COMP2009
Software Engineering

An Overview of The Unified Modelling Language

The Unified Modeling Language (UML)

- UML is a language for:
  - visualising
  - specifying
  - constructing
  - documenting
  - communicating

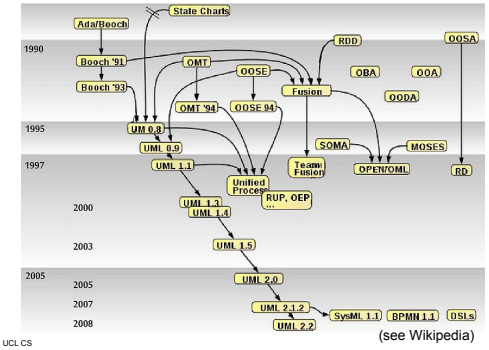
  … the artefacts of a software-intensive system.
- Adopted as a standard by the Object Management Group (OMG)
  - International consortium of over 600 members (IBM, Sun, Microsoft etc.).
  - Founded in 1989 to promote the theory and practice of OO technology.
    - Guidelines and specifications (CORBA, CCM, UML, MDA).
    - Key goals: Reusability, portability and interoperability of object-based software in distributed, heterogeneous environments.

Why UML?

- Supports diverse applications areas.
- Is based on experience and needs of the user community.
- Supported by many methods and tools.
- Core skill for software engineers.

De facto standard for software design modelling

UML History & Status

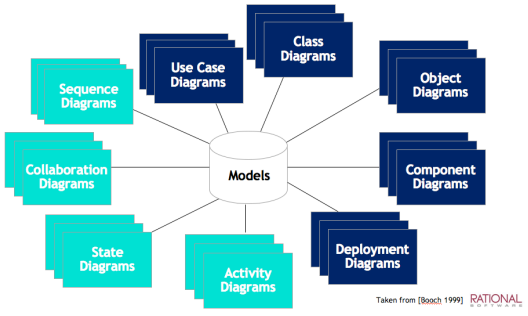


(see Wikipedia)

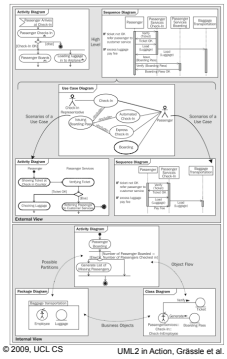
The Three Amigos - Original developers of UML

- Grady Booch
  - The Booch Method (clouds)
  - Better for Design
- James Rumbaugh
  - OMT - Object Modelling Technique
    - Core UML notation derived from OMT
  - Better for Analysis
- Ivar Jacobson
  - OOSE - Object-Oriented Software Engineering
  - Use case approach
  - Boundary, entity and control objects

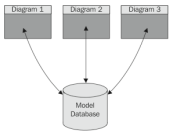
UML Models, Views and Diagrams



Views and diagrams



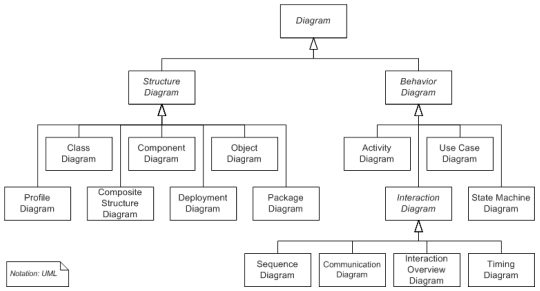
- During modelling many views comprising one or more diagrams will be created.
- They should all be consistent with the model in the model repository.
- A UML CASE Tool will enforce consistency and correct construction of diagrams.
  - CASE == Computer Aided Software Engineering



UML Diagrams

- A diagram is a view into a model.
  - Presented from the aspect of a particular stakeholder.
  - Provides a partial representation of the system.
  - Depicts modelling elements and their relationships.
  - Is (or should be) semantically consistent with other views.
- In UML 1.4, there are nine standard diagrams.
  - Static views: use case, class, object, component, deployment.
  - Dynamic views: sequence, collaboration, state machine, activity.
- UML 2.0 adds four more
  - Static views: composite structure, package.
  - Dynamic views: interaction overview, timing.
  - … and now calls collaboration diagrams “communication diagrams”.
  - See UML 2.0 specification for details.

Diagram Overview



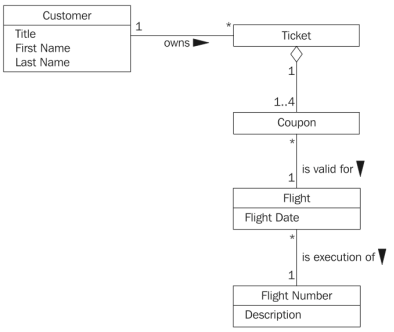
from Wikipedia

UML Modelling Elements

- Represent entities of interest in a model.
  - Structural elements.
    - Class, interface, collaboration, use case, active class, component, node.
  - Behavioural elements.
    - Interaction, state machine.
  - Grouping elements.
    - Package, subsystem.
  - Extension mechanisms.
    - Stereotypes, tagged values, profiles.
  - Other elements.
    - Notes.

Adopted from [Booch 1999] RATIONAL

An Example



UML Relationships

- Represent relationships between modelling elements
  - Dependency
  - Association
  - Generalisation
  - Realisation

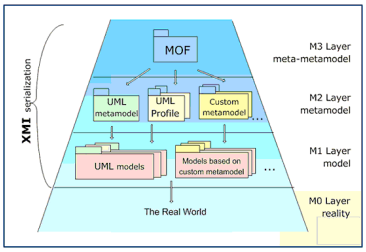
Taken from [Booch 1999] RATIONAL

UML Definition

- UML is defined using UML!
- The UML Metamodel models the UML language.
  - And is defined in UML.
  - A 'Meta' model is a model than defines a model.
- UML is also extensible:
  - Create specialised Profiles.
    - E.g., for web applications, JEE
  - Define new languages.

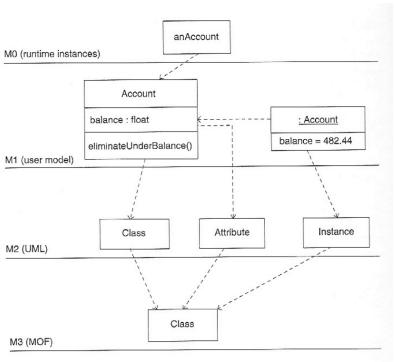
MOF and Metamodeling

- MOF = Meta Object Facility
- Provides a consistent representation for models that can be stored in a repository.
- MOF is defined in MOF.
- 4 layer architecture:



From http://www.jot.fm/issues/issue_2006_11/article4/

MOF Levels

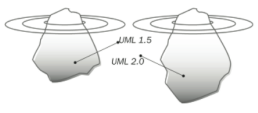


UML Element Extension Mechanisms

- Stereotypes
  - Specify a new model element, derived from an existing one.
  - Appear as stereotype name in guillemets («...» or << ... >>),
    - e.g., «constructor», «JSP» (but «interface» is a classifier).
    - Can also specify the appearance of the element; a different icon.
- Constraints
  - OCL, Object Constraint Language.
    - Specify code-like constraints on elements.
- Tagged Values
  - Tag an element with name-value pairs.

Overview of Diagrams

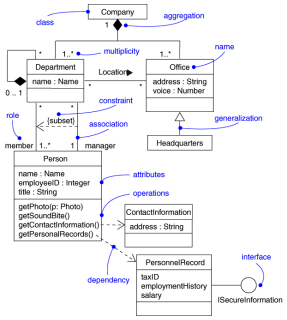
- The rest of the slides in this set introduce a sub-set of the UML diagrams.
  - Much more detail in text book.
  - But won't be attempting to cover all of UML.
  - Focus on what is presented in lectures, use book to expand.
  - The UML iceberg:



UML2 in Action, Grässle et al.

Class Diagrams (1) - Most familiar

- Capture *vocabulary* of a system: classes and associations.



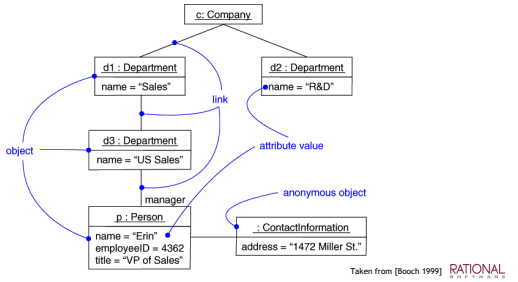
Taken from [Booch 1999] RATIONAL SOFTWARE

Class Diagrams (2)

- Built and refined throughout development.
  - At increasingly lower levels of abstraction.
  - Used to name and model *concepts* in the system.
  - Specify collaborations between objects.
- Notation based on Chen's *Entity-Relationship Diagrams* (ERDs) and Rumbaugh's OMT.

Object Diagrams (1)

- Capture *class instances* and the *links* between them.



Taken from [Booch 1999] RATIONAL

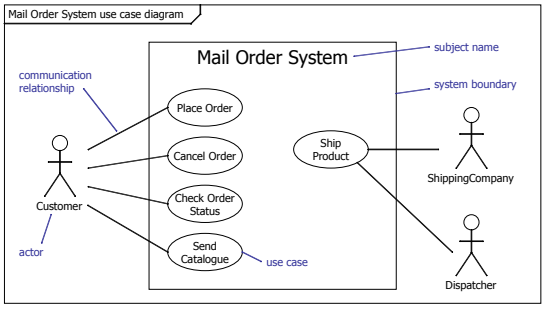
Object Diagrams (2)

- Built during analysis and design.
  - Illustrate data/object *structures*.
  - Specify *snapshots*.
- Don't usually mix class and object icons on the same diagram.
  - Structural v. dynamic views.

Use Case Diagrams

- Built in early stages of development.
  - Specify the context of a system.
  - Plan iterations of development.
  - Validate a system's architecture.
  - Drive implementation & generate test cases.
- Developed by analysts & domain experts during requirements analysis.

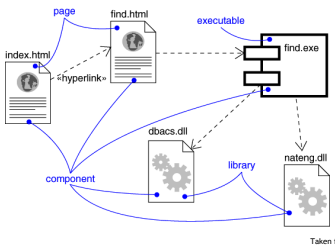
The UML Use Case Diagram



Component Diagrams (1)

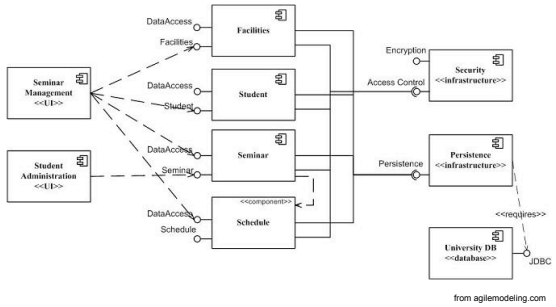
- Capture the *physical structure* of the implementation.

(Notation is a bit different in UML 2.0)



Taken from [Booch 1999] RATIONAL

UML 2 Component Diagram



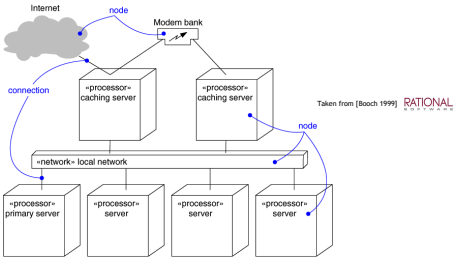
from agilemodeling.com

Component Diagrams (2)

- Built as part of the architectural specification.
  - Define an architecture for an application.
    - Components
    - Ports, connectors and interfaces.
  - The overall structure of the application.
    - The zoomed out point of view.
    - Zoom into a component to see the package and class levels.
    - Components can also be nested.
- Developed by architects and programmers.

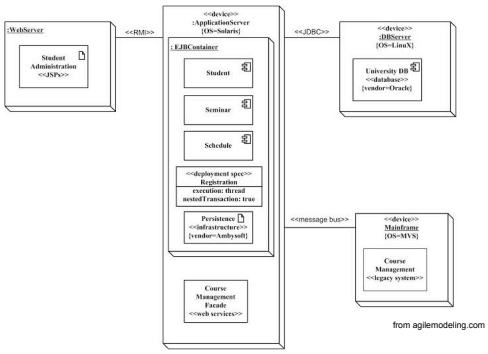
Deployment Diagrams (1)

- Capture the *topology* of a system's hardware and execution environment.



Taken from [Booch 1999] RATIONAL

UML2 Deployment Diagram



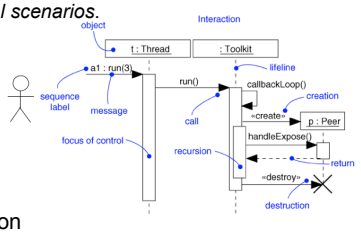
from agilemodeling.com

Deployment Diagrams (2)

- Built as part of the architectural specification.
  - The hardware of the system.
  - The software installed on the hardware.
  - The middleware used to connect machines and services.
  - The distribution of components.
- Zooming out to the look at the hardware environment where the application is installed.
- Developed by architects, networking engineers and system engineers.

Sequence Diagrams

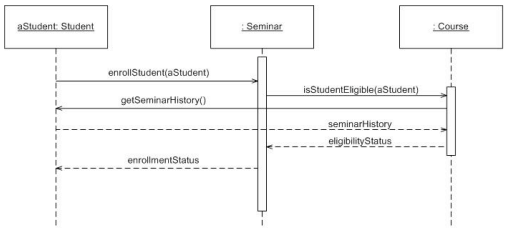
- Capture time-oriented dynamic behaviour.
  - Model *flow of control*.
  - Illustrate *typical scenarios*.



- Notation based on Message Sequence Charts (MSCs).

Taken from [Booch 1999] RATIONAL

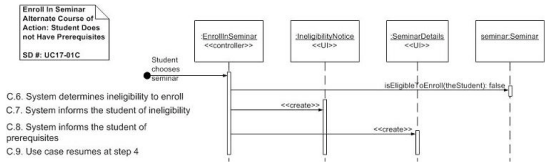
Sequence Diagram Example (1)



from agilemodeling.com

Sequence Diagram Example (2)

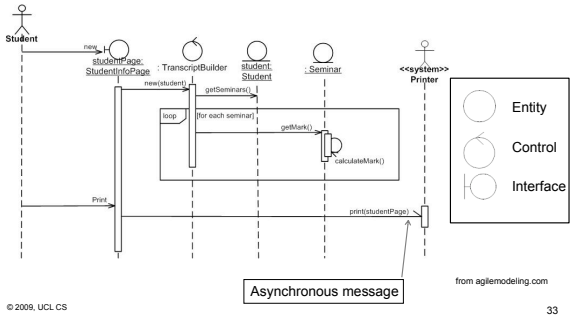
Can be used to illustrate Use Case scenarios



from agilemodeling.com

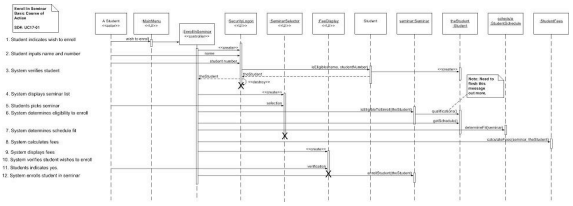
Sequence Diagram Example (3)

- Loops and selection can be shown.



from agilemodeling.com

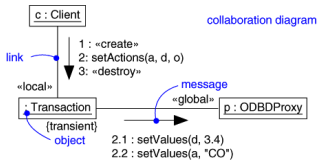
Sequence Diagram Example (4)



from agilemodeling.com

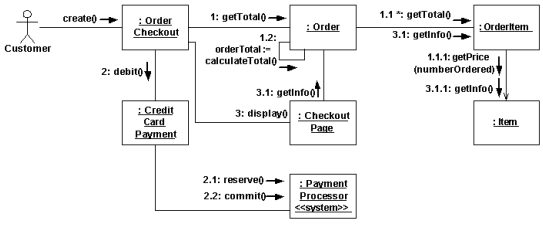
Collaboration/Communication Diagrams

- Capture *message-oriented* dynamic behaviour.
  - Model *flow of control*.
  - Illustrate *coordination* of object structure and control.



Taken from [Booch 1999] RATIONAL SOFTWARE

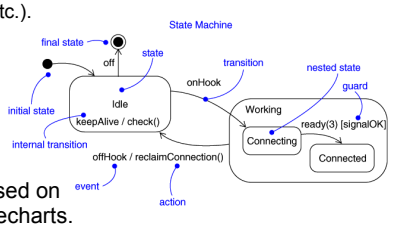
Collaboration Diagram Example



from agilemodeling.com

State Machine Diagrams

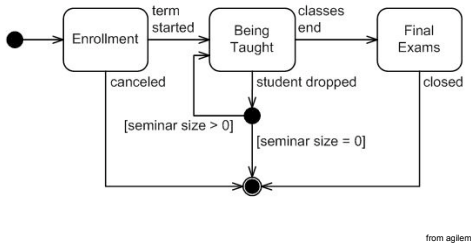
- Capture event-oriented dynamic behaviour.
  - Model *object lifecycle*.
  - Model *reactive objects* (GUIs, hardware devices, sensors, etc.).



- Notation based on Harel's Statecharts.

Taken from [Booch 1999] RATIONAL

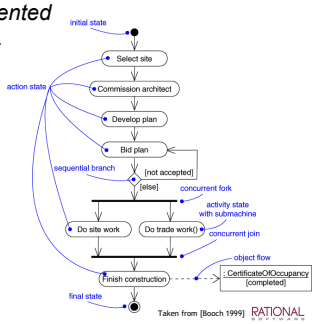
State Machine Example (1)



from agilemodeling.com

Activity Diagrams

- Capture *activity-oriented* dynamic behaviour.
  - Model *business workflows*.
  - Model *operations*.

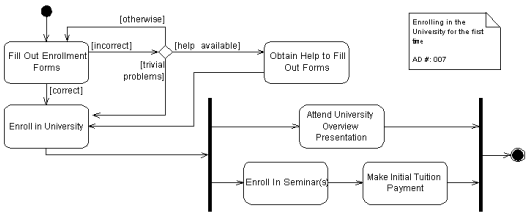


- Notation based on *Petri Nets*.

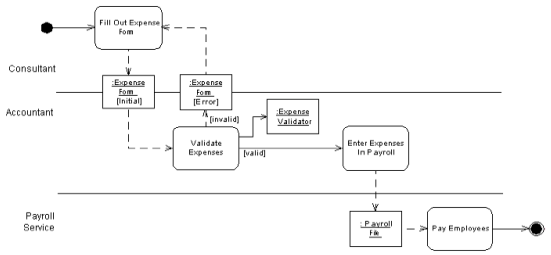
Taken from [Booch 1999] RATIONAL

Example Activity Diagram (1)

- This example captures a *business process*.

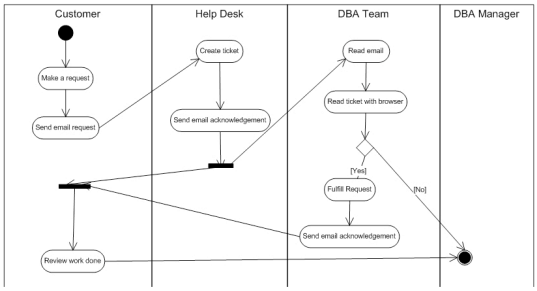


Example Activity Diagram (2)

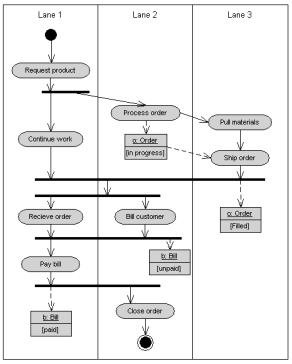


Example Activity Diagram (3)

Swim lanes show which entities carry out which tasks



Example Activity Diagram (4)



Summary

- Overview of various kinds of UML Diagrams.
- UML 2.n has added some more.
- See course text book for more detailed descriptions.
  - *UML 2 and the Unified Process*
    by Jim Arlow and Ila Neustadt,
    pub. by Addison Wesley, 2005
    ISBN 0321321278
    (978-0321321275)