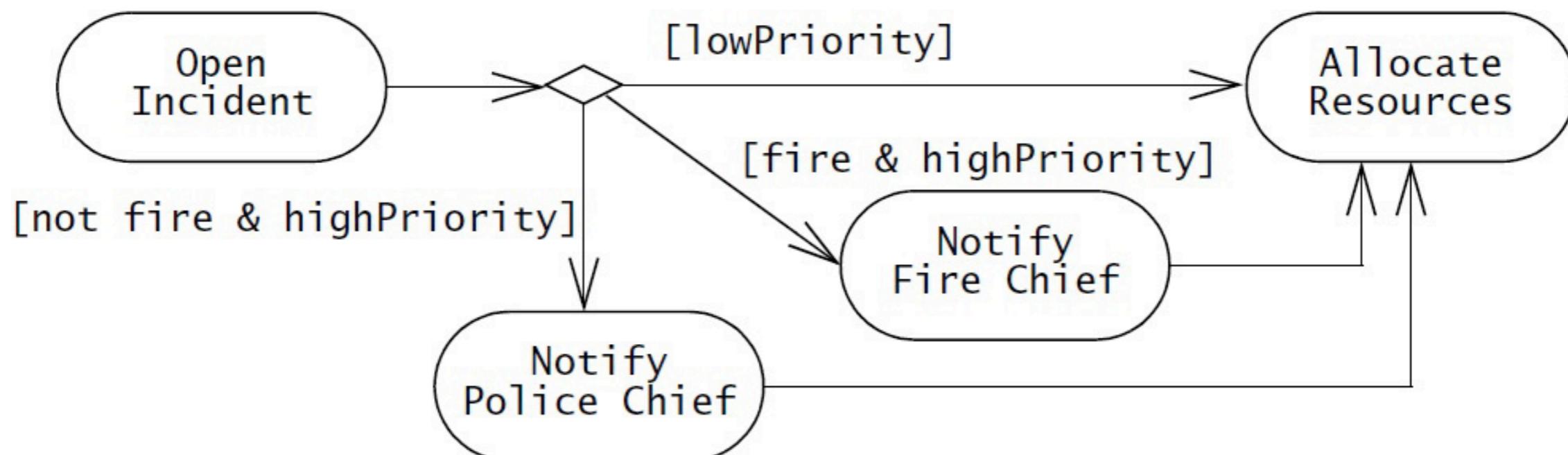
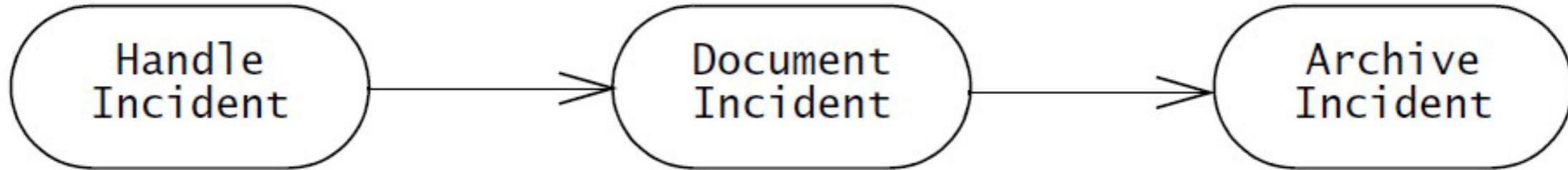


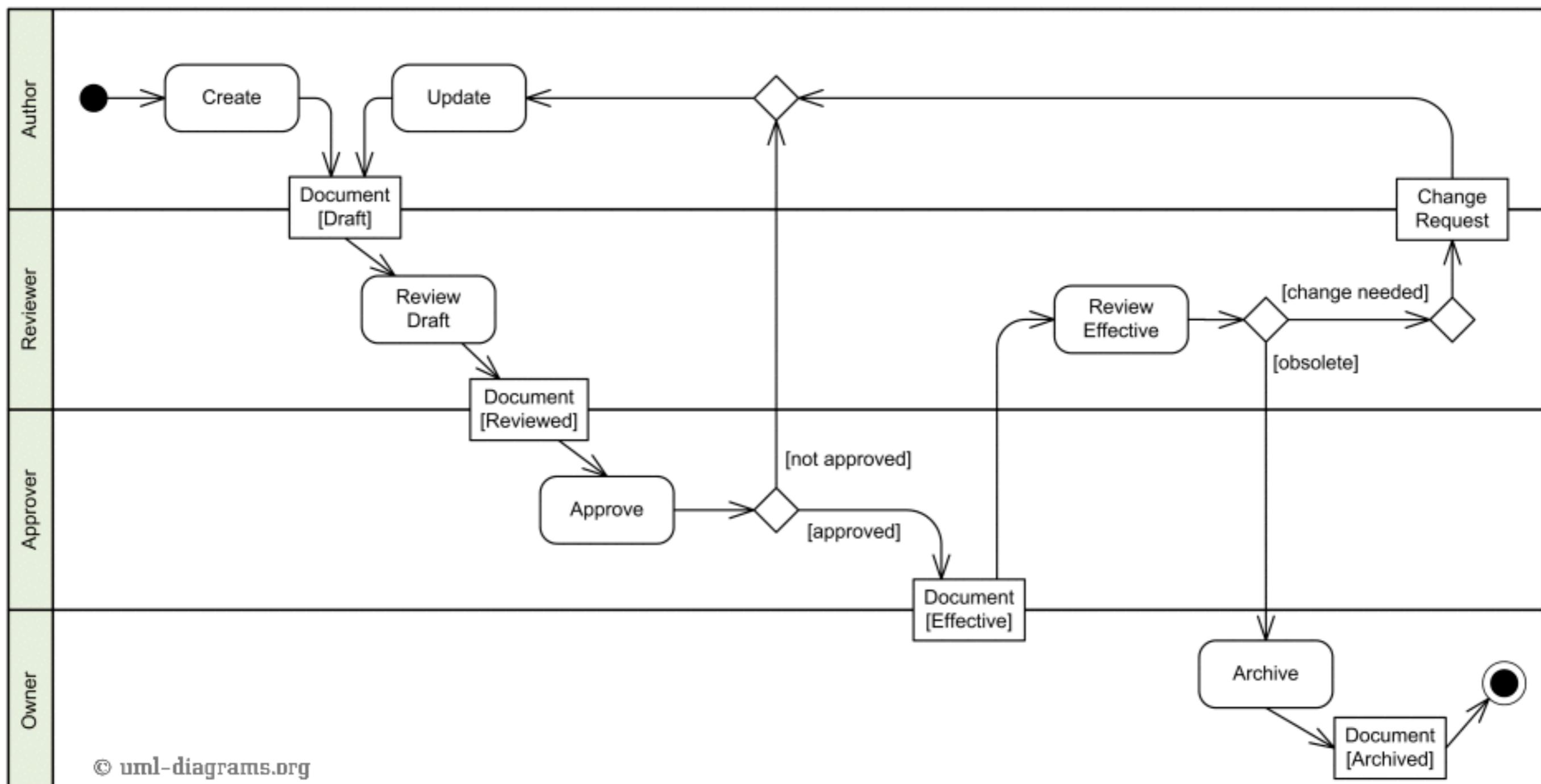


Activity diagram

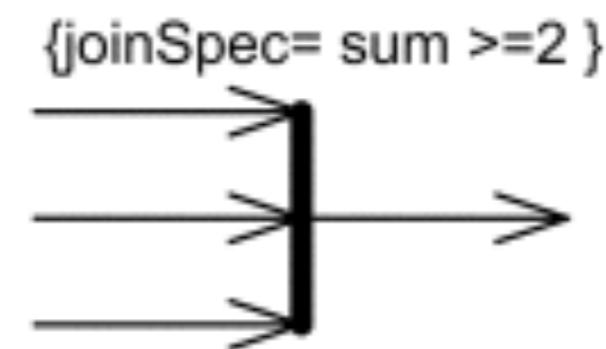
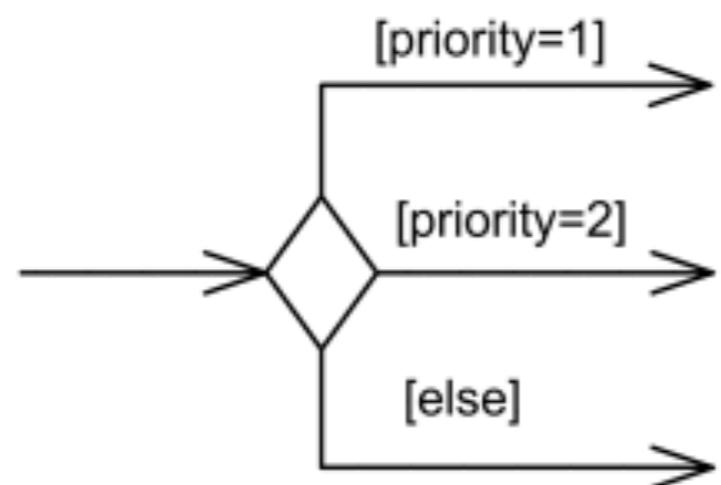
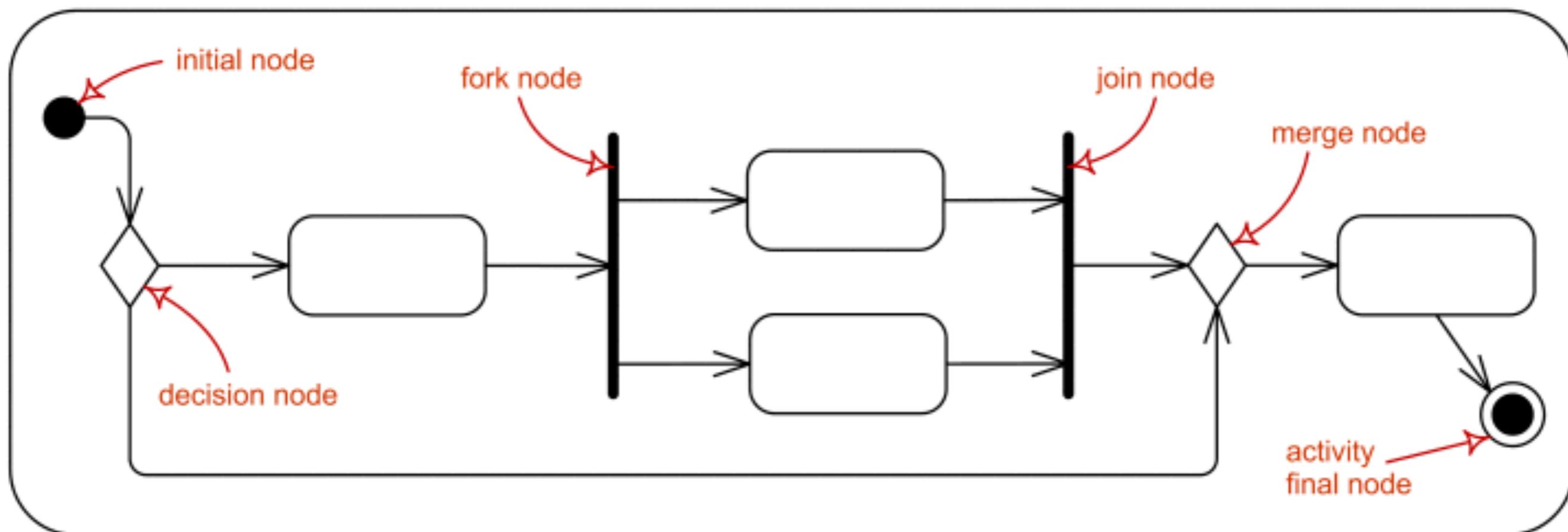


- Behaviour diagrams.
- They model the dynamic behaviour of a sub-system.
- They focus on lower level behaviour.
- They are realized on terms of one or several sequences of activities.
- Also known as flowchart.
- They include an initial and final state.
- They show decisions/merges and forks/joins.
- They can be partitioned in swimlanes to highlight concerns.



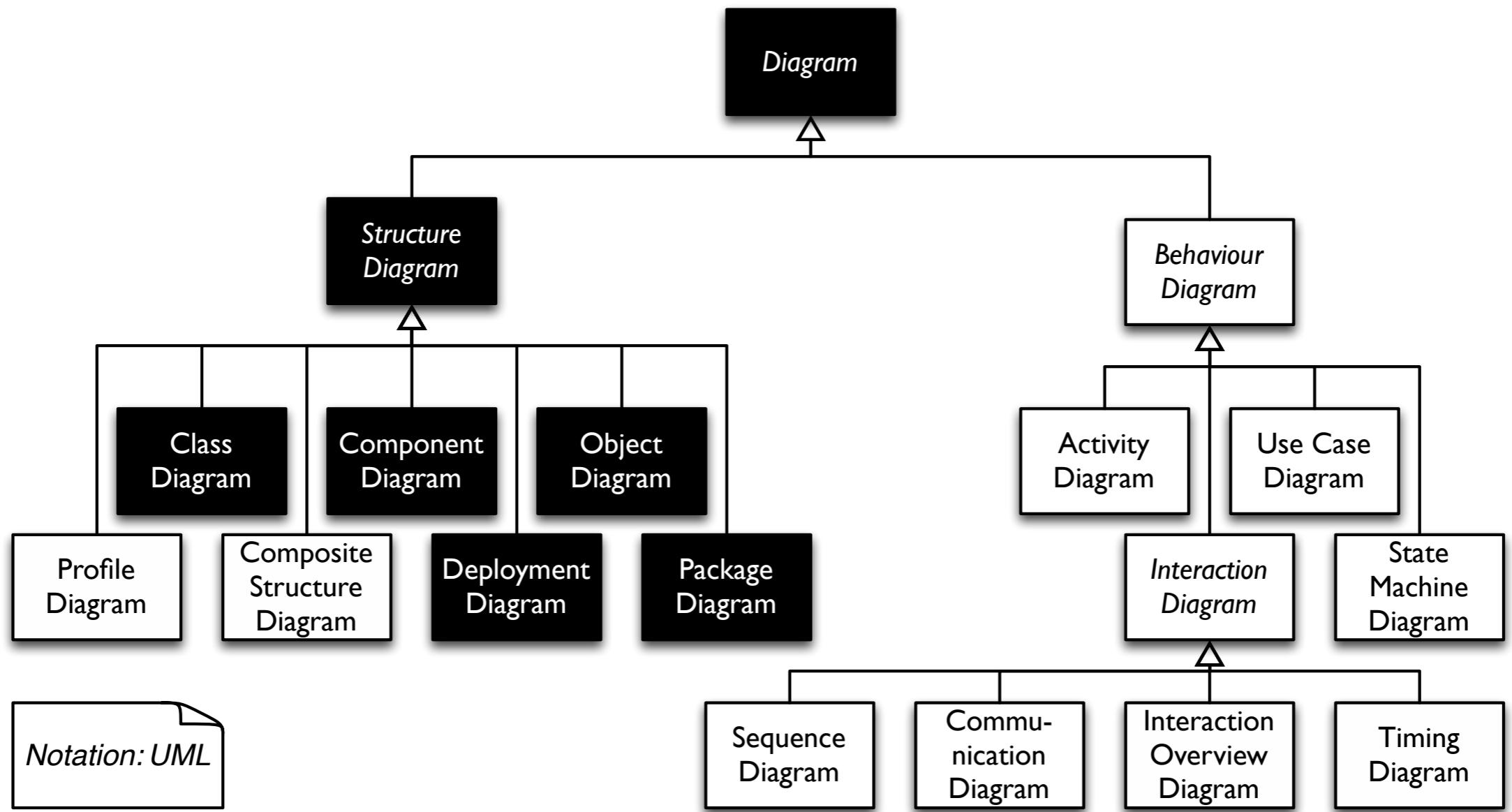


© uml-diagrams.org



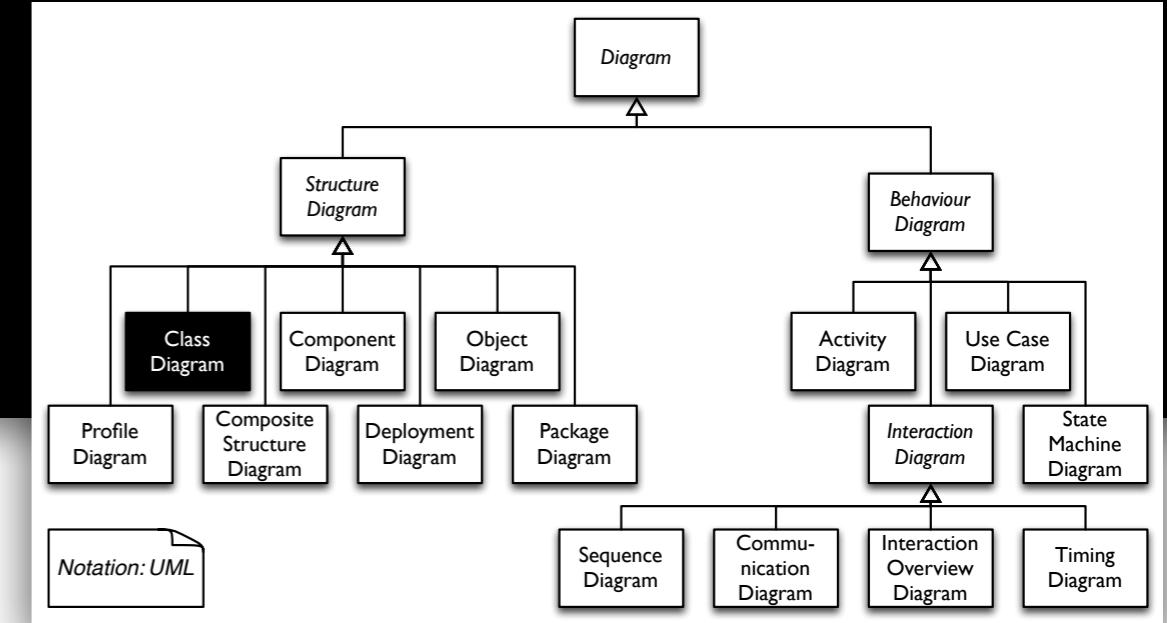
Exercise

- Based on the 10 functional requirements you identified about the Project for today and the Project description you received during the last exercise lecture, draw:
 - either an activity diagram representing an in-depth decomposition of an interesting activity you identified in the project;
 - or a state machine diagram describing the specification of the behaviour of an interesting object you identified in the project.



Diving into UML (2nd part)

Class diagram

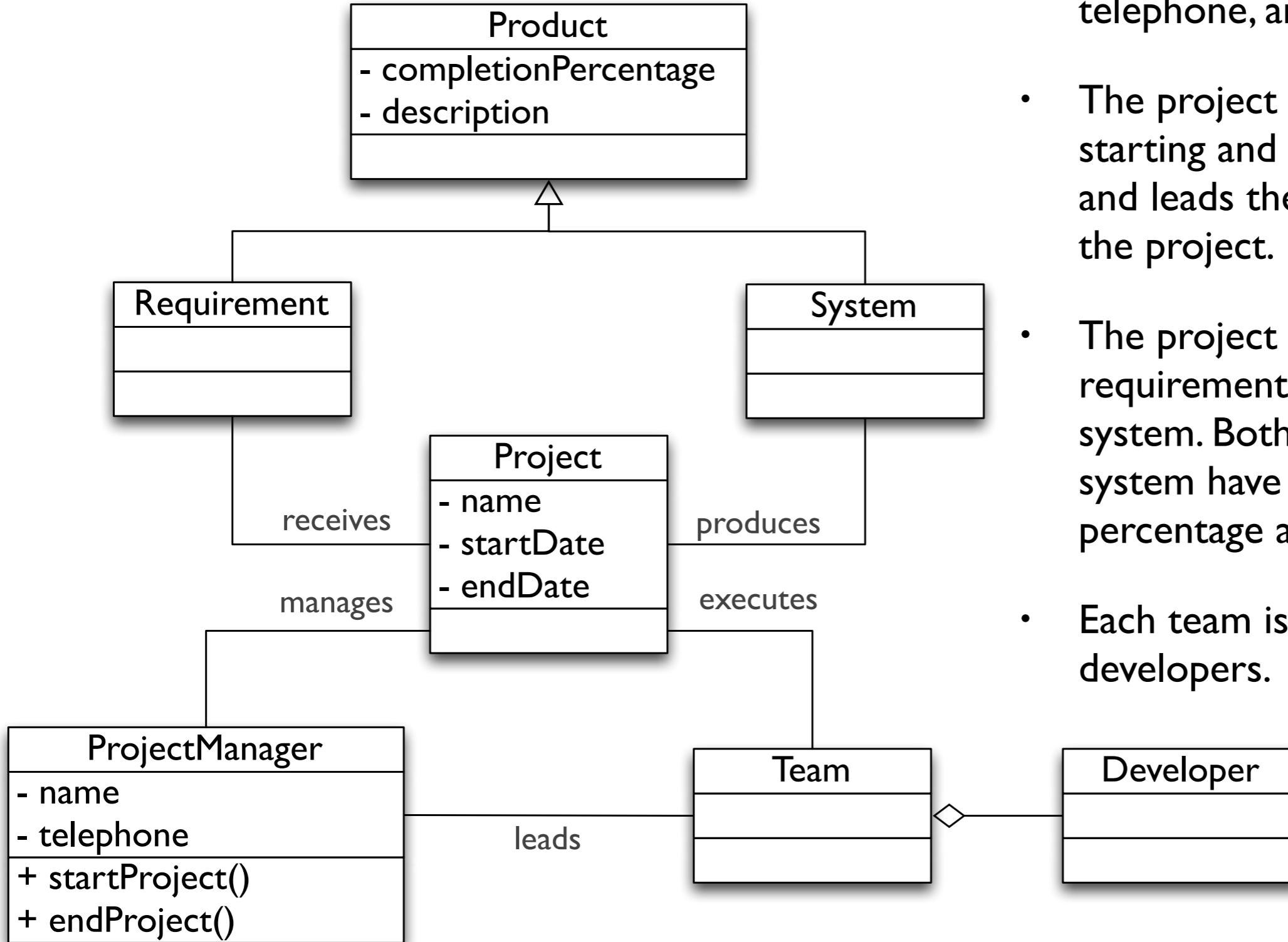


- Structure diagrams.
- Class diagrams are abstractions specifying attributes and behaviour of a set of objects.
- Relationships between classes (and interfaces and packages):
 - Dependancy
 - Association
 - Aggregation/Composition
 - Inheritance

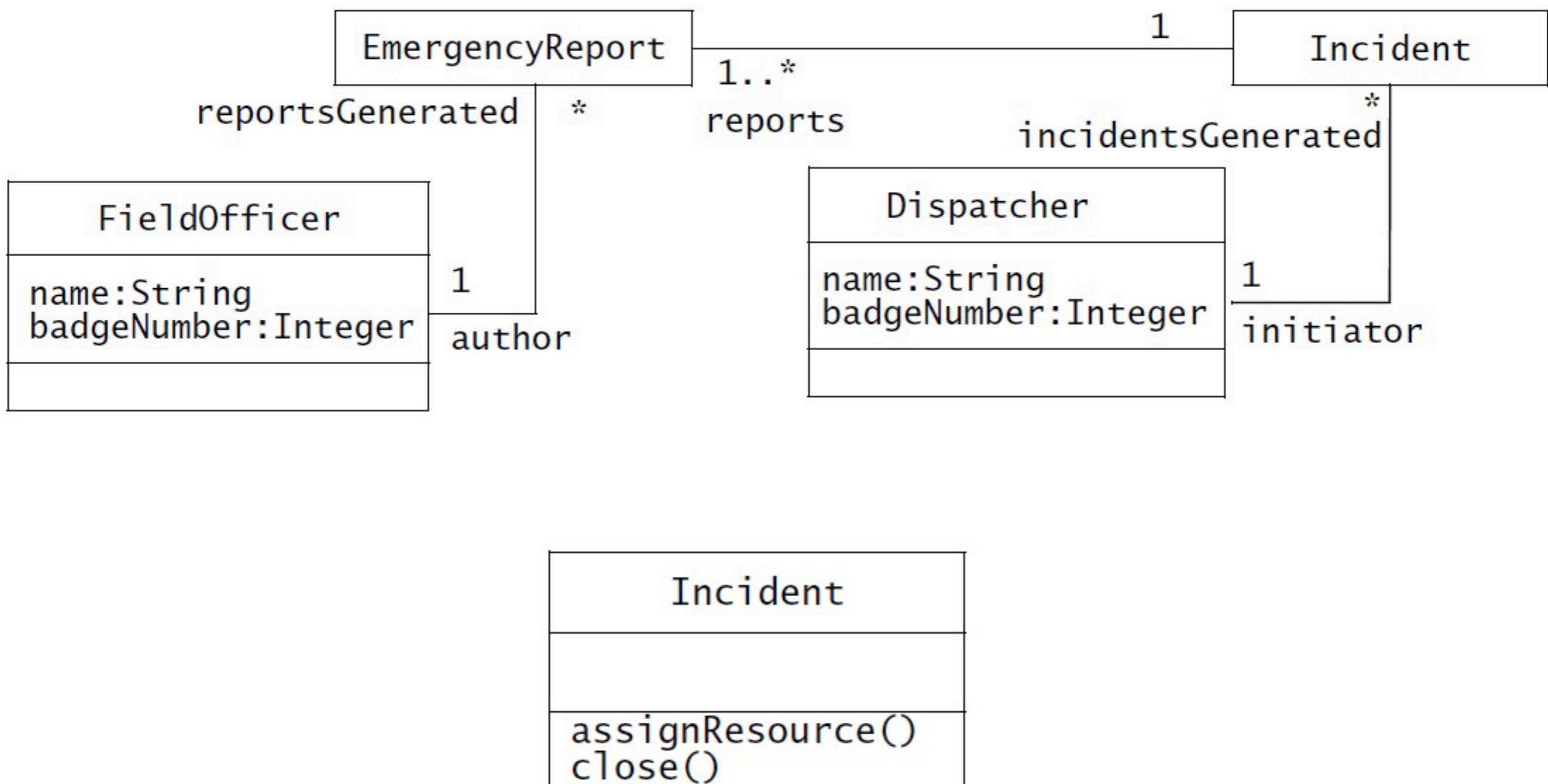
An example

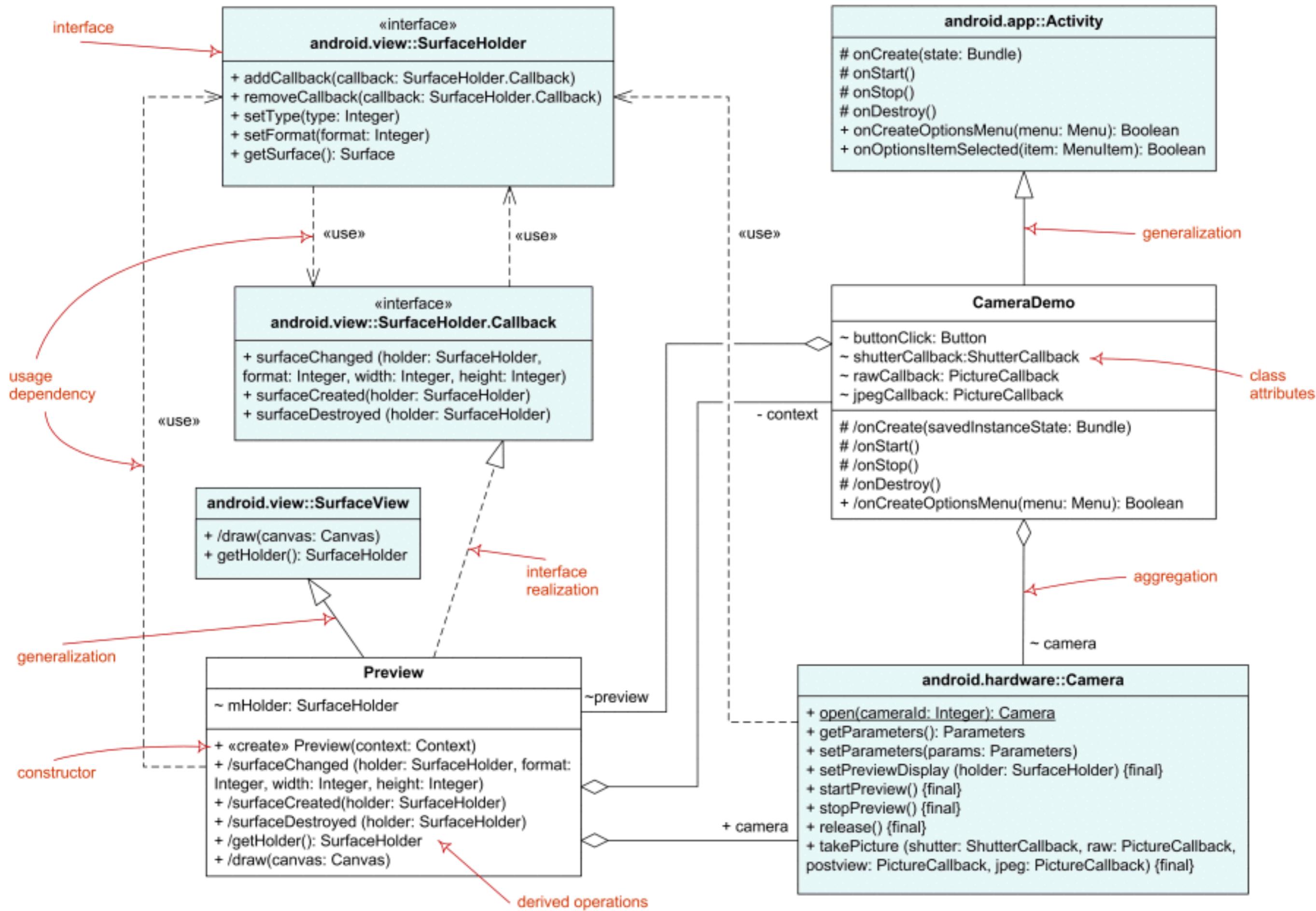
- Each project has a name, a start date, and an end date.
- Each project is associated to a project manager, a name, a telephone, and a team.
- The project manager manages (by starting and terminating) a project and leads the team associated with the project.
- The project receives as input the requirements and produces a system. Both requirements and the system have a completion percentage and a description.
- Each team is composed by developers.

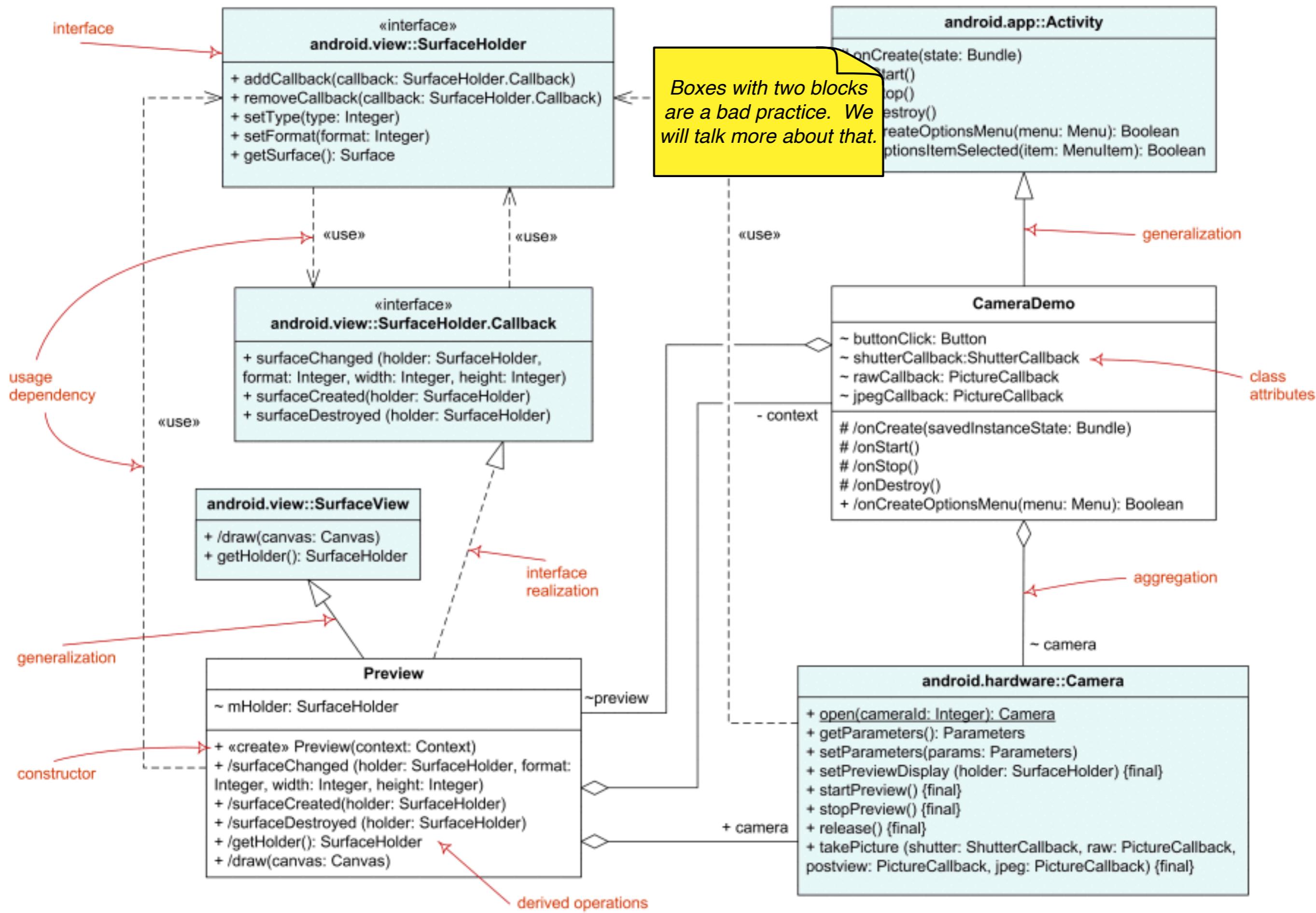
- Each project has a name, a start date, and an end date.
- Each project is associated to a project manager, a name, a telephone, and a team.
- The project manager manages (by starting and terminating) a project and leads the team associated with the project.
- The project receives as input the requirements and produces a system. Both requirements and the system have a completion percentage and a description.
- Each team is composed by developers.



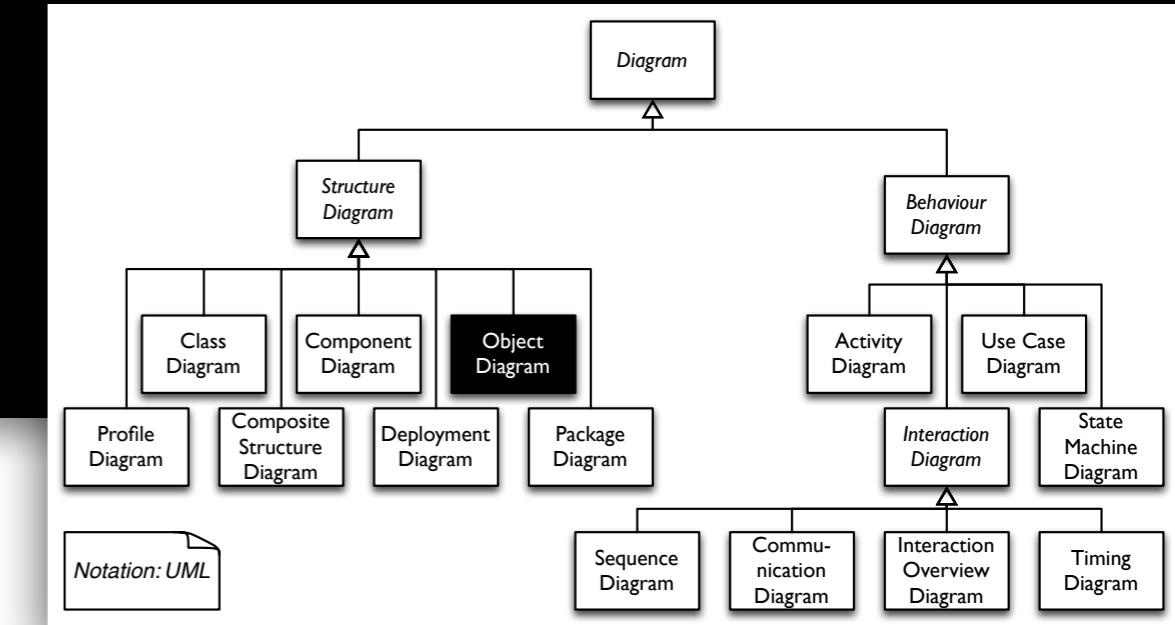
Another example





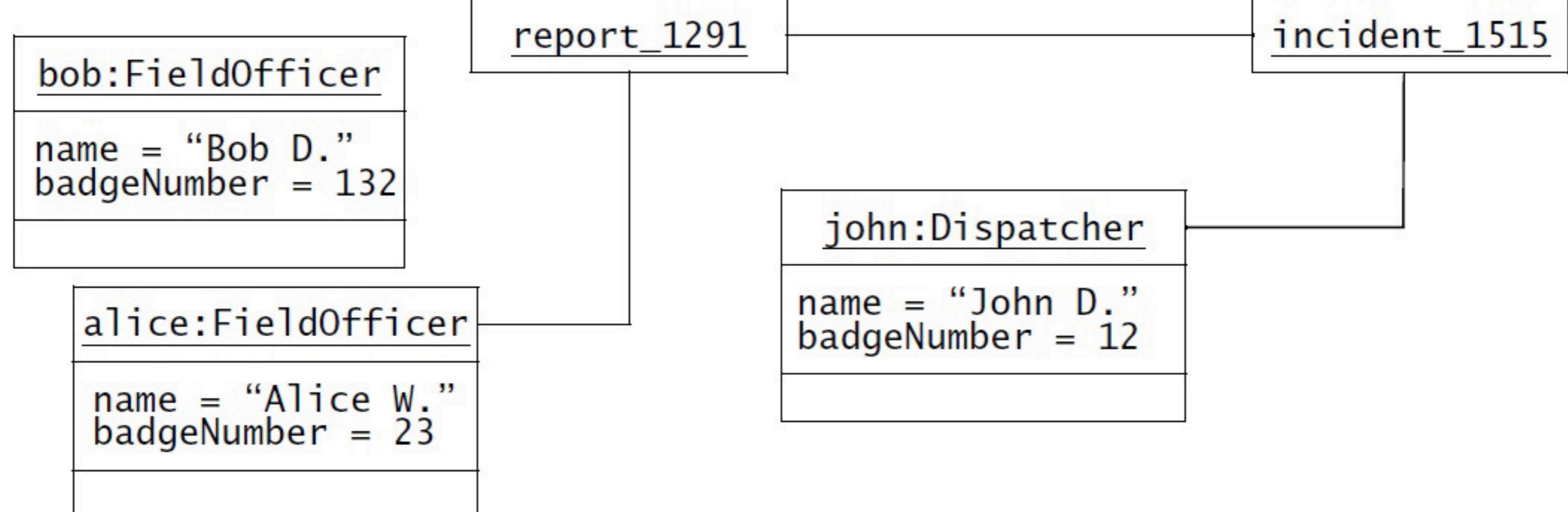
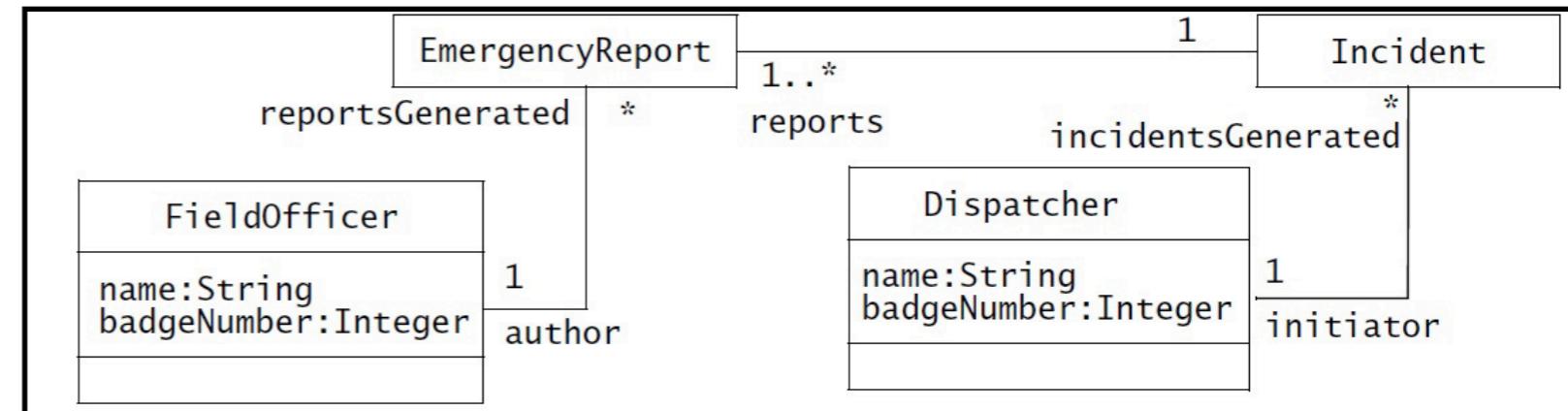


Object diagram



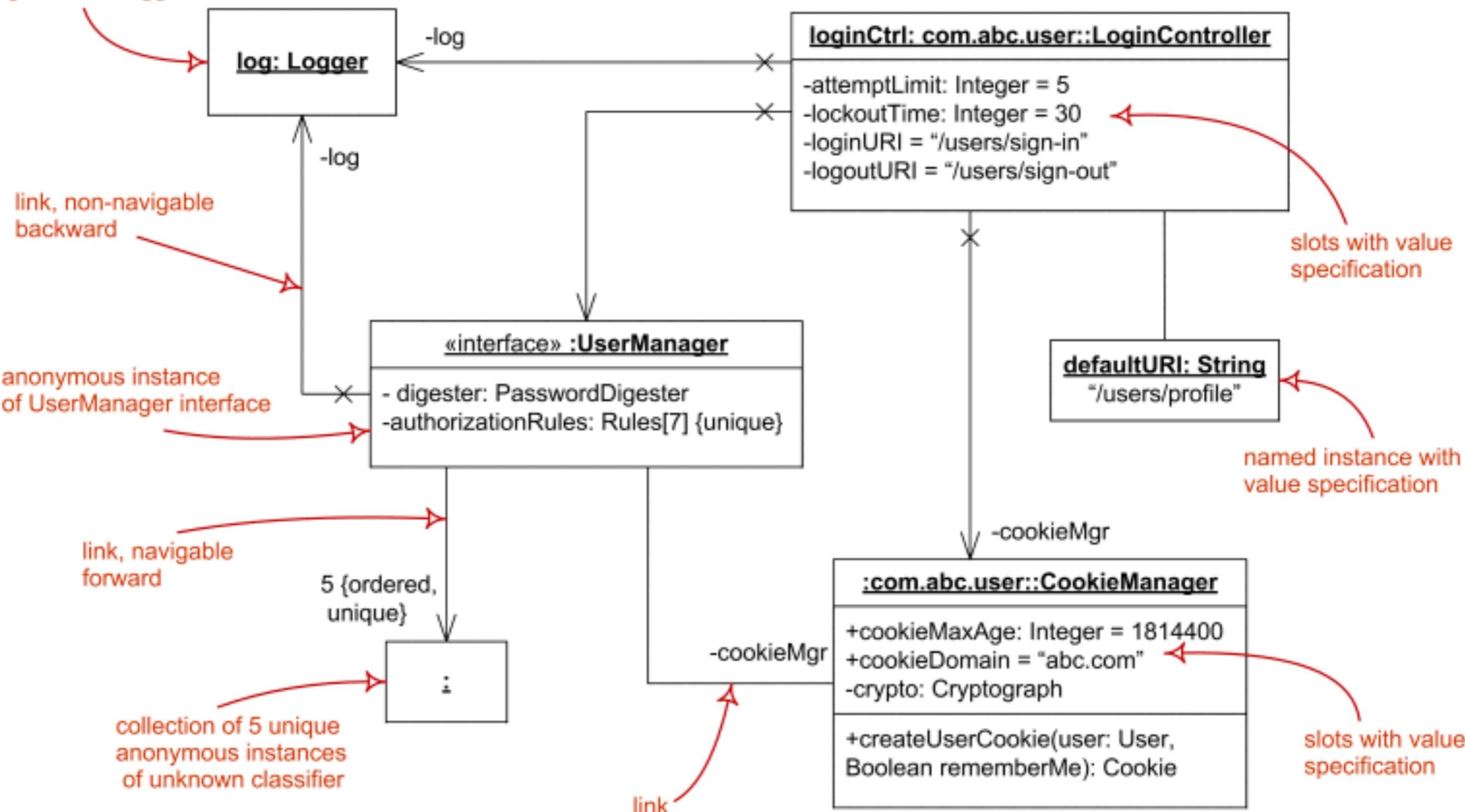
- Structure diagrams.
- Object diagrams describe the relationship between specific objects, i.e., instances of classes at a specific point in time.
- Similar to class diagrams.
 - Instances instead of classes.
 - Links instead of (instances of) associations. We will see more later.
 - Values and types of attributes describing the actual state of an instance.
 - Object names are underlined to indicate that they are instances. This rule applies in general in UML.
- They are a snapshot of a system structure at a given point in time during execution.

Previous class diagram

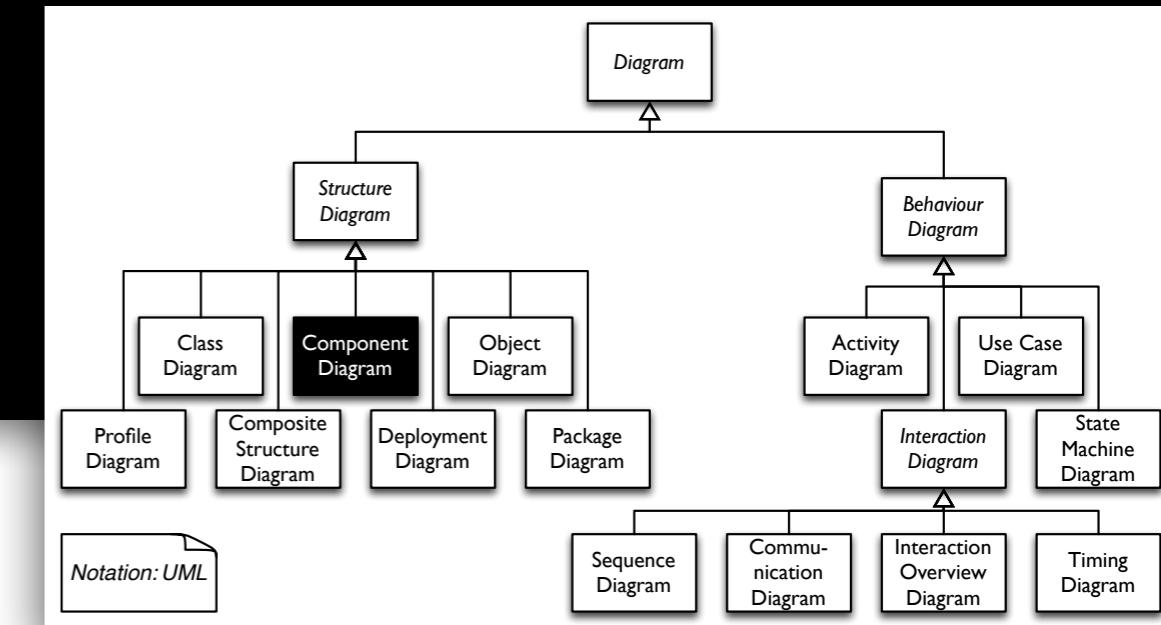


instance specification
log of class Logger

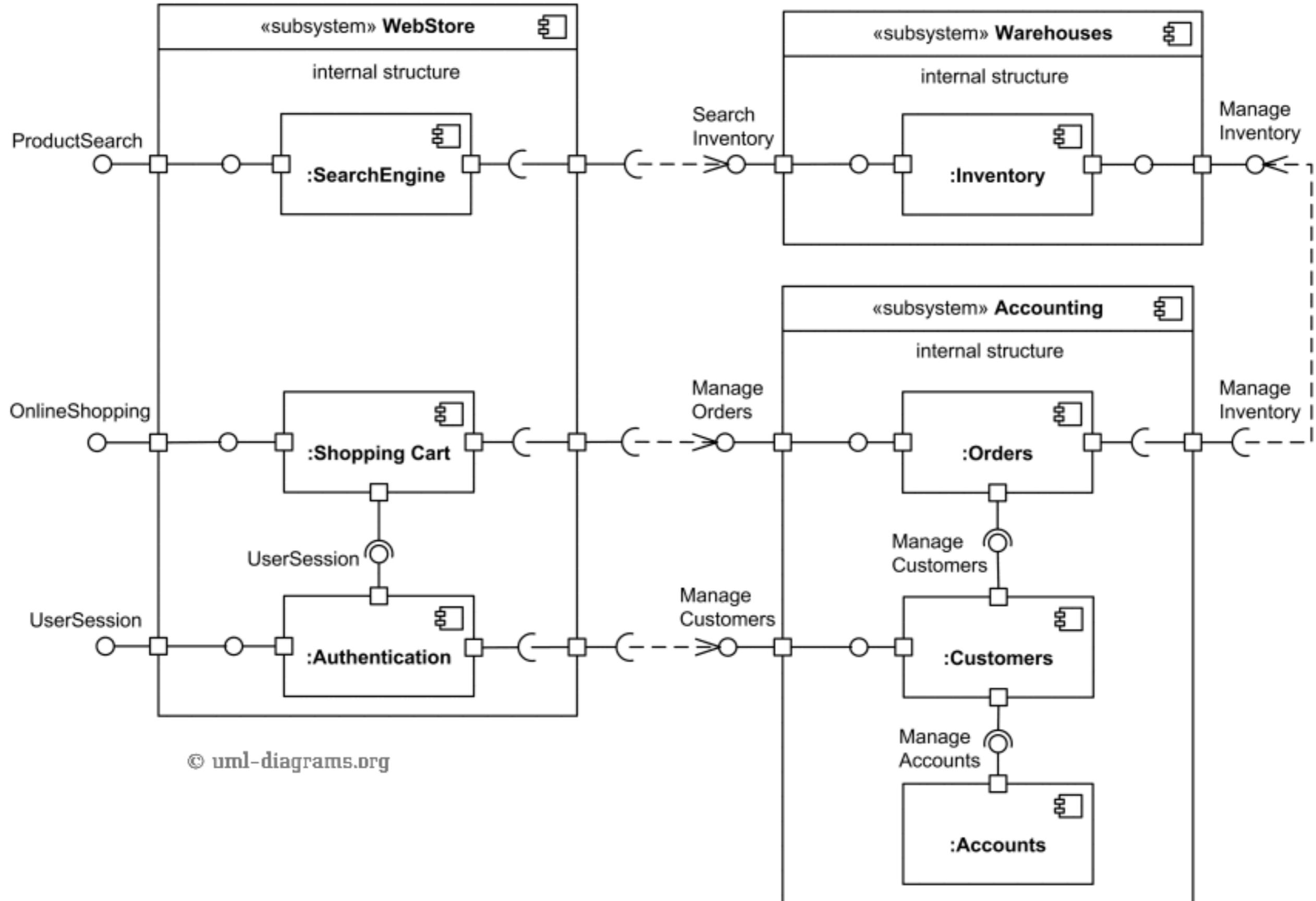
© uml-diagrams.org



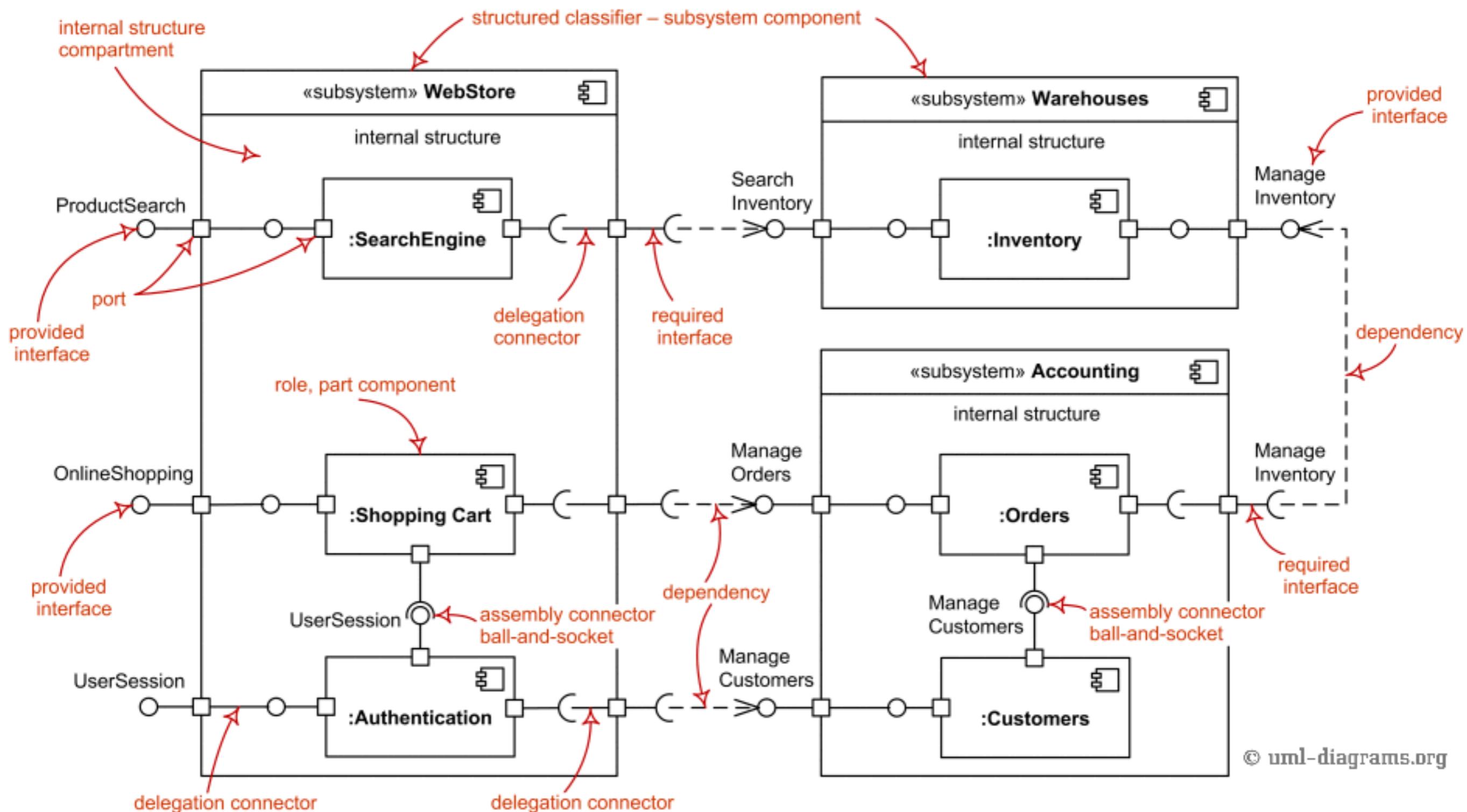
Component diagram



- Structure diagrams.
- Component diagram shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is heavily used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA).
- Components in UML could represent
 - logical components (e.g., business components, process components), and
 - physical components (e.g., CORBA components, EJB components, .NET components, WSDL components, etc.),
 - along with the artifacts that implement them and the nodes on which they are deployed and executed.

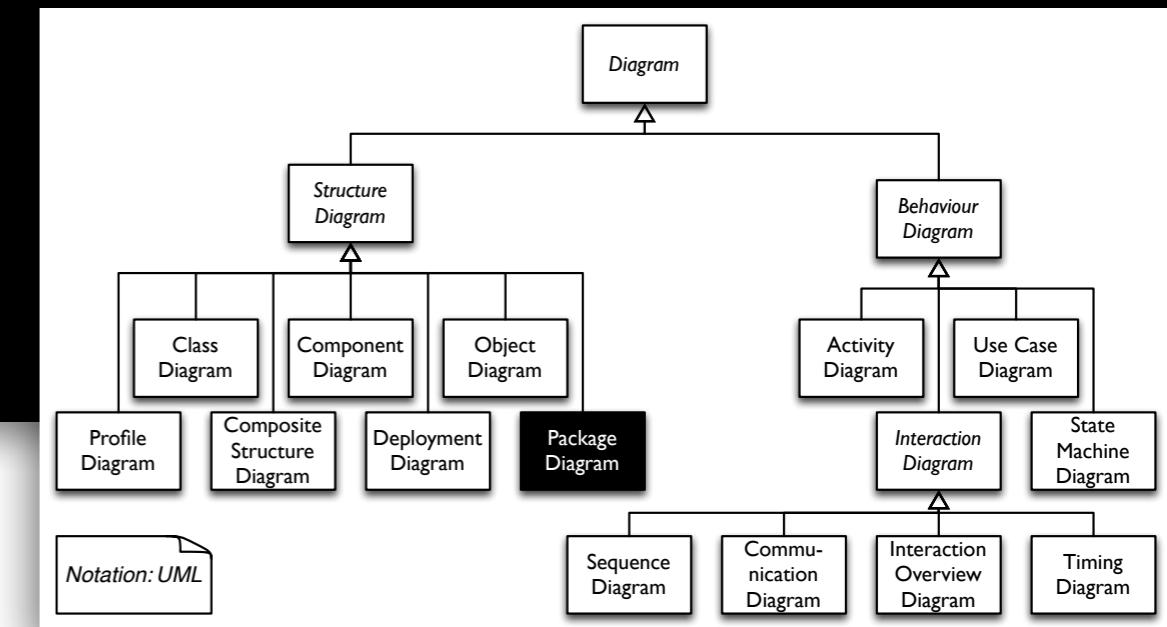


© uml-diagrams.org

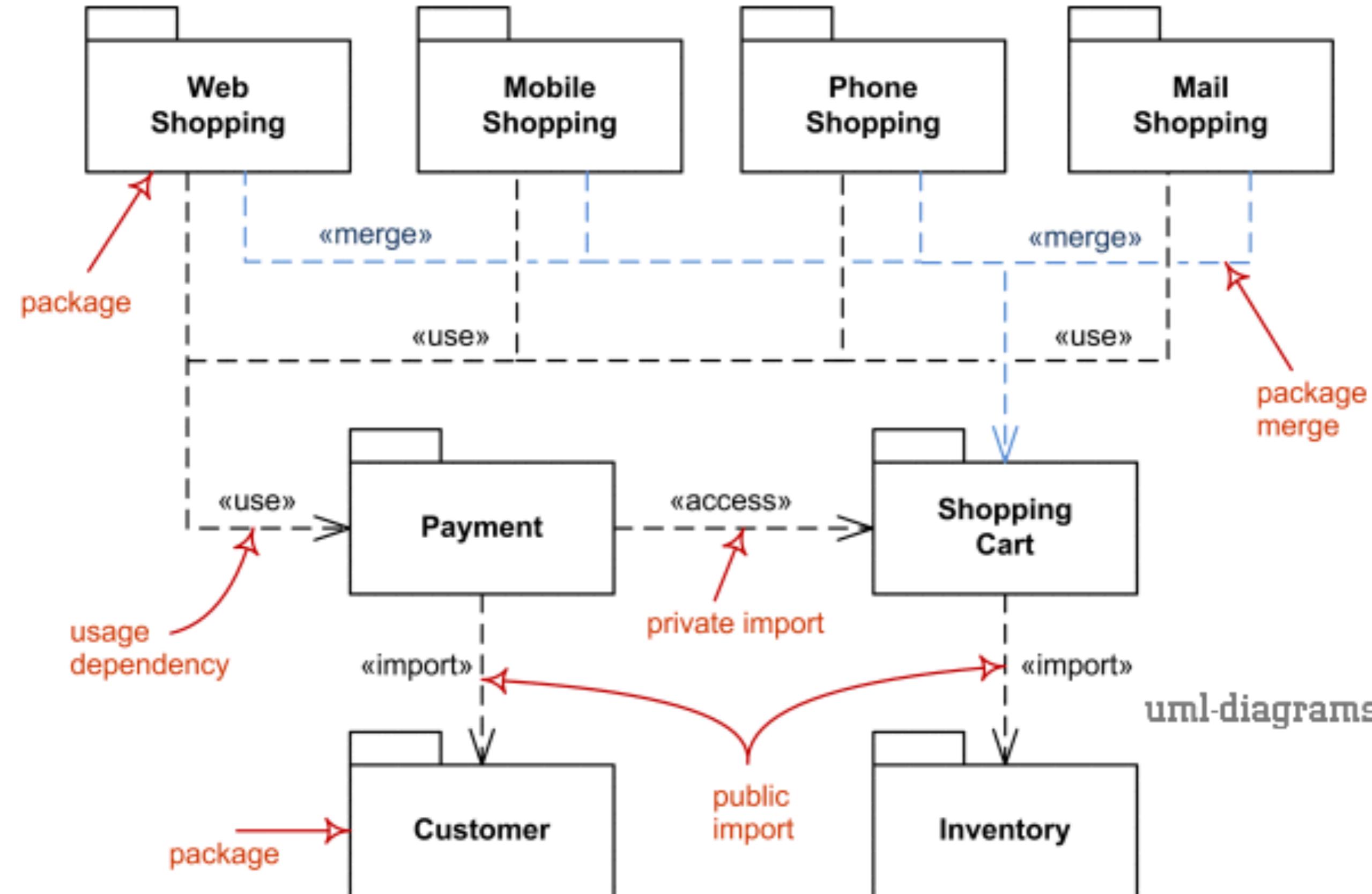


© uml-diagrams.org

Package diagram

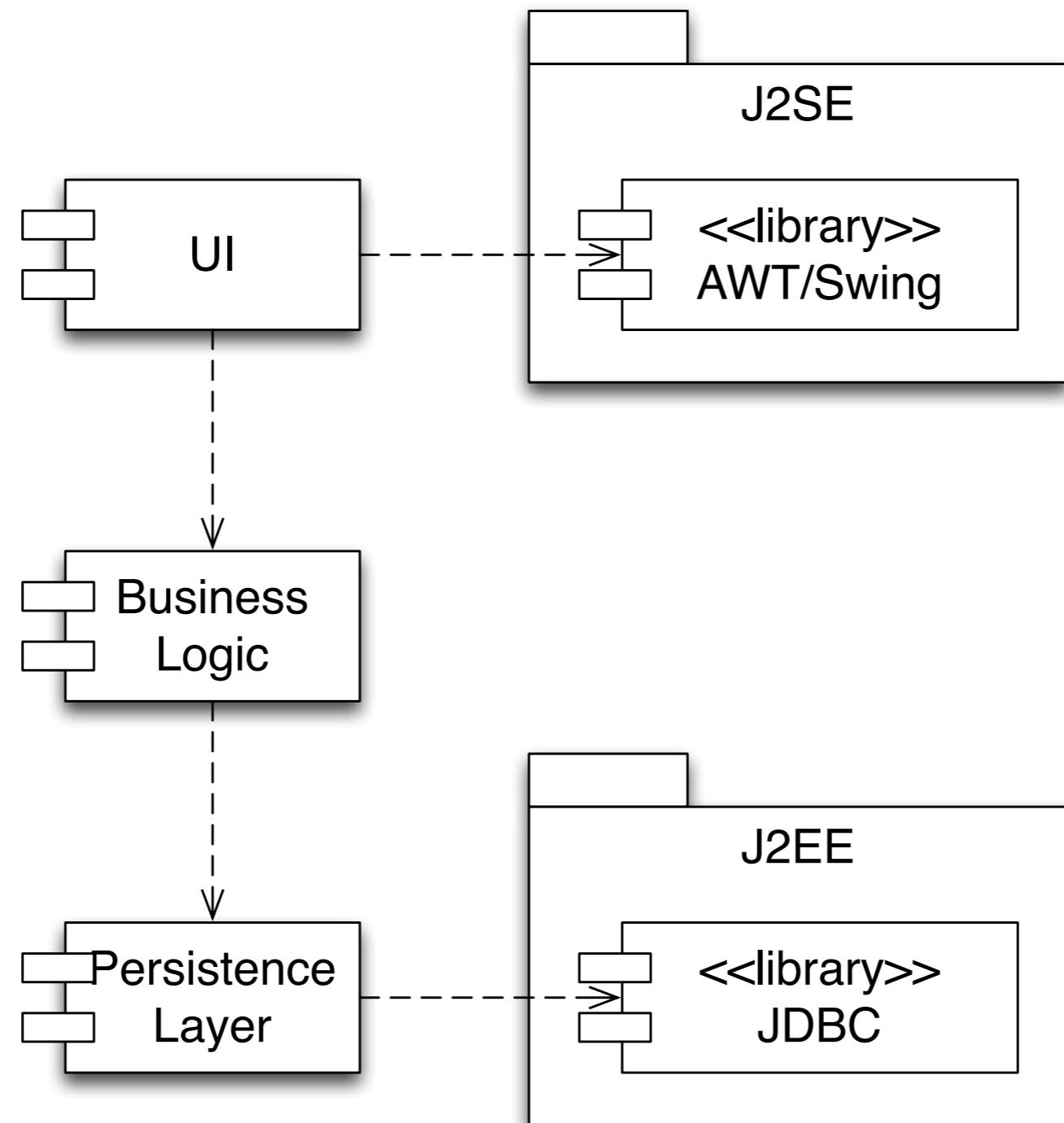


- Structure diagrams.
- Very versatile.
- They are used to depict how various packages of a system are connected.

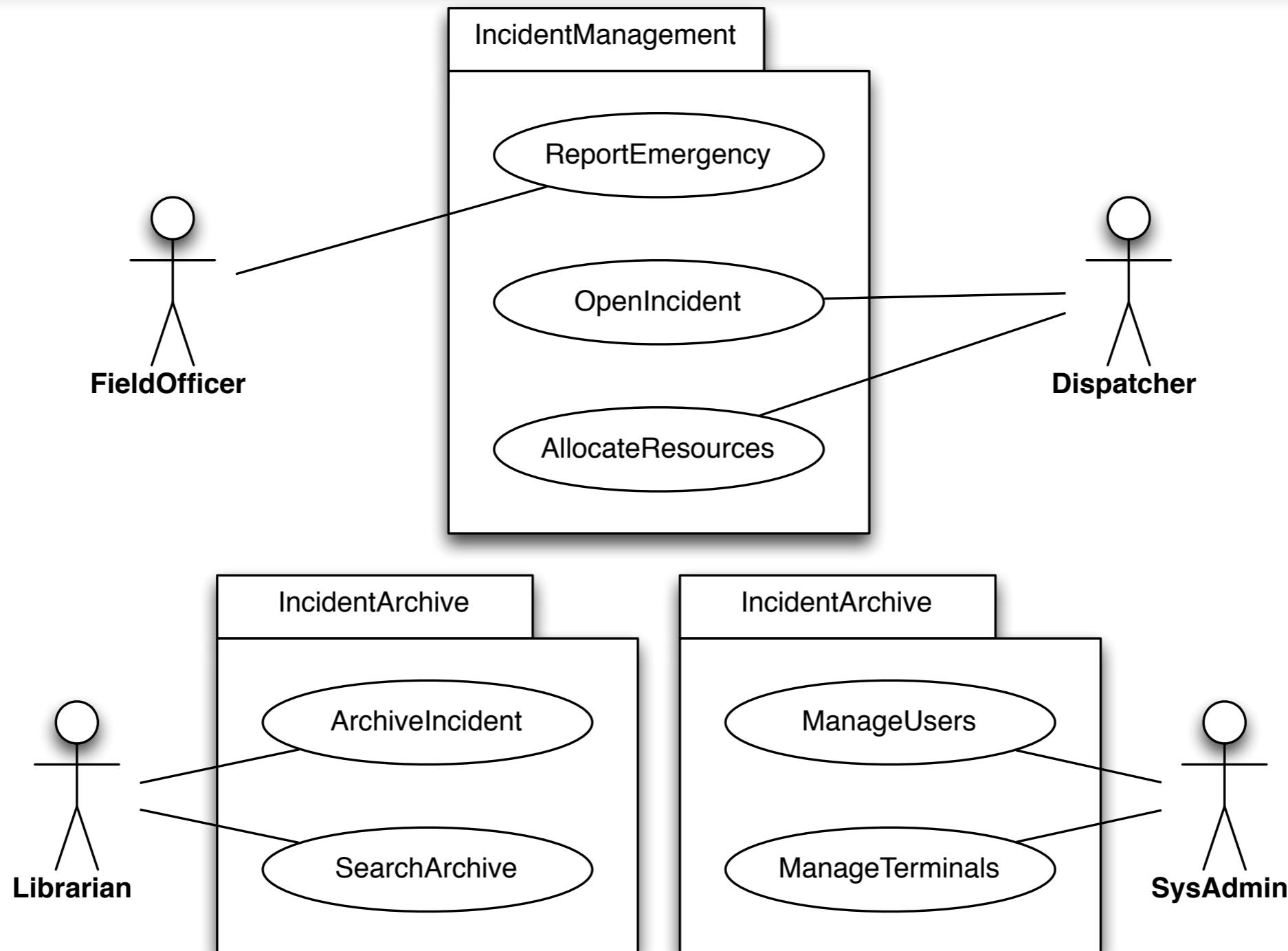


uml-diagrams.org

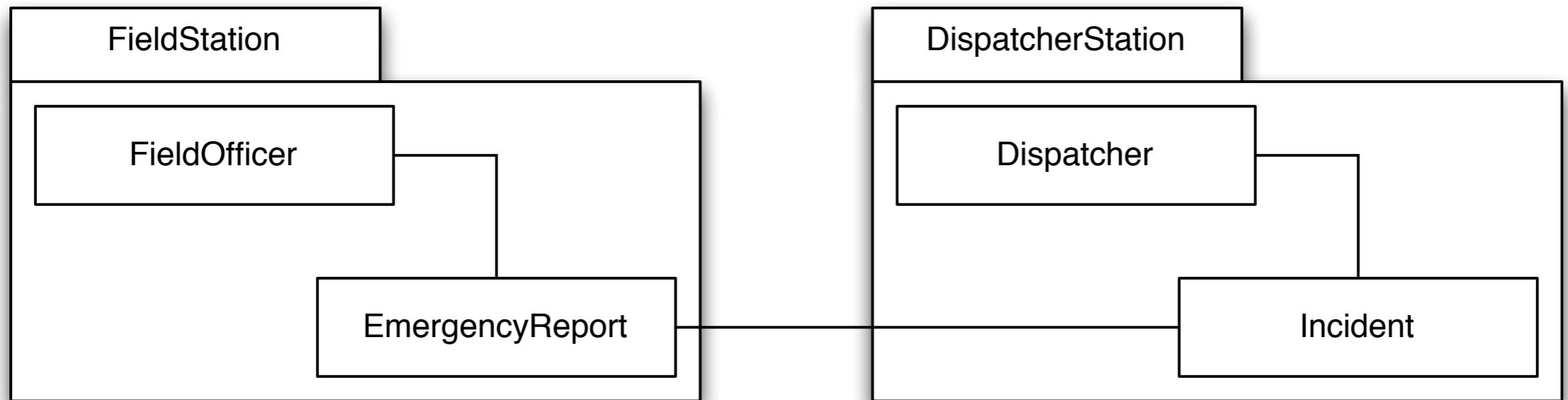
Package and component



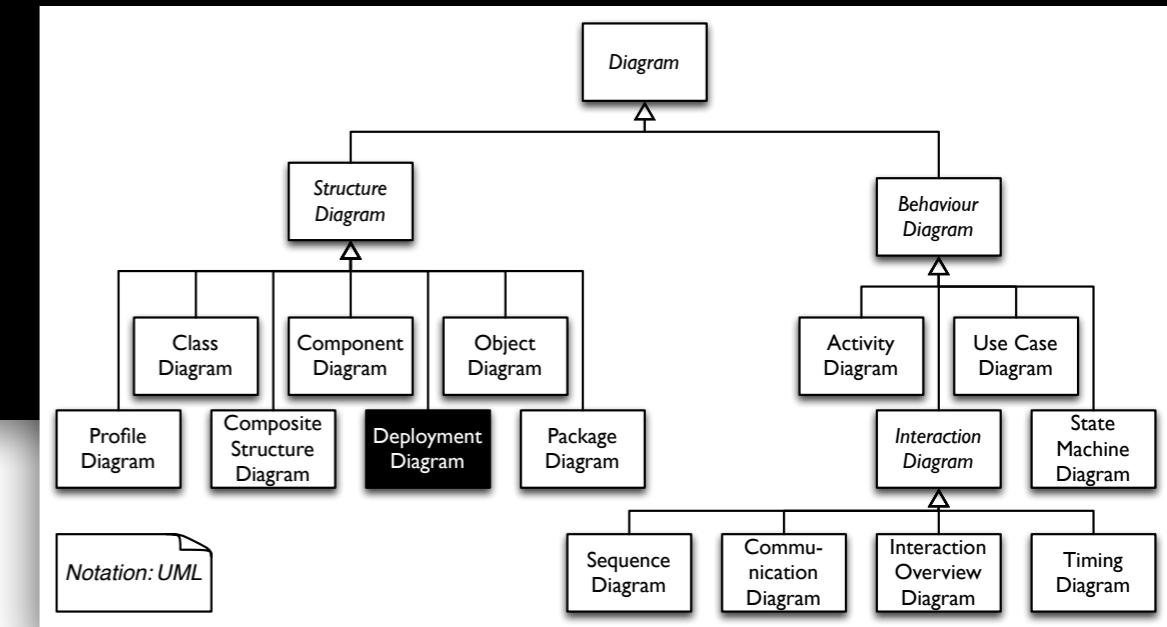
Package and use case



Package and object



Deployment diagram

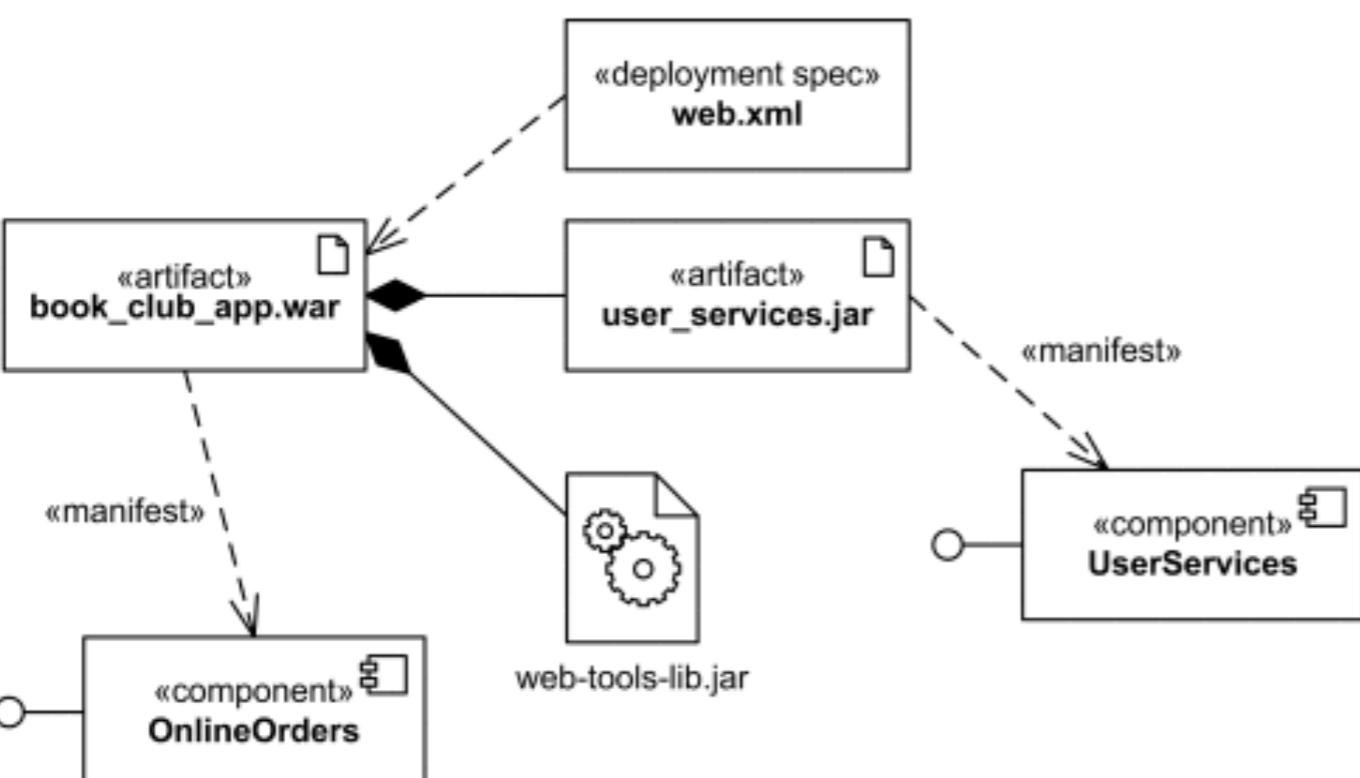


- Structure diagrams.
- They are used to depict where the various elements of a system are located.
- For instance, a distributed system based on a client/server architecture.
- Deployment diagrams contain
 - nodes and communication paths
 - components and dependancies

«device» Application Server

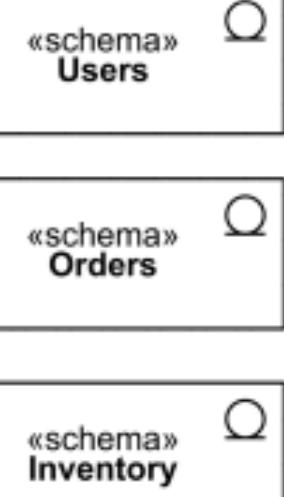
«JSP server» Tomcat 5.5

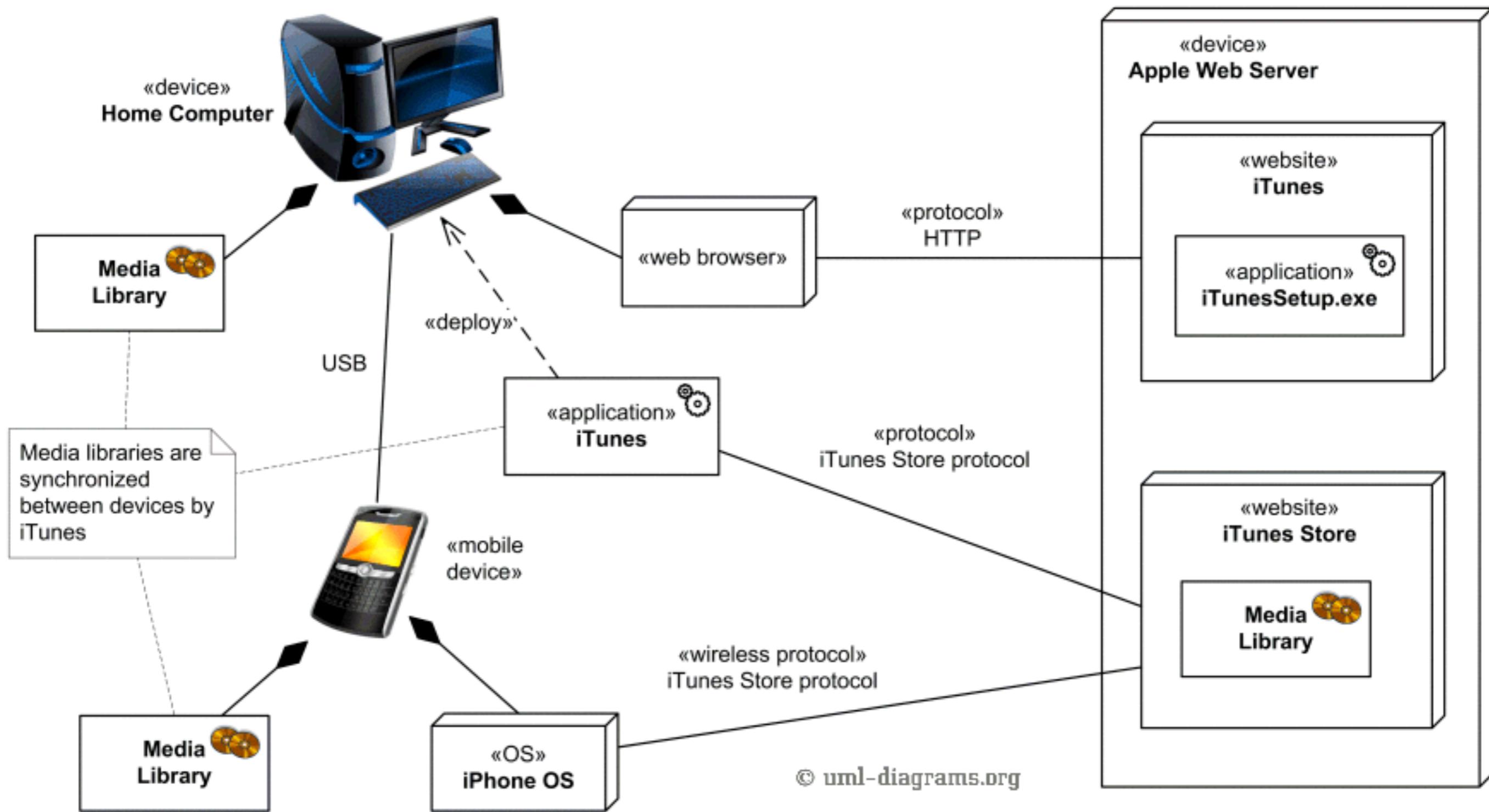
«executionEnvironment»
Catalina Servlet 2.4 / JSP 2.0 Container

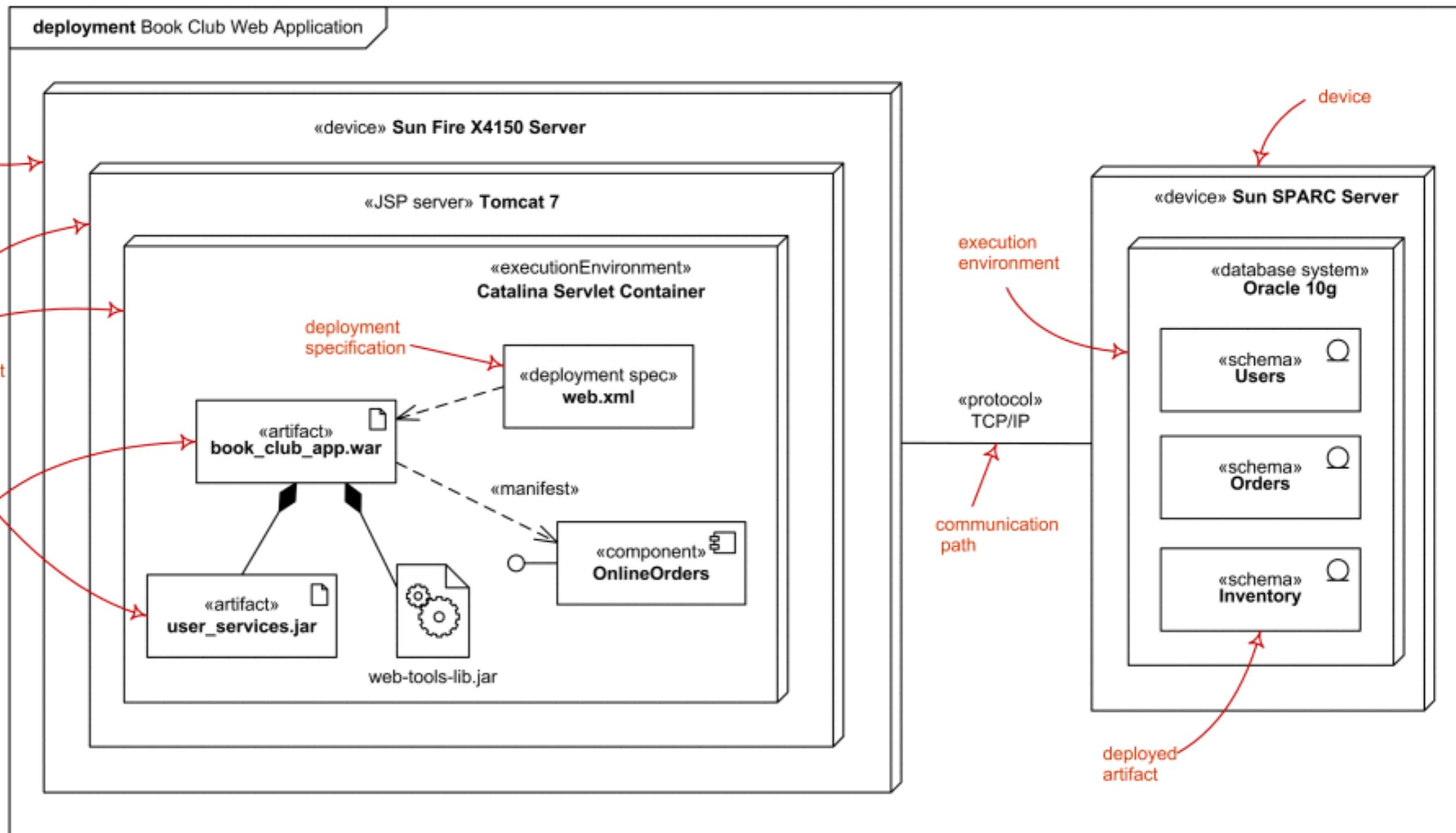


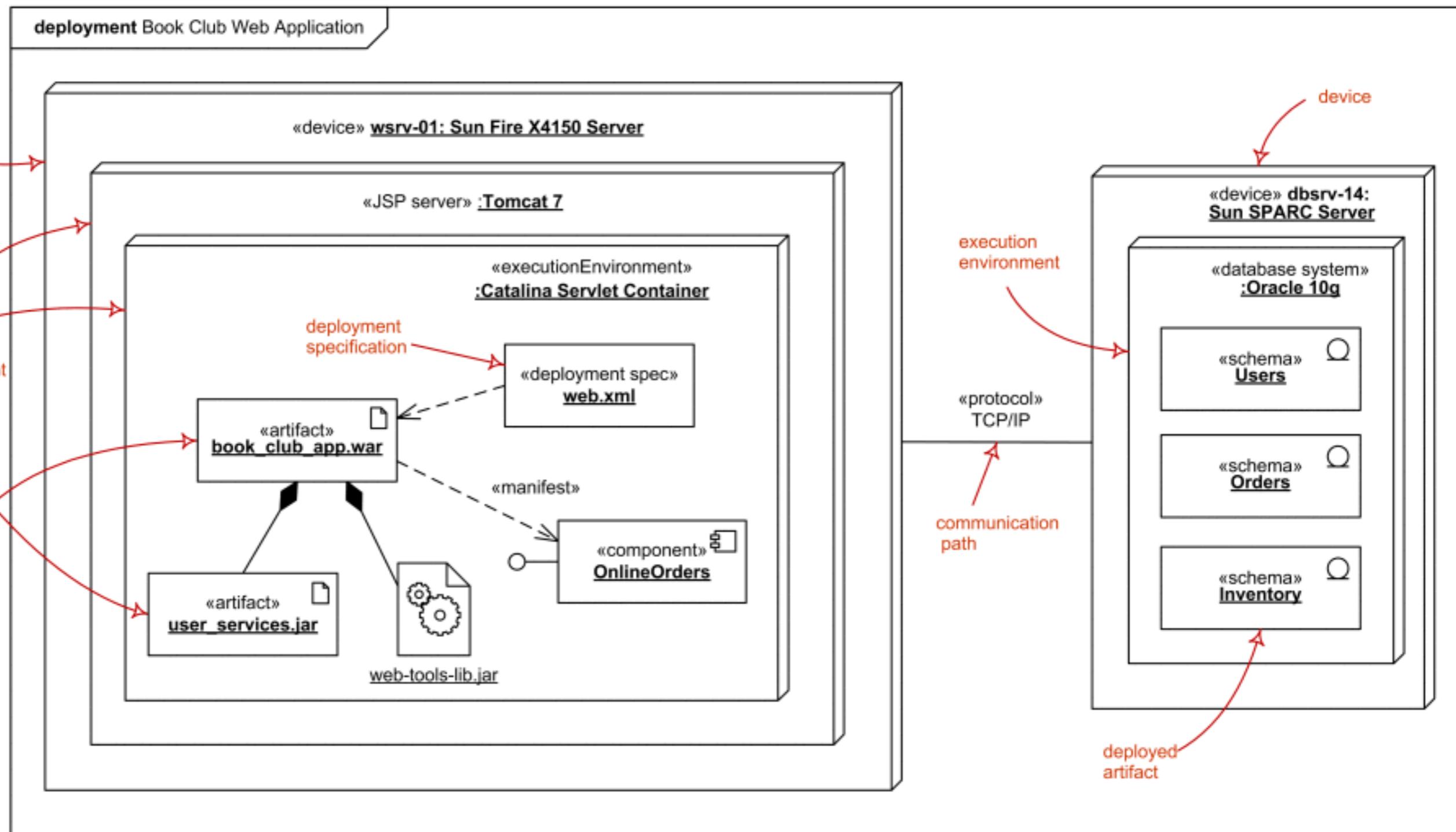
«device» Database Server

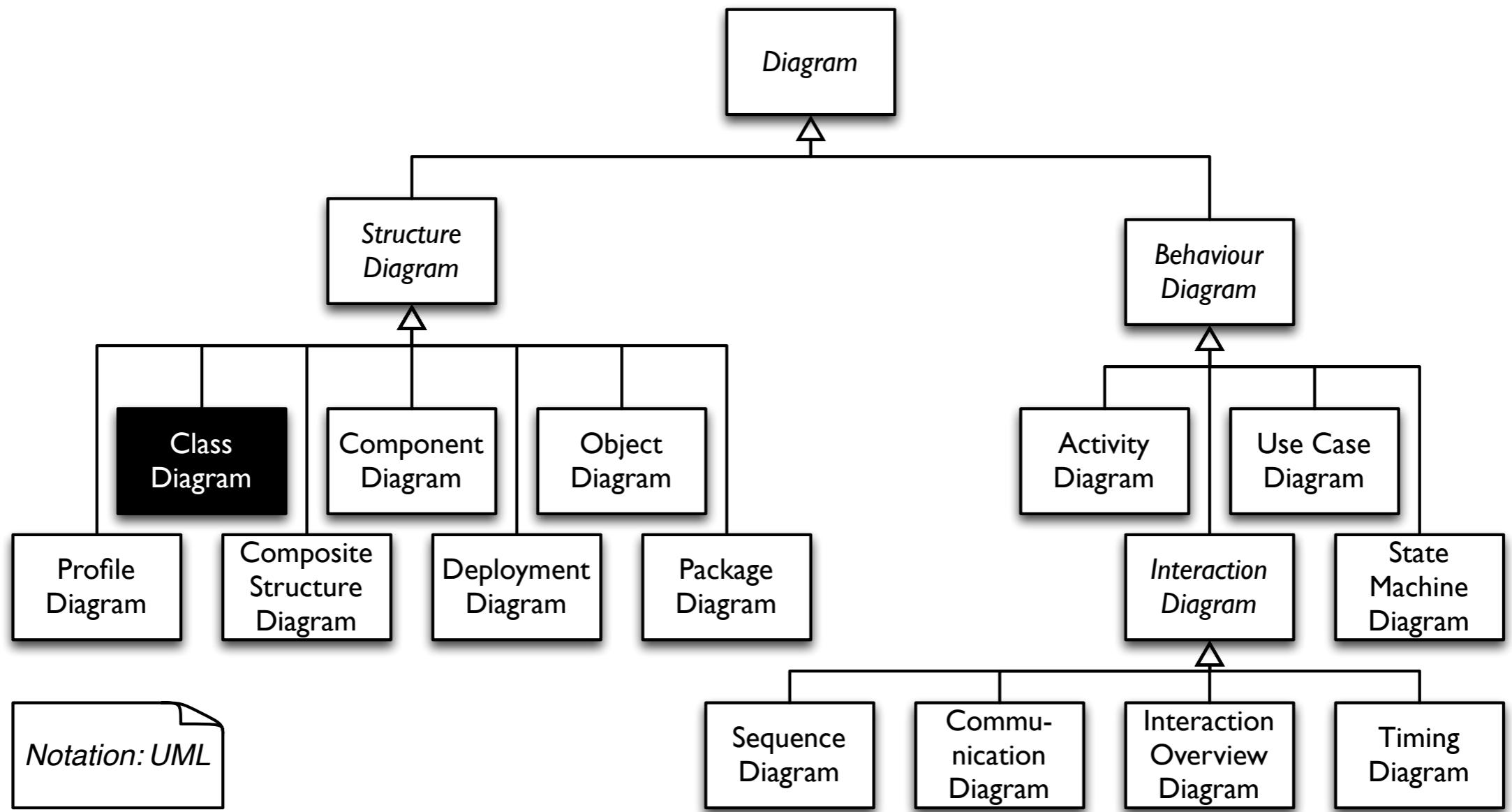
«database system» Oracle 10g











Diving into class diagrams

The class symbol

- Areas
 - Class name
 - Attributes
 - Methods (signatures)
- Abbreviations
 - Without attributes and methods
 - Without attributes or methods
- Visibility
 - Public
 - Private
 - Protected
- Underline when static

ClassName
attribute1
attribute2
method1()
method2()

ProjectManager

ProjectManager
name
telephone

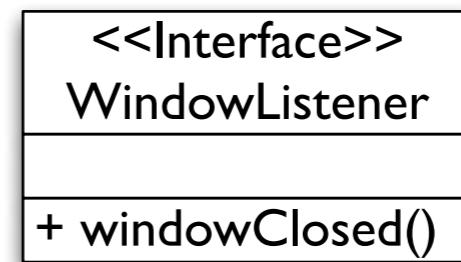
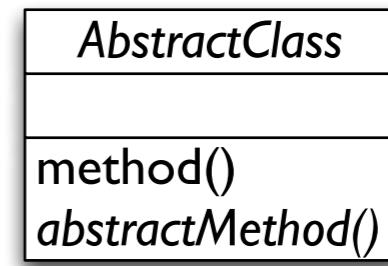
ProjectManager
startProject()
endProject()

ClassName
+ public()
- private()
protected()

StrangeQueue
- <u>instance</u> : StrangeQueue
- Singleton()
+ getInstance() : StrangeQueue

Abstract classes and interfaces

- Abstract classes
 - Italic
 - or using the label {abstract}
 - holds also for methods
- Interfaces
 - Same as the classes
 - Use the stereotype <>Interface<>
 - or the lollipop notation

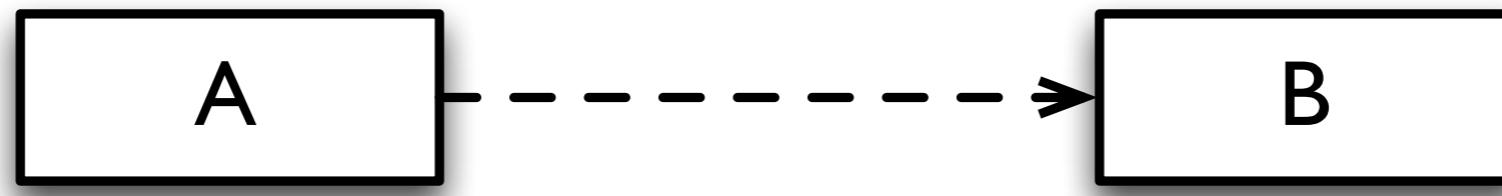


WindowListener

Relationships between classes

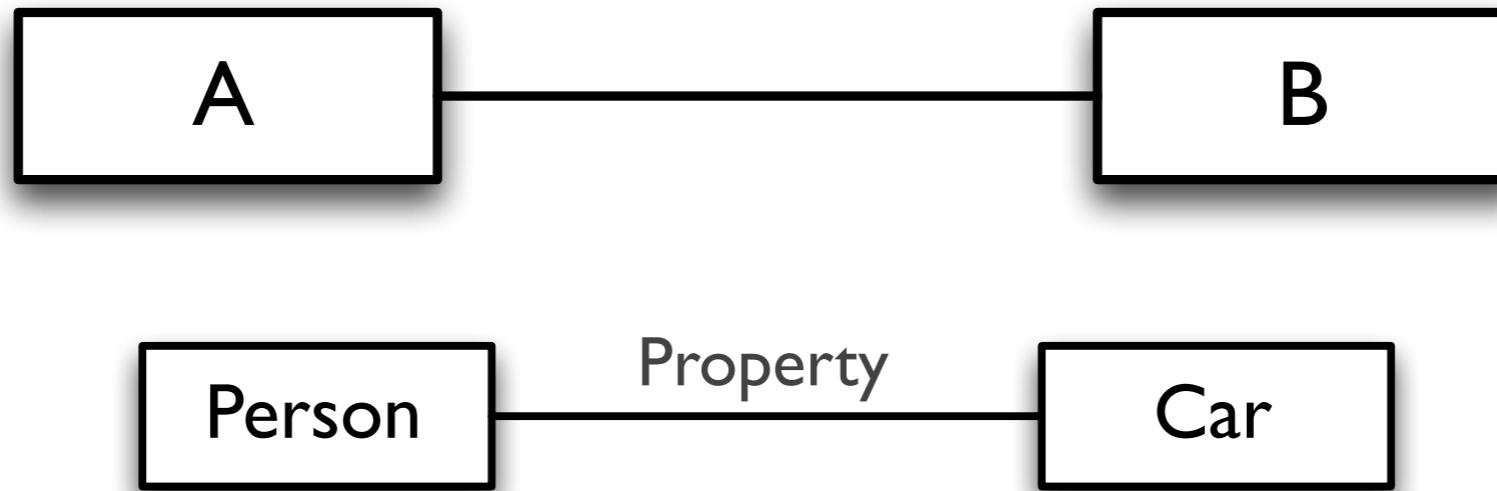
- Relationships between classes (and interfaces and packages):
 - Dependancy
 - Association
 - Aggregation/Composition
 - Inheritance
- Navigability, Multiplicity, Roles, and Qualification.

Relationships: generic dependency



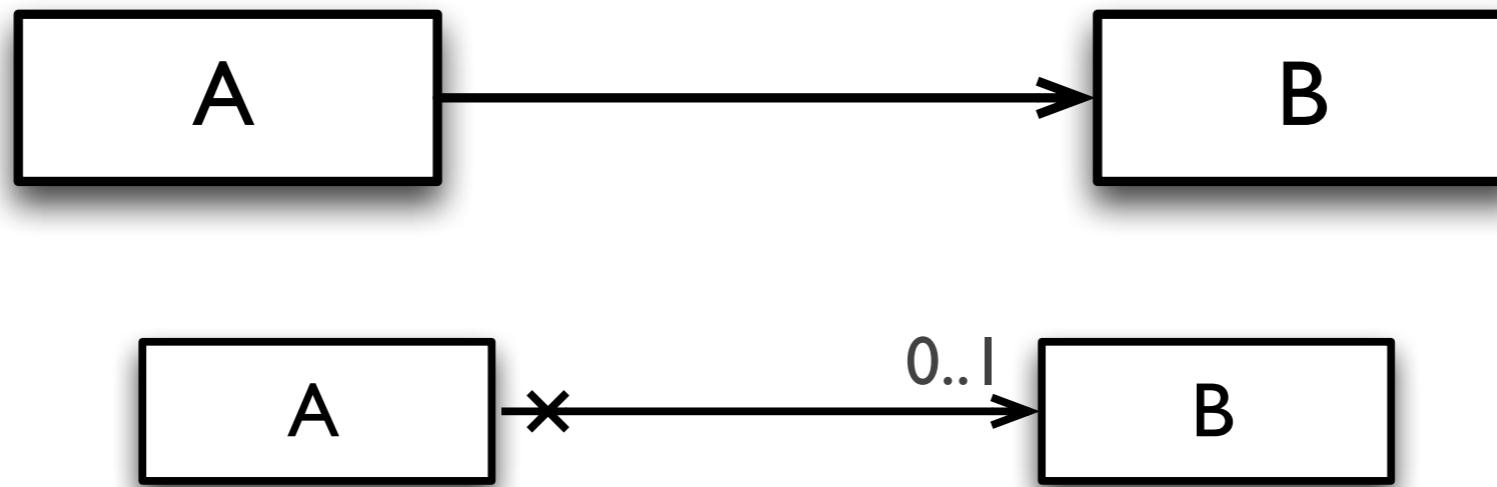
- A depends on B if A needs B to execute.
 - A calls B
 - A uses the type B

Relationships: generic association



- A generic (but interesting) relationship between A and B.

Navigability



- Arrow that indicated if the instance of A has direct access to the instance of B.

Multiplicity and roles

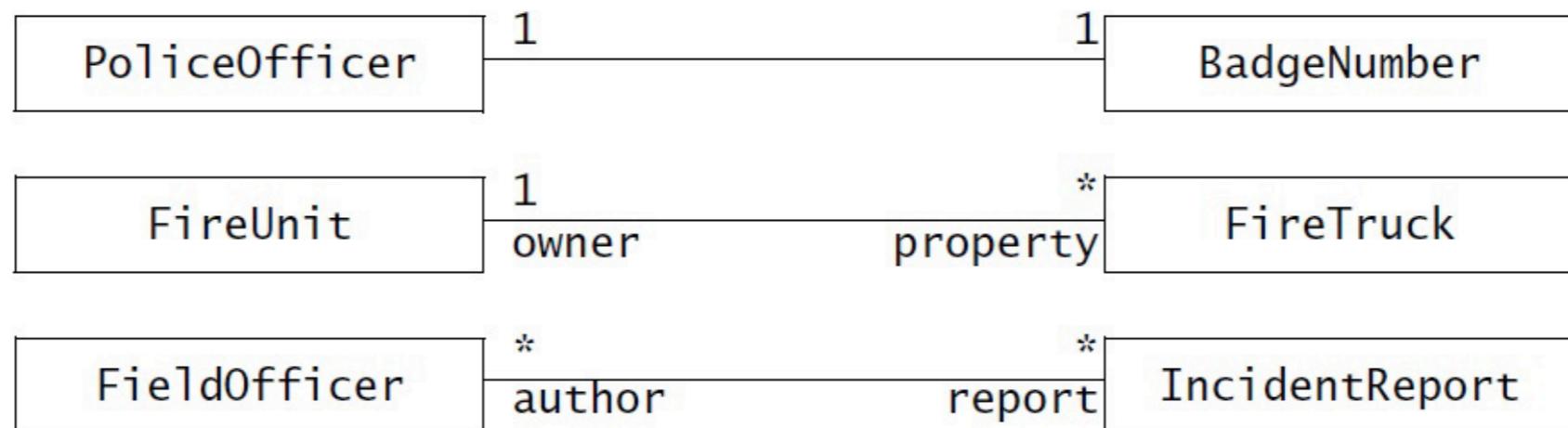
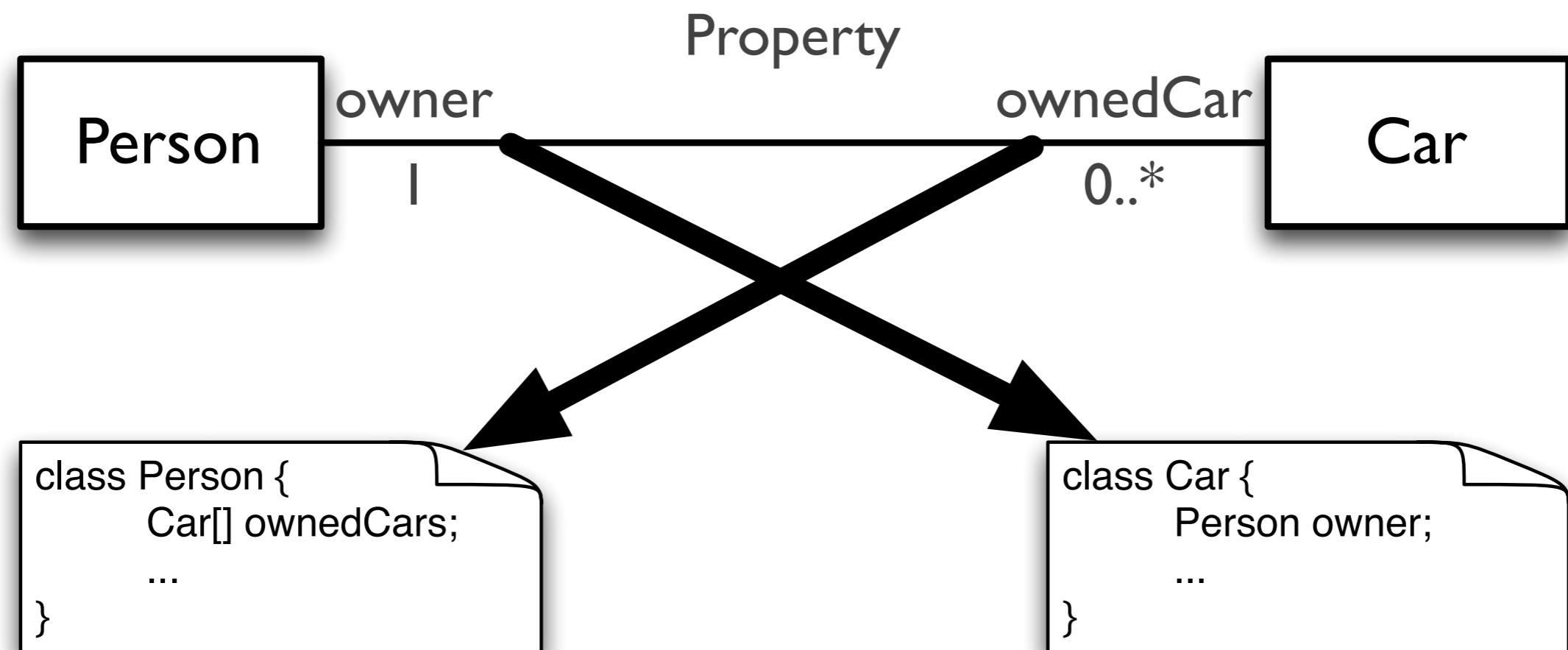


Figure 2-27 Examples of multiplicity (UML class diagram). The association between **PoliceOfficer** and **BadgeNumber** is one-to-one. The association between **FireUnit** and **FireTruck** is one-to-many. The association between **FieldOfficer** and **IncidentReport** is many-to-many.

Copyright © 2011 Pearson Education, Inc. publishing as Prentice Hall

- Indicates how many instances for each relationship are allowed.

An example



Qualification

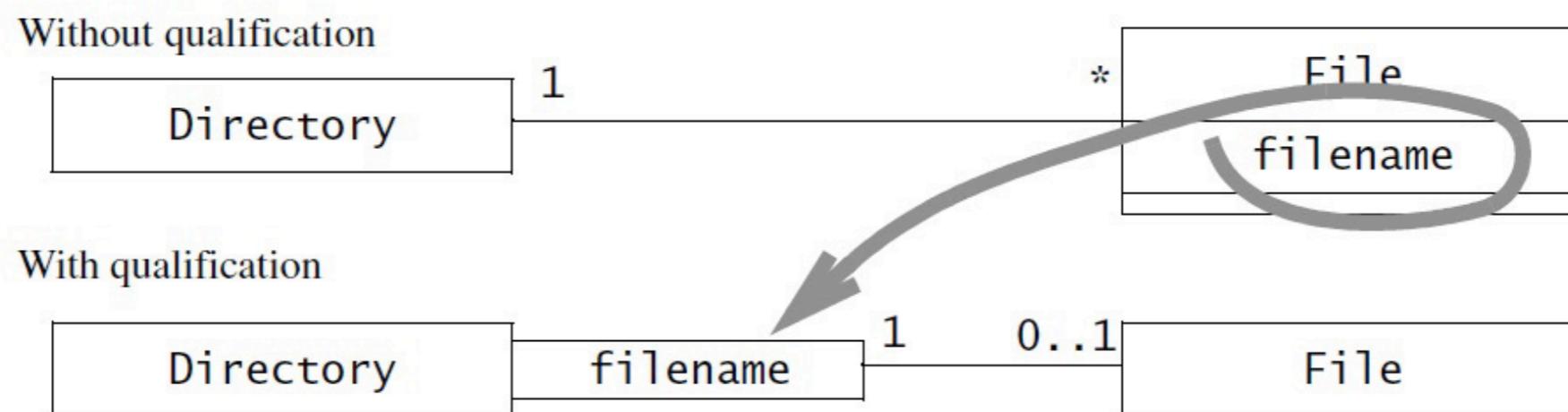
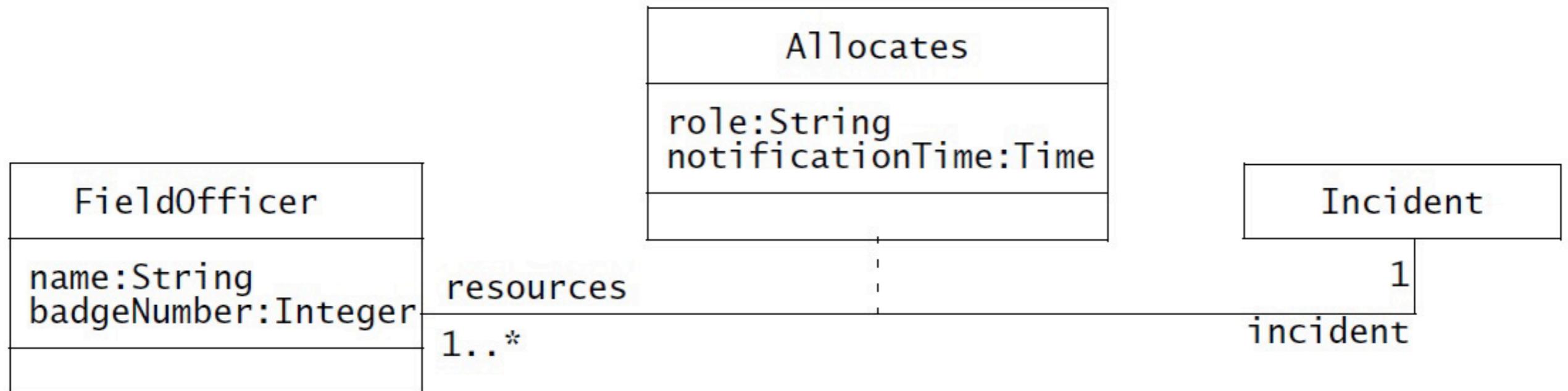


Figure 2-31 Example of how a qualified association reduces multiplicity (UML class diagram). Adding a qualifier clarifies the class diagram and increases the conveyed information. In this case, the model including the qualification denotes that the name of a file is unique within a directory.

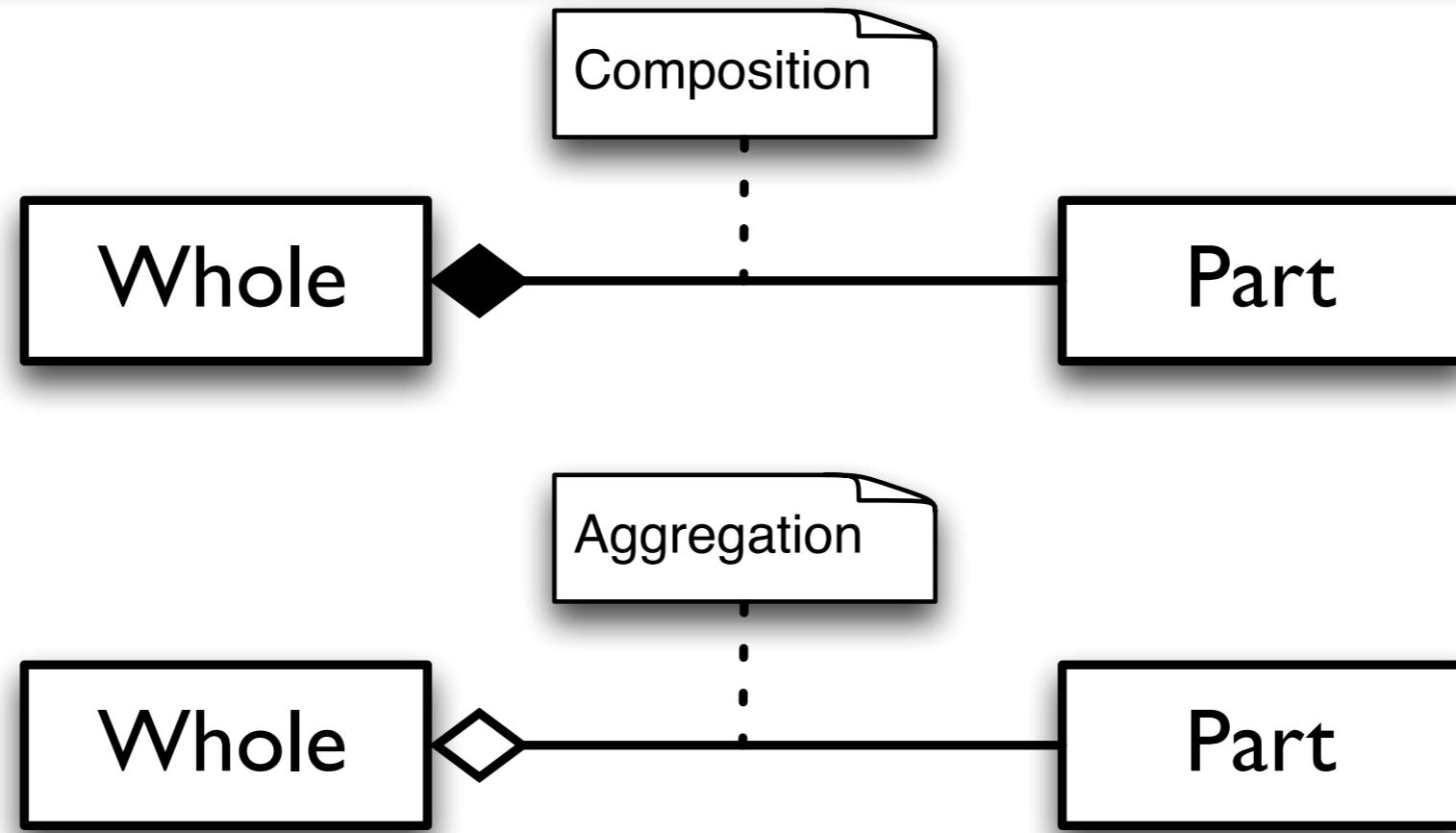
Copyright © 2011 Pearson Education, Inc. publishing as Prentice Hall

Relationships: association class



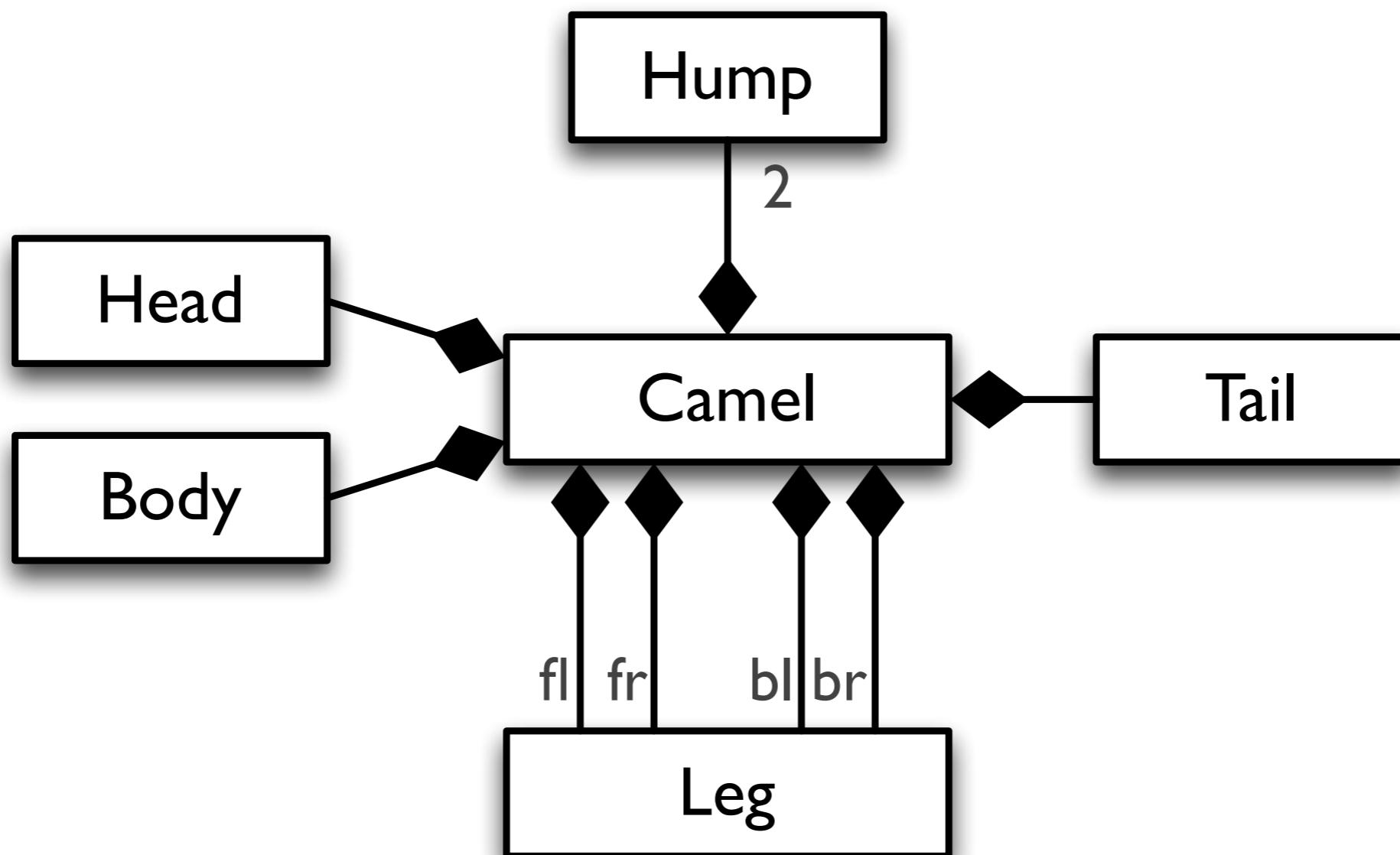
- An association may be refined to have its own set of features; that is, features that do not belong to any of the connected classifiers but rather to the association itself.

Relationships: aggregation/composition



- They represent a whole/part relationship between classes.
- Composition is stronger than aggregation. In the composition relationship the whole cannot exist without the parts and “usually” when the composite (whole) is deleted, so are the components (parts).

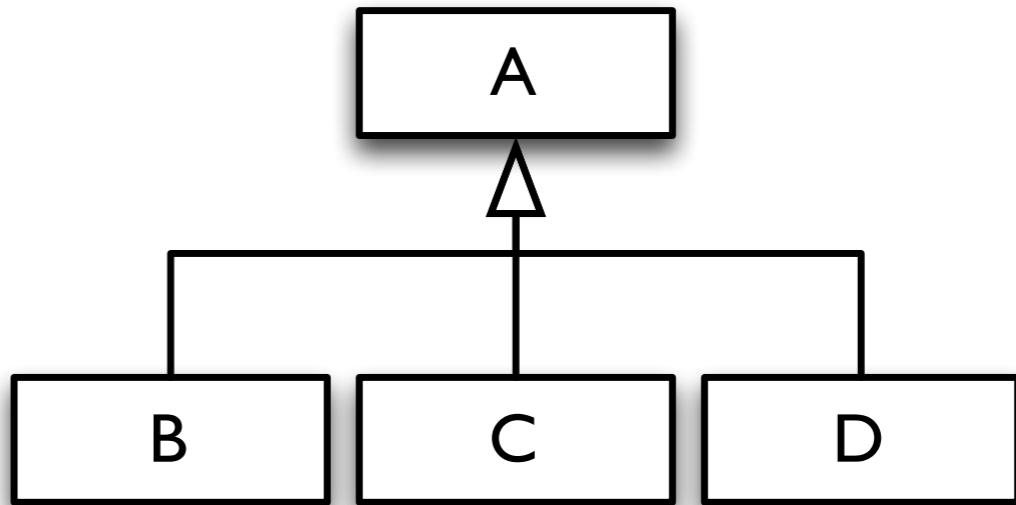
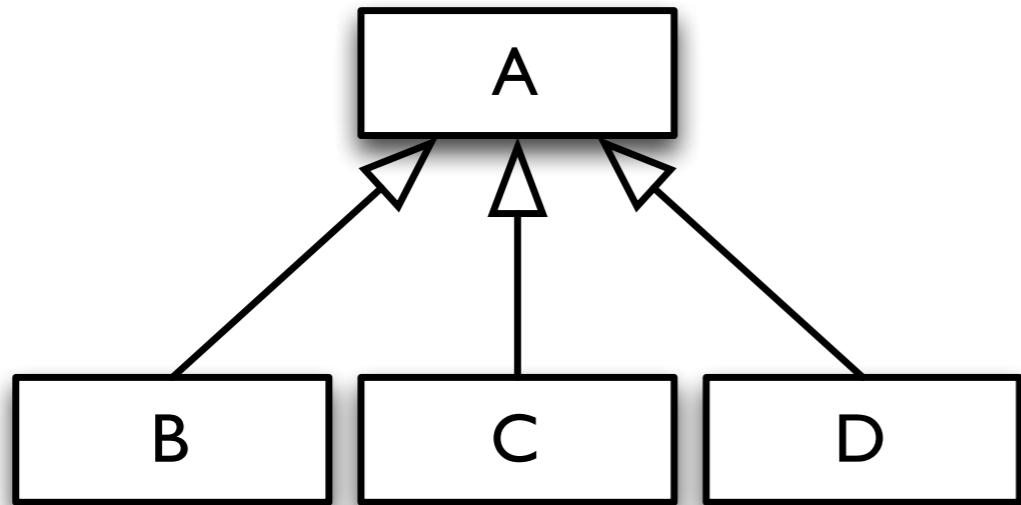
An example of composition



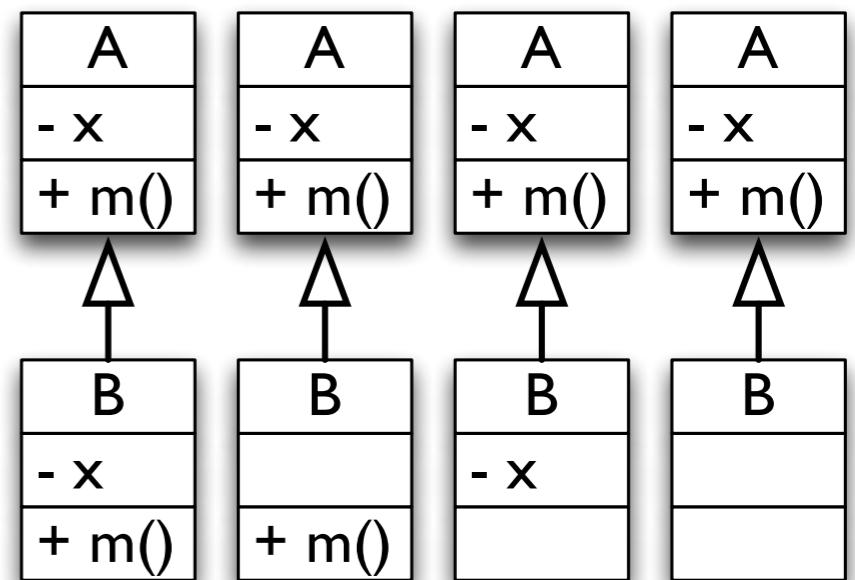
Note!

- It is not always easy to define whether a relationship is a composition or an aggregation ...
- ... it is important to know that a distinction exists,
- ... it is important to know the implications of a composition over an aggregation,
- ... however, in practice do not spend too much time on it!

Relationships: inheritance

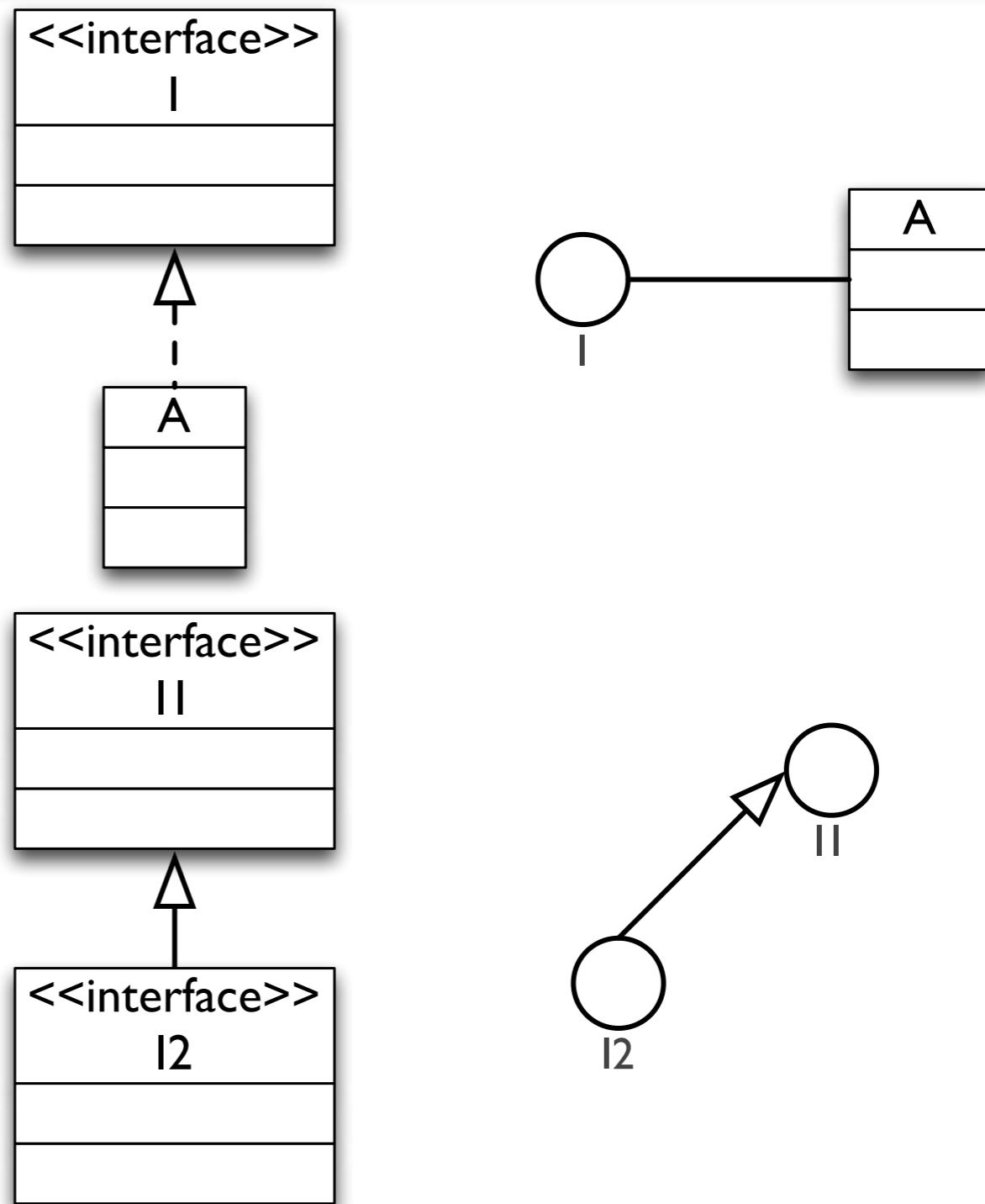


- B, C, D are a subclass of A.
 - B, C, D are a specialization of the superclass A.
 - B, C, D inherit from A.
- Attributes and methods inherited can be either shown or not (is it a relevant detail?).

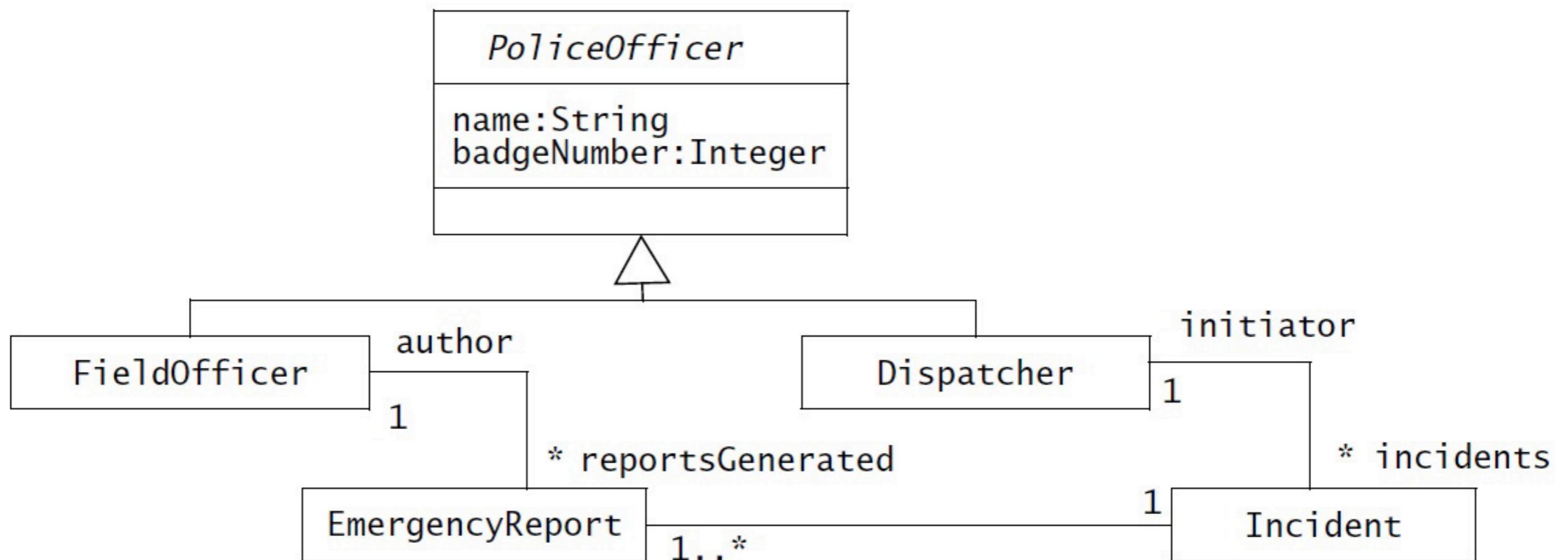


Relationships: inheritance with interfaces

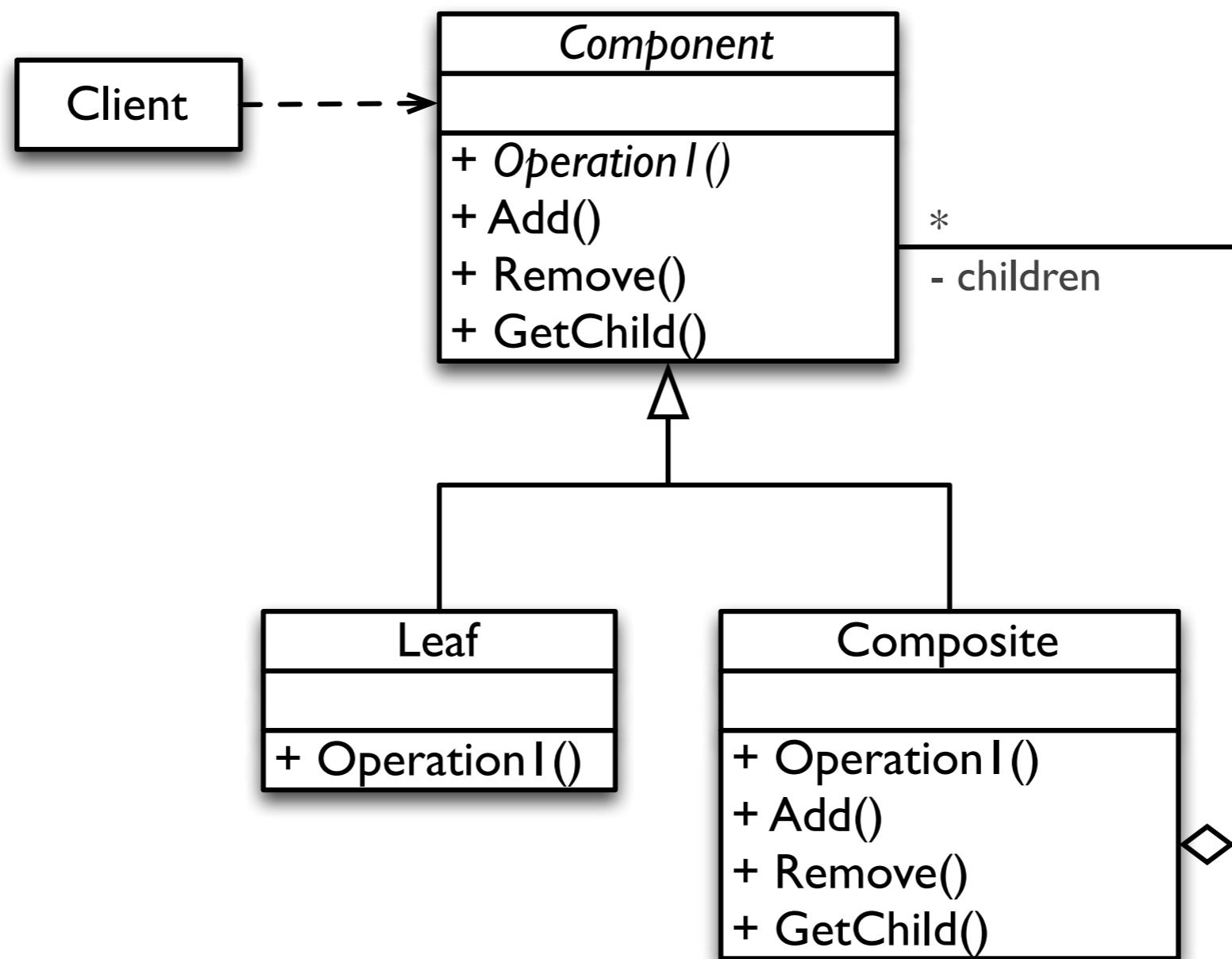
- A implements I
- II extends I2



An example



Another example



Concluding

What else is left about UML?

- Additional details about ALL the diagrams.
- Extensions.
- Object Constraint Language (OCL).
- Best practices.
- ... but when necessary, we will cover additional topics as the necessity arises.

Pareto principle - 80/20 rule

- [http://en.wikipedia.org/wiki/Pareto_principle]
 - 80% of your profits come from 20% of your customers
 - 80% of your complaints come from 20% of your customers
 - 80% of your profits come from 20% of the time you spend
 - 80% of your sales come from 20% of your products
- With UML is the same!
 - You can model 80% of most problems by using about 20% of the UML concepts.

Key Points I

- System, modeling, model, view, notation.
- UML emerged victorious from the war of notations. An aggregation of several notation by the 3 amigos: Rumbaugh, Jacobson, Booch.
- 3 views on UML: as sketch, as blueprint, as a programming language.
- In this course you need to be able to draw syntactically correct UML diagrams – important part of the exam.
- Structure diagrams versus behaviour diagrams.

Outline

- Literature
 - [OOSE] ch. 2
- Topics covered:
 - Modeling
 - UML and its history
 - UML diagrams
 - Brief description
 - Example(s)
 - Comment (or another example)
 - Template from [\[uml-diagrams.org\]](http://uml-diagrams.org)
 - A deeper dive into class diagrams

