Component & Deployment Diagrams

- Extending UML
- Component
- Deployment

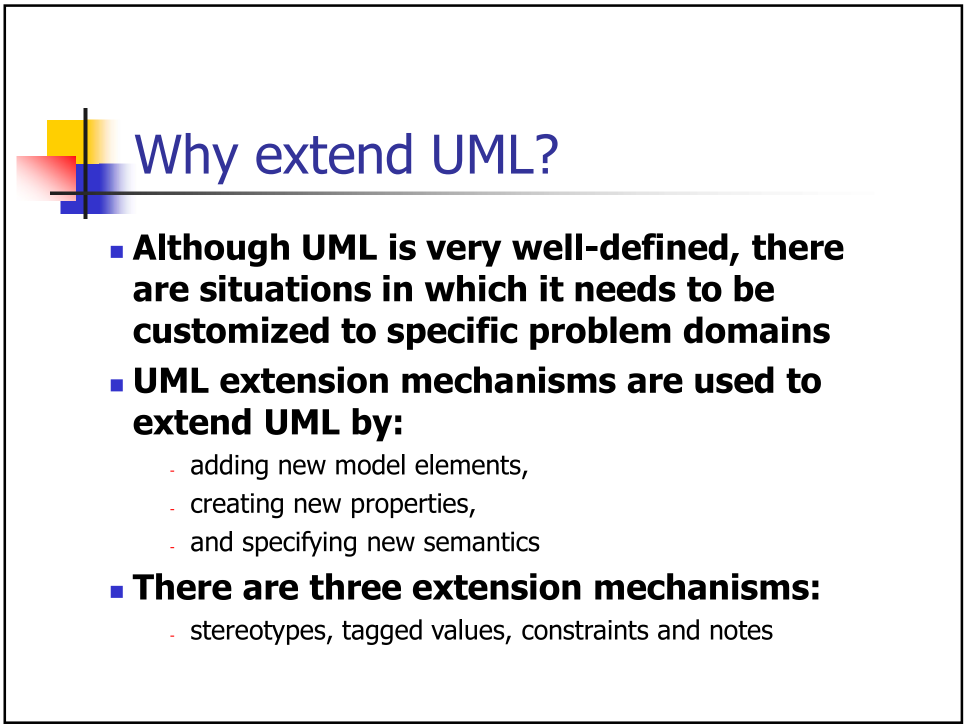
INTRODUCTION

An Uml diagram classification:

- Static
  - Use case diagram, Class diagram
- Dynamic
  - State diagram, Activity diagram, Sequence diagram, Collaboration diagram
- Implementation
  - Component diagram, Deployment diagram

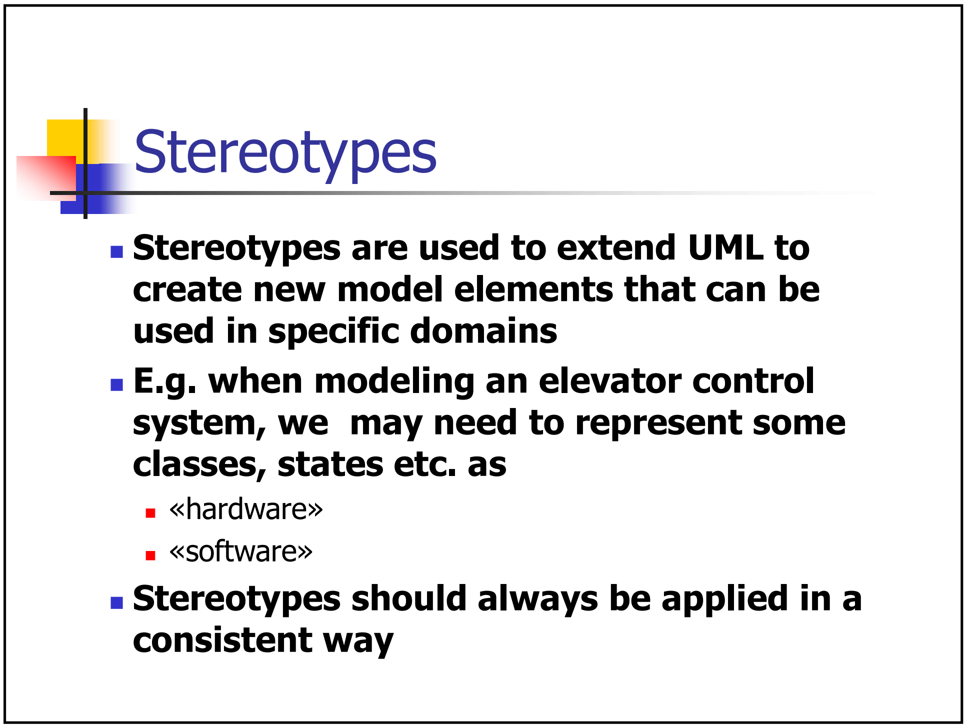
UML components diagrams are

- **Implementation diagrams:** describe the different elements required for implementing a system



Why extend UML?

- **Although UML is very well-defined, there are situations in which it needs to be customized to specific problem domains**
- **UML extension mechanisms are used to extend UML by:**
  - adding new model elements,
  - creating new properties,
  - and specifying new semantics
- **There are three extension mechanisms:**
  - stereotypes, tagged values, constraints and notes

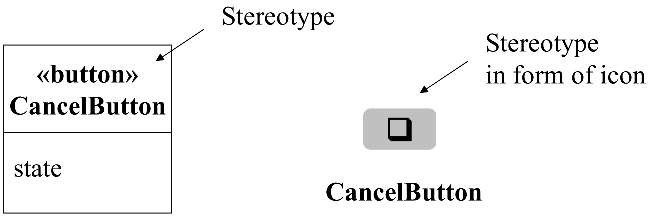


Stereotypes

- **Stereotypes are used to extend UML to create new model elements that can be used in specific domains**
- **E.g. when modeling an elevator control system, we may need to represent some classes, states etc. as**
  - «hardware»
  - «software»
- **Stereotypes should always be applied in a consistent way**

Stereotypes (cont.)

- **Ways of representing a stereotype:**
  - Place the name of the stereotype above the name of an existing UML element (if any)
    - The name of the stereotype needs to be between «» (e.g. «node»)
    - Don't use double '<' or '>' symbols, there are special characters called open and close guillemets
  - Create new icons



Stereotype

«button»
CancelButton

state

Stereotype
in form of icon

CancelButton

5

Tagged Values

Tagged values

Define additional properties for any kind of model elements

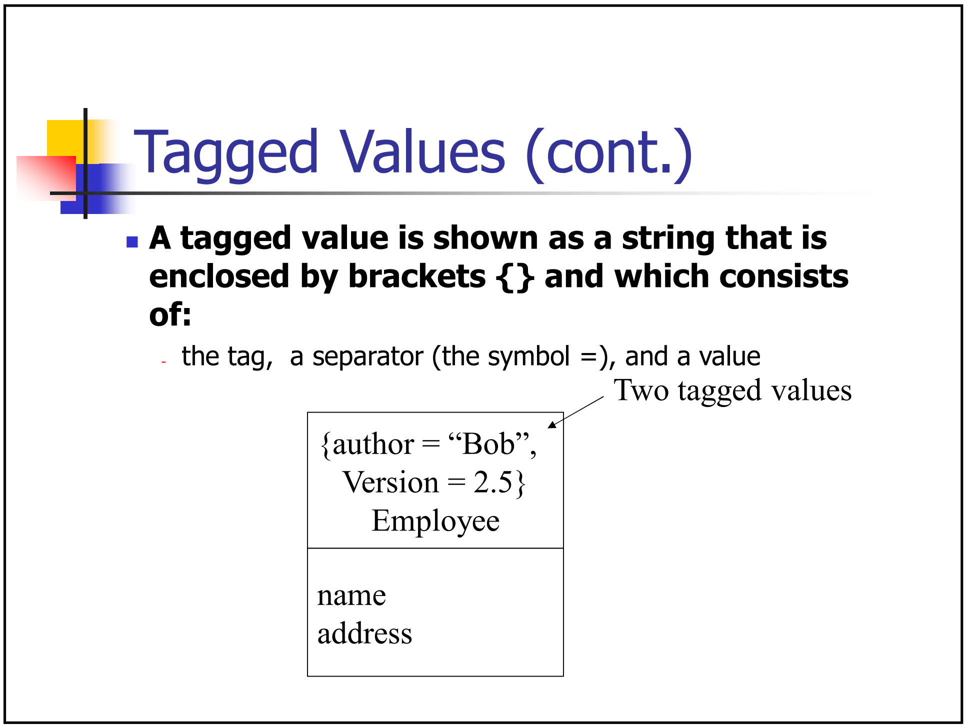
- Can be defined for existing model elements and for stereotypes

Are shown as a tag-value pair where the tag represent the property and the value represent the value of the property

Tagged values can be useful for adding properties about

- code generation
- version control
- configuration management
- authorship
- etc.

6



Tagged Values (cont.)

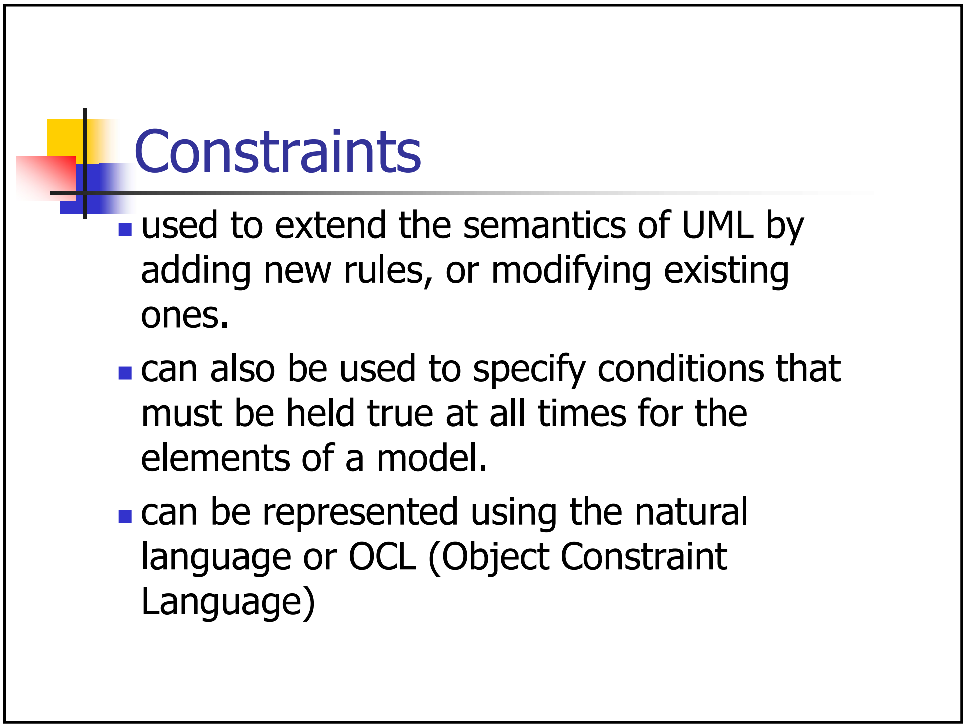
- **A tagged value is shown as a string that is enclosed by brackets {} and which consists of:**
  - the tag, a separator (the symbol =), and a value

Two tagged values

{author = “Bob”,
Version = 2.5}
Employee

name
address

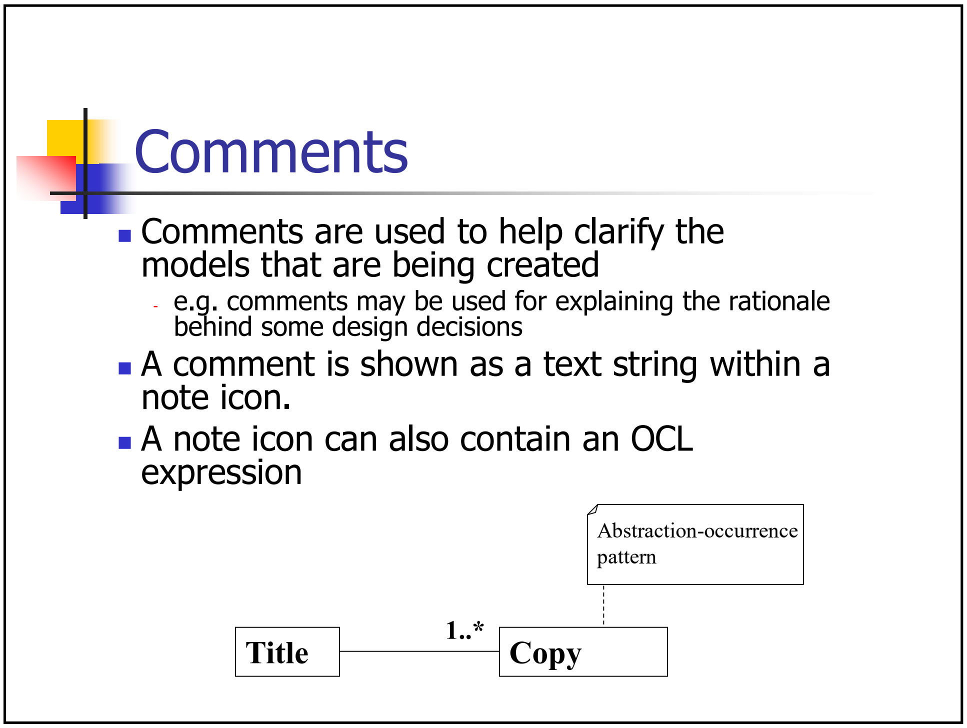
7



Constraints

- used to extend the semantics of UML by adding new rules, or modifying existing ones.
- can also be used to specify conditions that must be held true at all times for the elements of a model.
- can be represented using the natural language or OCL (Object Constraint Language)

8



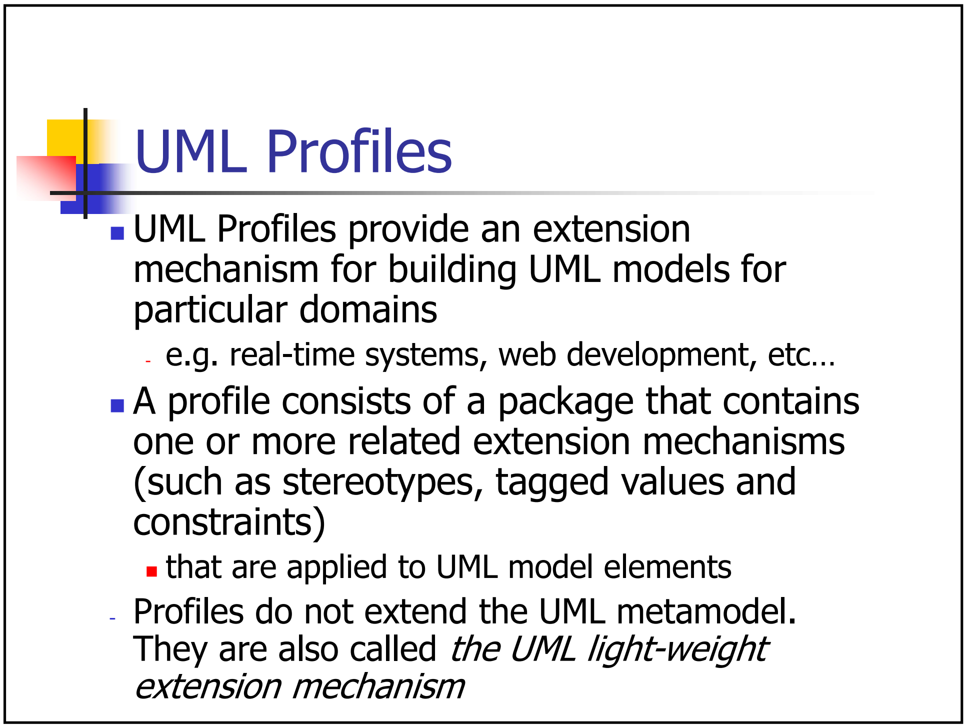
Comments

- Comments are used to help clarify the models that are being created
  - e.g. comments may be used for explaining the rationale behind some design decisions
- A comment is shown as a text string within a note icon.
- A note icon can also contain an OCL expression

Abstraction-occurrence pattern

Title 1..* Copy

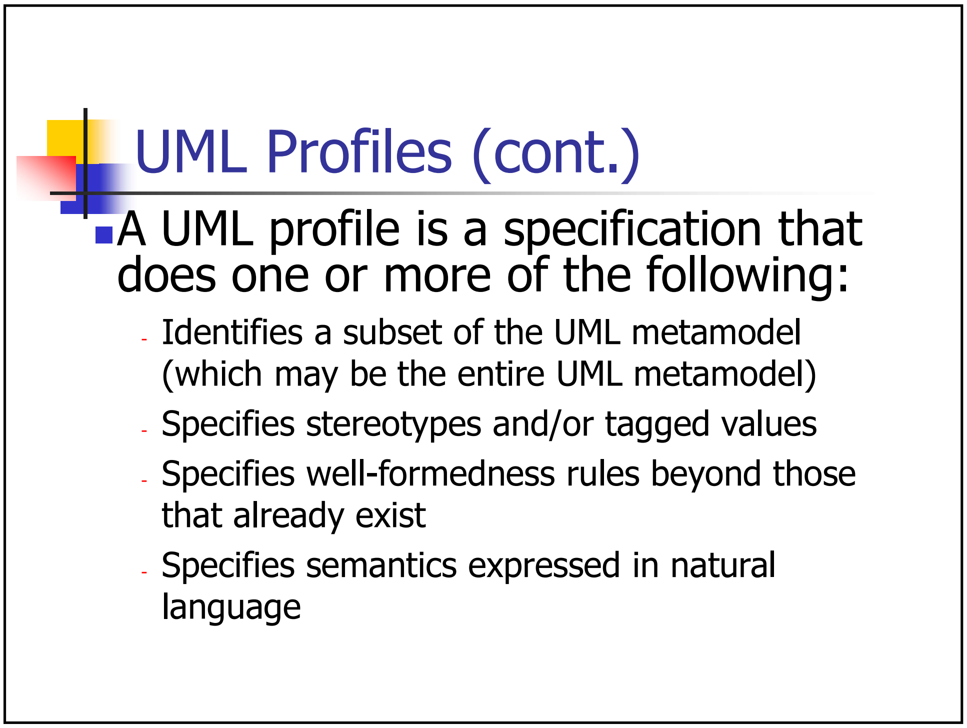
9



UML Profiles

- UML Profiles provide an extension mechanism for building UML models for particular domains
  - e.g. real-time systems, web development, etc…
- A profile consists of a package that contains one or more related extension mechanisms (such as stereotypes, tagged values and constraints)
  - that are applied to UML model elements
- Profiles do not extend the UML metamodel. They are also called *the UML light-weight extension mechanism*

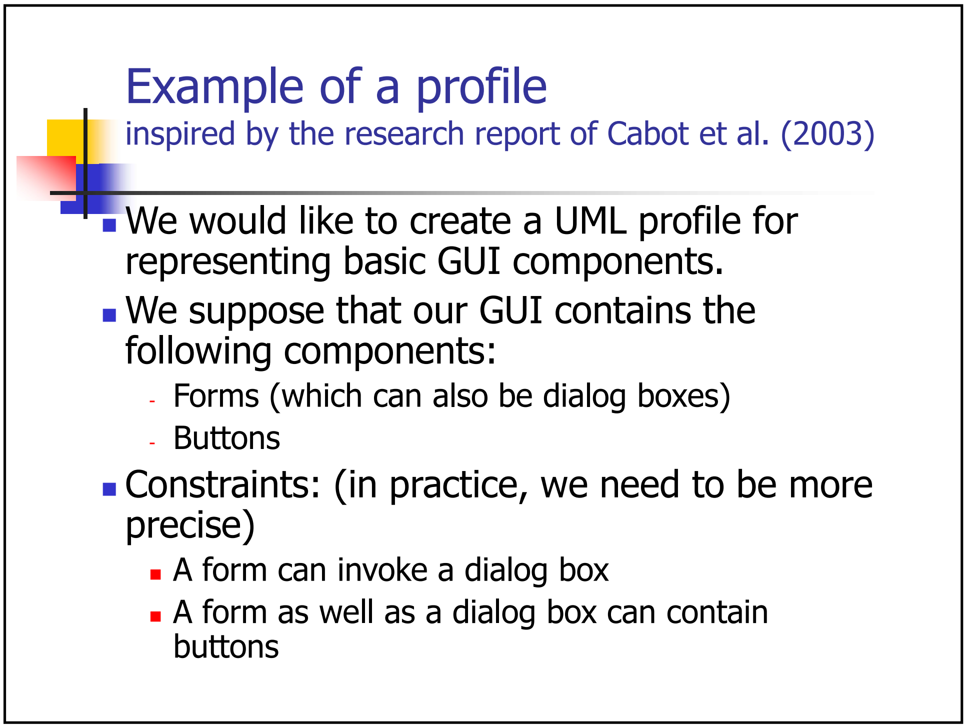
10



UML Profiles (cont.)

- A UML profile is a specification that does one or more of the following:
  - Identifies a subset of the UML metamodel (which may be the entire UML metamodel)
  - Specifies stereotypes and/or tagged values
  - Specifies well-formedness rules beyond those that already exist
  - Specifies semantics expressed in natural language

11



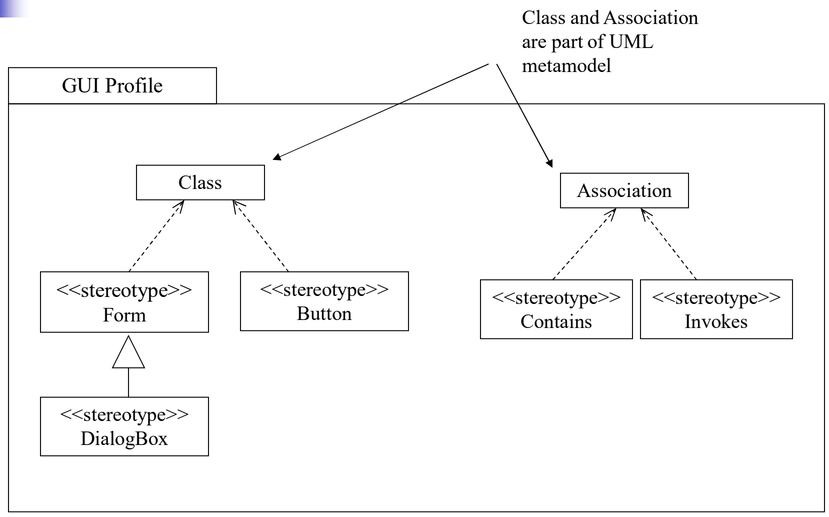
Example of a profile

inspired by the research report of Cabot et al. (2003)

- We would like to create a UML profile for representing basic GUI components.
- We suppose that our GUI contains the following components:
  - Forms (which can also be dialog boxes)
  - Buttons
- Constraints: (in practice, we need to be more precise)
  - A form can invoke a dialog box
  - A form as well as a dialog box can contain buttons

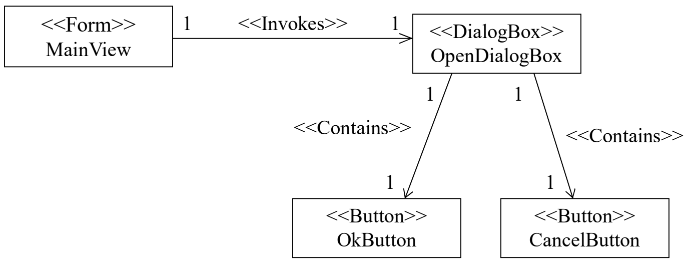
12

The GUI profile package



13

Instance Diagram of the GUI Profile



14

Drawing Subsystems

- System design must model static and dynamic structures:
  - Component Diagrams for static structures
    - show the structure at design time or compilation time
  - Deployment Diagram for dynamic structures
    - show the structure of the run-time system

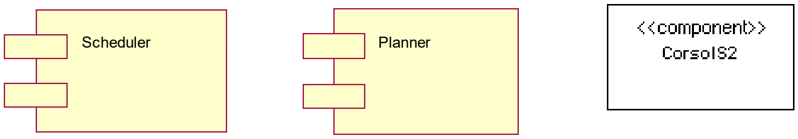
15

Component

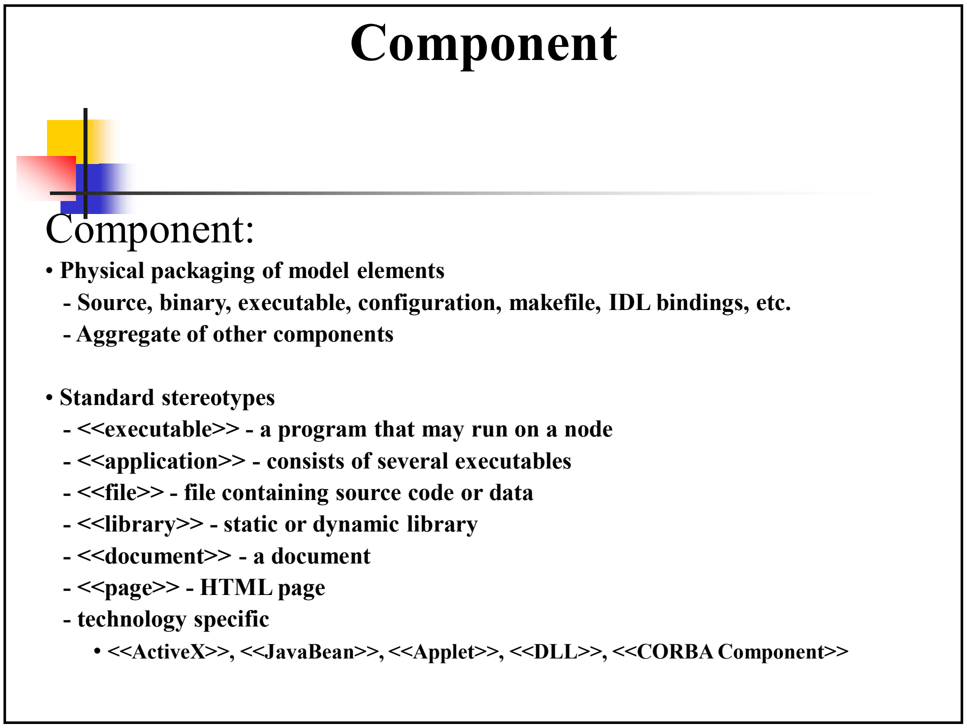
Component:

- **A component is a named physical and replaceable part of a system that represents physical packaging of otherwise logical elements and that conforms to, and provides the realization of, one or more interfaces.**
- A component type represents a piece of software code (source, binary, or executable)
  - A component type has a type name
- A component instance represents a run-time code unit
  - A component instance has a name and a type (***component-name : component-type***)

A component is represented as a rectangle with two small rectangles protruding from its side



16



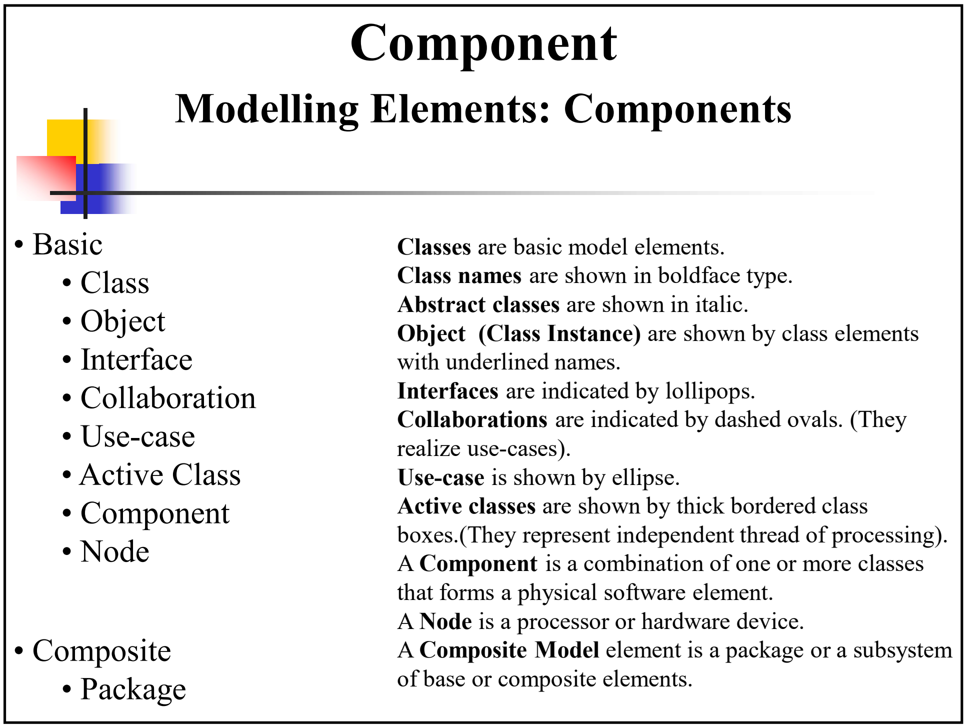
Component

Component:

- **Physical packaging of model elements**
  - **- Source, binary, executable, configuration, makefile, IDL bindings, etc.**
  - **- Aggregate of other components**

- **Standard stereotypes**
  - **- <<executable>> - a program that may run on a node**
  - **- <<application>> - consists of several executables**
  - **- <<file>> - file containing source code or data**
  - **- <<library>> - static or dynamic library**
  - **- <<document>> - a document**
  - **- <<page>> - HTML page**
  - **- technology specific**
    - **<<ActiveX>>, <<JavaBean>>, <<Applet>>, <<DLL>>, <<CORBA Component>>**

17



Component

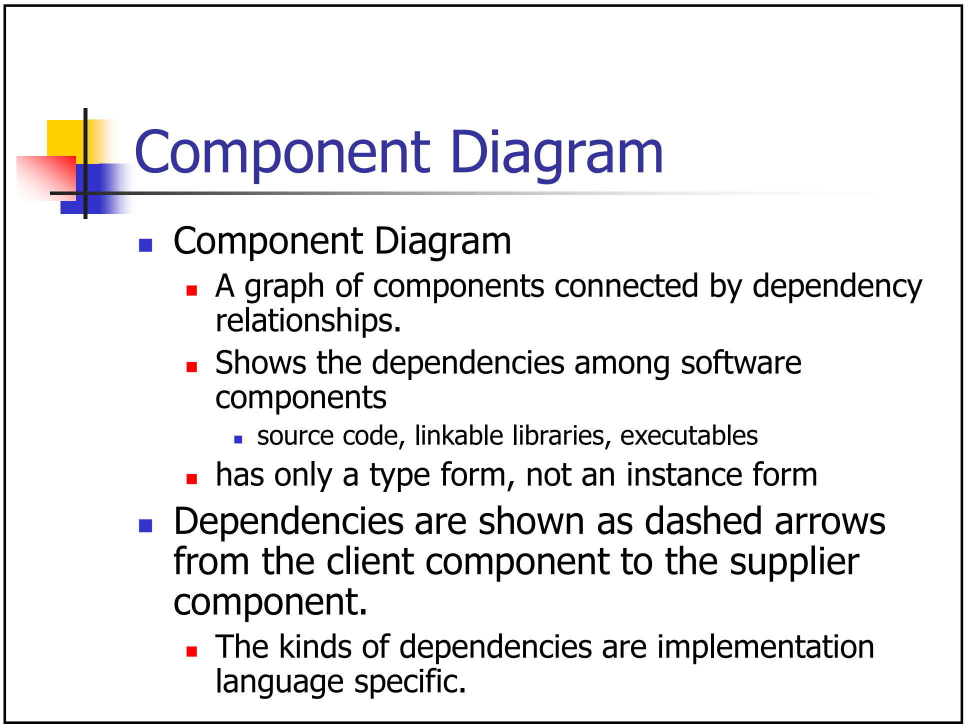
Modelling Elements: Components

- Basic
  - Class
  - Object
  - Interface
  - Collaboration
  - Use-case
  - Active Class
  - Component
  - Node

- Composite
  - Package

Classes are basic model elements.
Class names are shown in boldface type.
Abstract classes are shown in italic.
Object (Class Instance) are shown by class elements with underlined names.
Interfaces are indicated by lollipops.
Collaborations are indicated by dashed ovals. (They realize use-cases).
Use-case is shown by ellipse.
Active classes are shown by thick bordered class boxes.(They represent independent thread of processing).
A **Component** is a combination of one or more classes that forms a physical software element.
A **Node** is a processor or hardware device.
A **Composite Model** element is a package or a subsystem of base or composite elements.

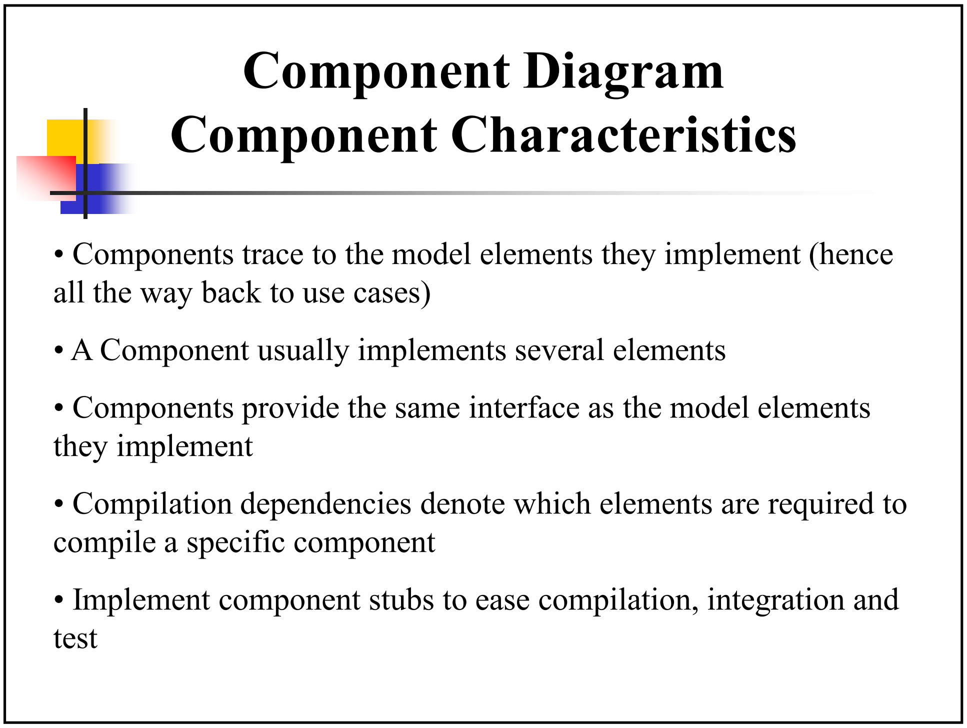
18



Component Diagram

- Component Diagram
  - A graph of components connected by dependency relationships.
  - Shows the dependencies among software components
    - source code, linkable libraries, executables
  - has only a type form, not an instance form
- Dependencies are shown as dashed arrows from the client component to the supplier component.
  - The kinds of dependencies are implementation language specific.

19



Component Diagram
Component Characteristics

• Components trace to the model elements they implement (hence all the way back to use cases)

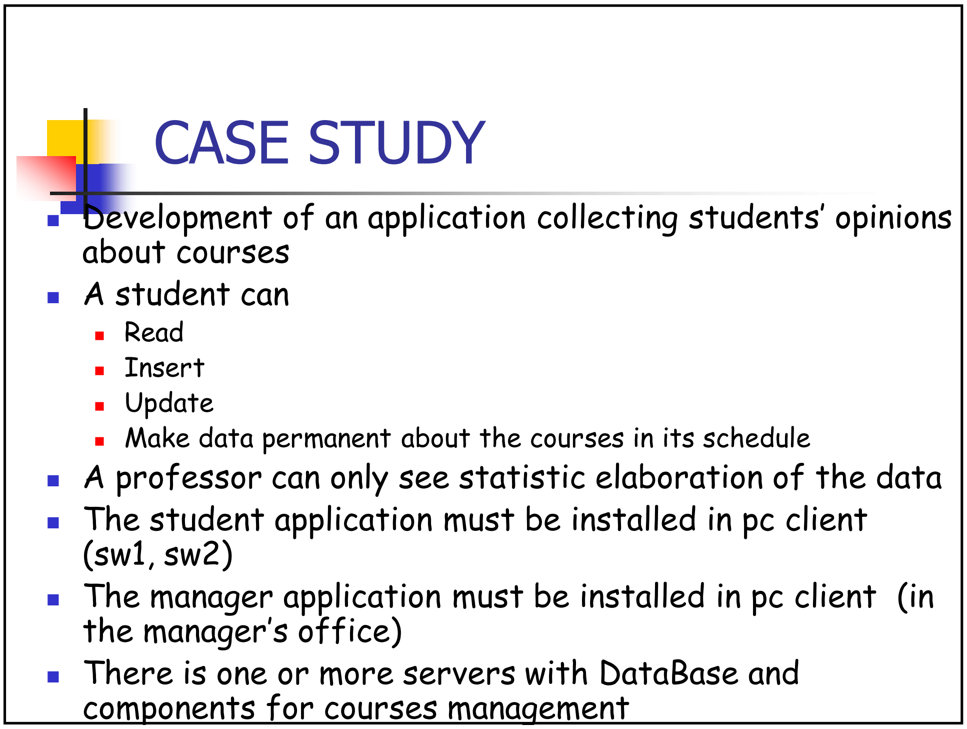
• A Component usually implements several elements

• Components provide the same interface as the model elements they implement

• Compilation dependencies denote which elements are required to compile a specific component

• Implement component stubs to ease compilation, integration and test

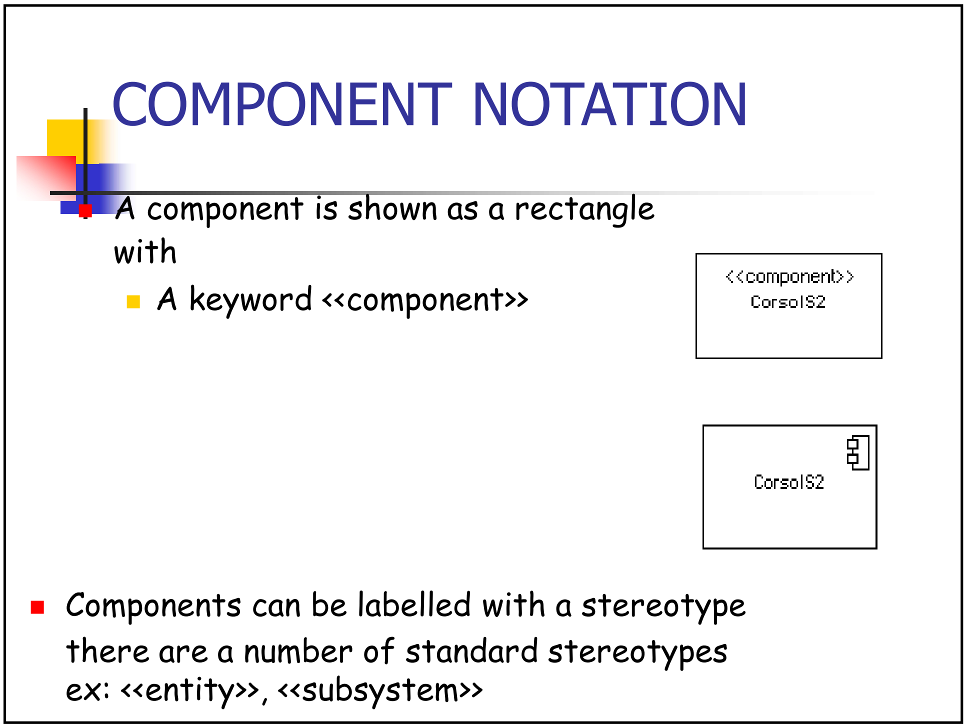
20



CASE STUDY

- Development of an application collecting students' opinions about courses
- A student can
  - Read
  - Insert
  - Update
  - Make data permanent about the courses in its schedule
- A professor can only see statistic elaboration of the data
- The student application must be installed in pc client (sw1, sw2)
- The manager application must be installed in pc client (in the manager's office)
- There is one or more servers with DataBase and components for courses management

21



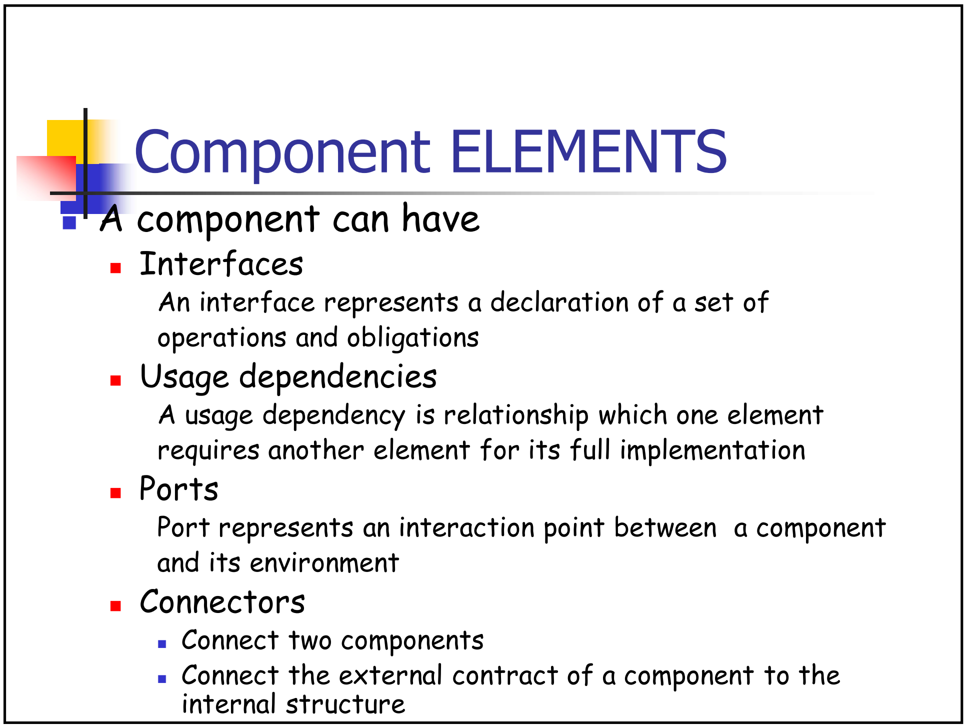
COMPONENT NOTATION

- A component is shown as a rectangle with
  - A keyword <<component>>

<<component>>
CorsoIS2
CorsoIS2

- Components can be labelled with a stereotype there are a number of standard stereotypes ex: <<entity>>, <<subsystem>>

22



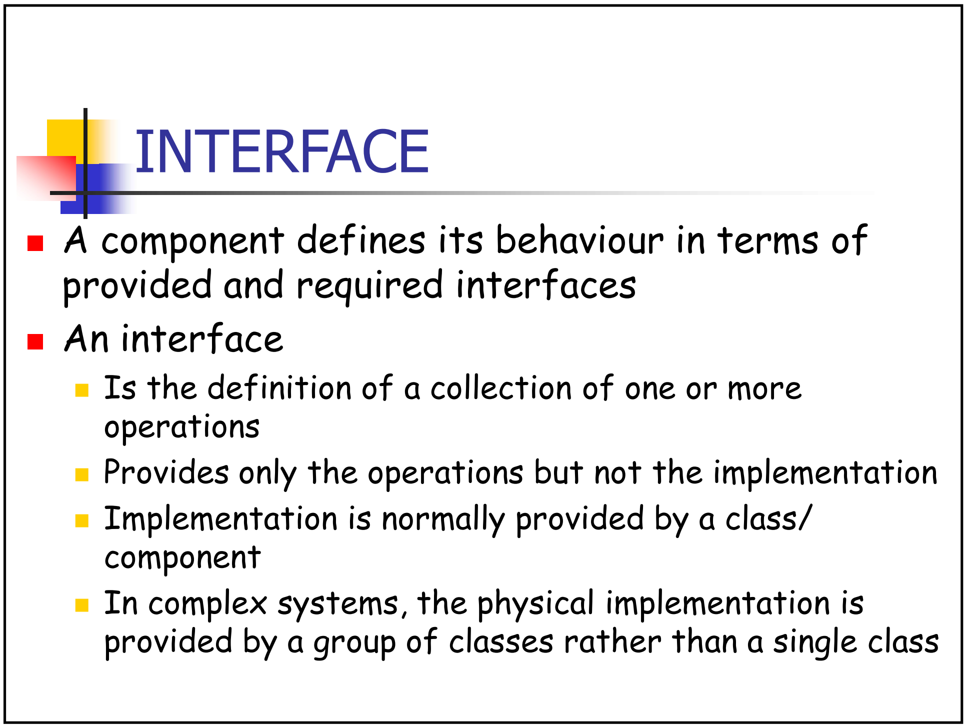
Component ELEMENTS

- A component can have
  - Interfaces

    An interface represents a declaration of a set of operations and obligations
  - Usage dependencies

    A usage dependency is relationship which one element requires another element for its full implementation
  - Ports

    Port represents an interaction point between a component and its environment
  - Connectors
    - Connect two components
    - Connect the external contract of a component to the internal structure

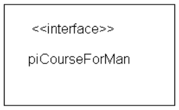


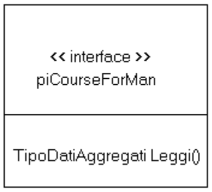
INTERFACE

- A component defines its behaviour in terms of provided and required interfaces
- An interface
  - Is the definition of a collection of one or more operations
  - Provides only the operations but not the implementation
  - Implementation is normally provided by a class/ component
  - In complex systems, the physical implementation is provided by a group of classes rather than a single class

INTERFACE

- May be shown using a rectangle symbol with a keyword <<interface>> preceding the name
- For displaying the full signature, the interface rectangle can be expanded to show details

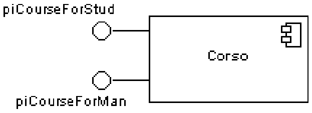


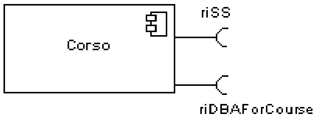


- Can be
  - Provided
  - Required

INTERFACE

- A provided interface
  - Characterize services that the component offers to its environment
  - Is modeled using a ball, labelled with the name, attached by a solid line to the component

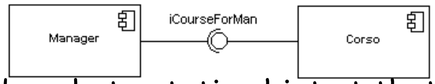




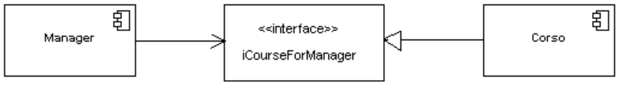
- A required interface
  - Characterize services that the component expects from its environment
  - Is modeled using a socket, labelled with the name, attached by a solid line to the component
  - In UML 1.x were modeled using a dashed arrow

INTERFACE

- Where two components/classes provide and require the same interface, these two notations may be combined



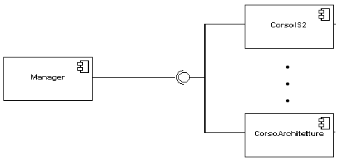
- The ball-and-socket notation hint at that interface in question serves to mediate interactions between the two components
- If an interface is shown using the rectangle symbol, we can use an alternative notation, using dependency arrows



27

INTERFACE

- In a system context where there are multiple components that require or provide a particular interface, a notation abstraction can be used that combines by joining the interfaces

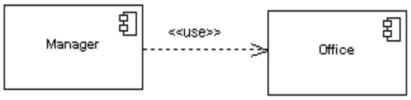


- A component
  - Specifies a CONTRACT of the services that it provides to its clients and that it requires from others components in terms of its provided and required interfaces
  - Can be replaced
  - The system can be extended

28

DEPENDENCIES

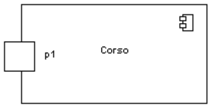
- Components can be connected by usage dependencies

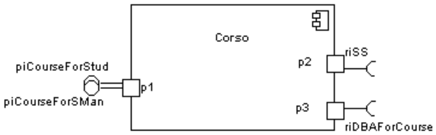


- Usage Dependency
  - A usage dependency is relationship which one element requires another element for its full implementation
  - Is a dependency in which the client requires the presence of the supplier
  - Is shown as dashed arrow with a <<use>> keyword
  - The arrowhead point from the dependent component to the one of which it is dependent

PORT

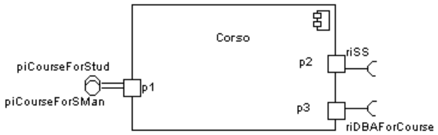
- Specifies a distinct interaction point
  - Between that component and its environment
  - Between that component and its internal parts
- Is shown as a small square symbol
- Ports can be named, and the name is placed near the square symbol
- Is associated with the interfaces that specify the nature of the interactions that may occur over a port

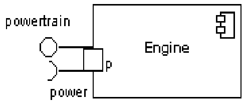




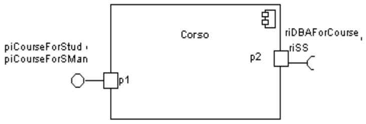
PORT

- Ports can support unidirectional communication or bi-directional communication





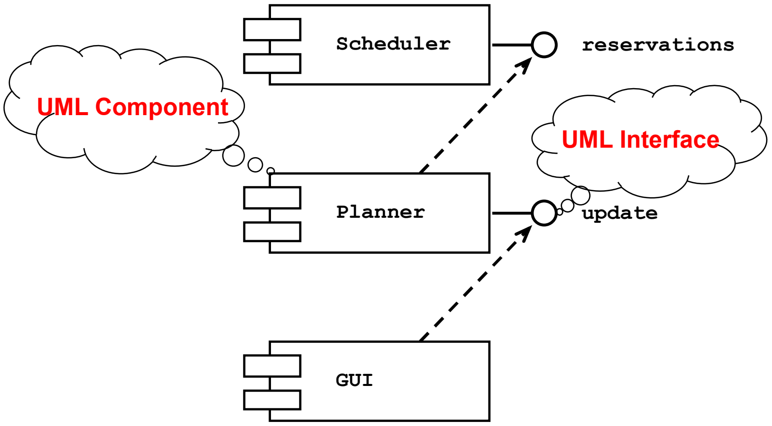
- If there are multiple interfaces associated with a port, these interfaces may be listed with the interface icon, separated by a commas



PORT

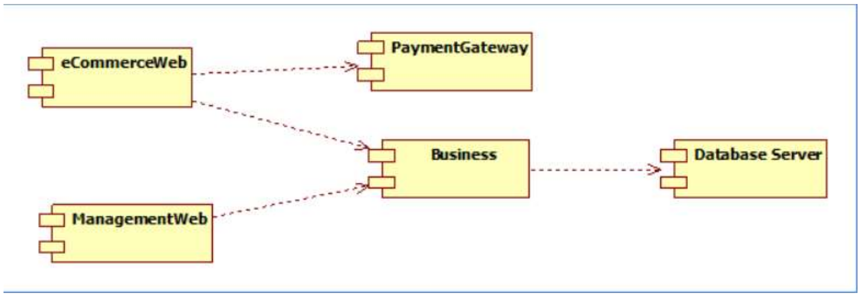
- All interactions of a component with its environment are achieved through a port
- The internals are fully isolated from the environment
- This allows such a component to be used in any context that satisfies the constraints specified by its ports
- Ports are not defined in UML 1.x

Component Diagram Examples



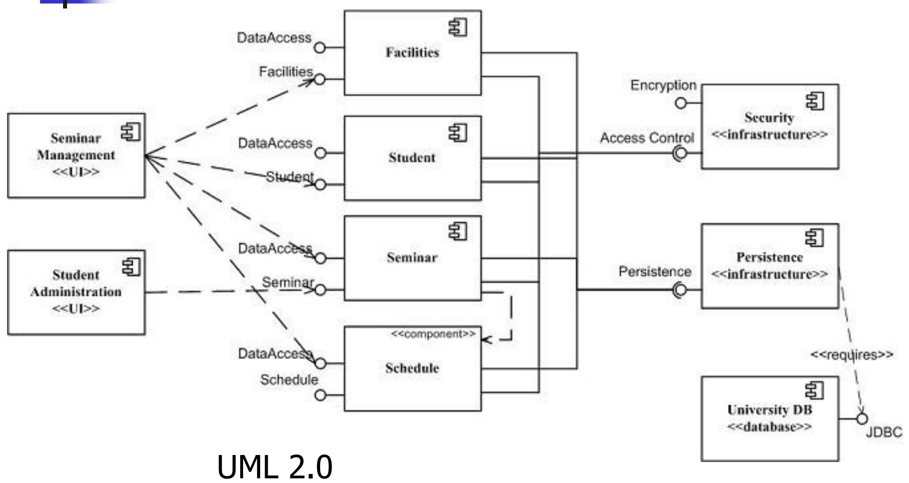
33

Component Diagram Examples



34

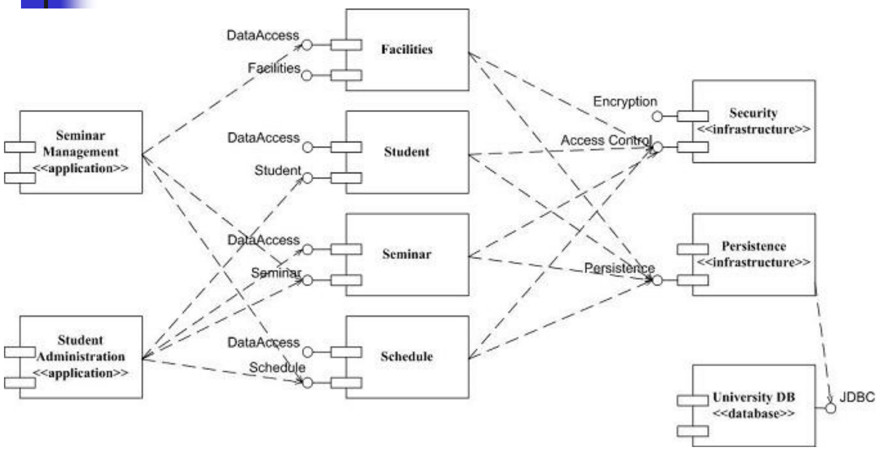
Component Diagram Examples



UML 2.0

35

Component Diagram Examples



UML 1.0

36

UML 1.x vs. 2.x Component Diagrams

Notational differences:

- UML 2 components are modeled as simple rectangles
  - uses this symbol as a visual stereotype within the rectangle
- UML 1.x there were depicted as rectangles with two smaller rectangles jutting out from the left-hand side. As you can see
- Both diagrams model dependencies, either between components or between components and interfaces.
  - both diagrams use the lollipop symbol to indicate an implemented interface
    - the UML 2 version introduces the socket symbol to indicate a required interface.

DEPLOYMENT DIAGRAMS

- There is a strong link between components diagrams and deployment diagrams
- Deployment diagrams
  - Show the physical relationship between hardware and software in a system
  - Hardware elements:
    - Computers (clients, servers)
    - Embedded processors
    - Devices (sensors, peripherals)
  - Are used to show the nodes where software components reside in the run-time system

Deployment Diagram

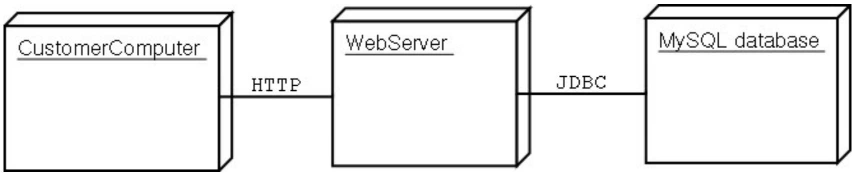
- A deployment diagram is a graph of nodes connected by communication associations. Nodes may contain component instances; indicates “Component” run on nodes.
- Components may contain objects; indicates “Objects” is part of the component.
- Components are connected to other components by dashed-arrow dependencies.

Deployment Diagram

A Deployment Diagram shows the actual Hardware configuration consisting of

- Nodes (processors)
- Software - Components
- Processes
- Objects

Deployment Diagram



- Captures the distinct number of computers involved
- Shows the communication modes employed
- Component diagrams can be embedded into deployment diagrams effectively…

41

DEPLOYMENT DIAGRAMS

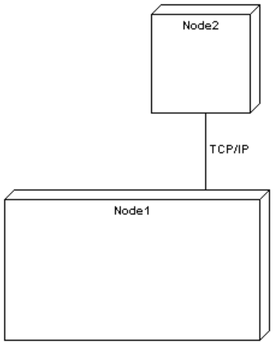
Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestation of code units.

Components that do not exist as run-time entities do not appear in Deployment diagrams.

42

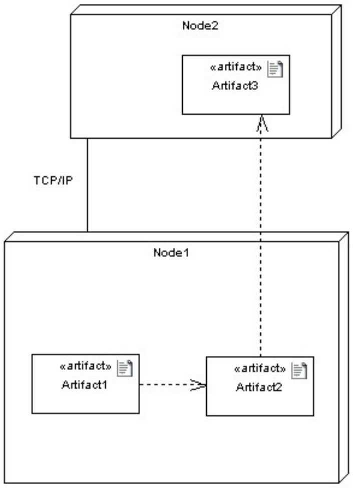
DEPLOYMENT DIAGRAMS

- Deployment diagram
  - Contains nodes and connections
  - A node usually represent a piece of hardware in the system
    - A connection depicts the communication path used by the hardware to communicate
    - Usually indicates the method such as TCP/IP



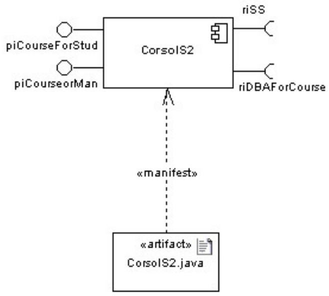
DEPLOYMENT DIAGRAMS

- Deployment diagrams contain artifact
- An artifact
  - Is the specification of a phisycal piece of information
  - Ex: source files, binary executable files, table in a database system,….
  - An artifact defined by the user represents a concrete element in the physical world



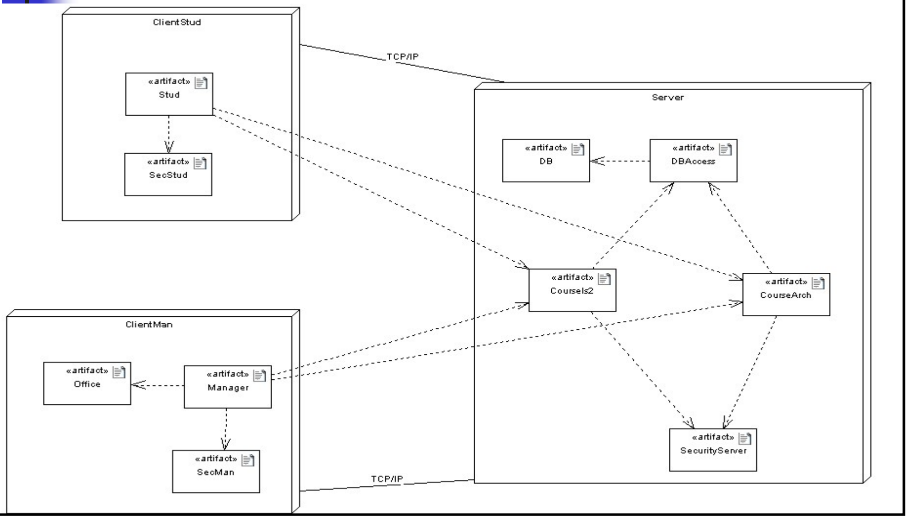
DEPLOYMENT DIAGRAMS

- An artifact manifest one or more model elements
- A <<manifestation>> is the concrete physical of one or more model elements by an artifact
- This model element often is a component



- A manifestation is notated as a dashed line with an open arrow-head labeled with the keyword <<manifest>>

DEPLOYMENT DIAGRAMS

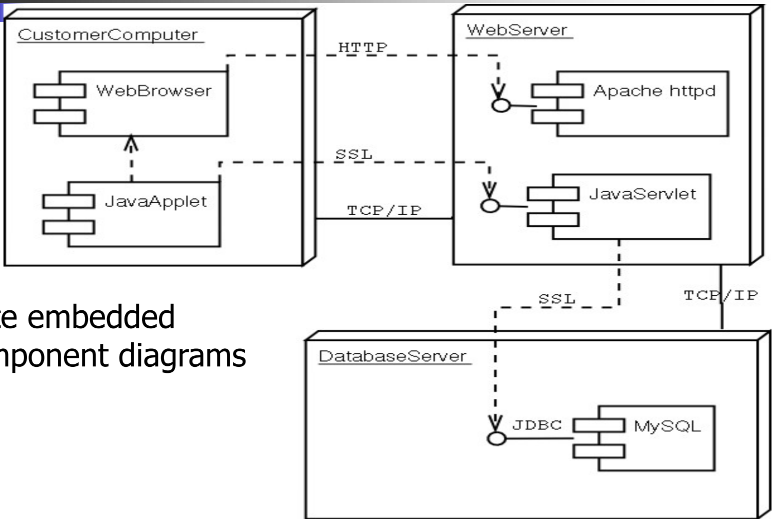


Deployment Diagram

- Deployment diagrams are useful for showing a system design after the following decisions are made
  - Subsystem decomposition
  - Concurrency
  - Hardware/Software Mapping
- A deployment diagram is a graph of nodes connected by communication associations.
  - Nodes are shown as 3-D boxes.
  - Nodes may contain component instances.
  - Components may contain objects (indicating that the object is part of the component)

47

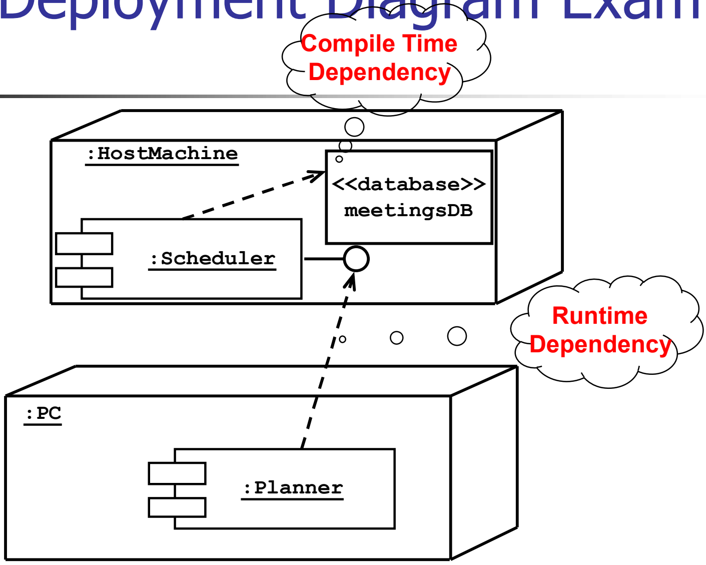
Deployment Diagrams



Note embedded component diagrams

48

Deployment Diagram Example



Compile Time Dependency

:HostMachine

<<database>>
meetingsDB

:Scheduler

Runtime Dependency

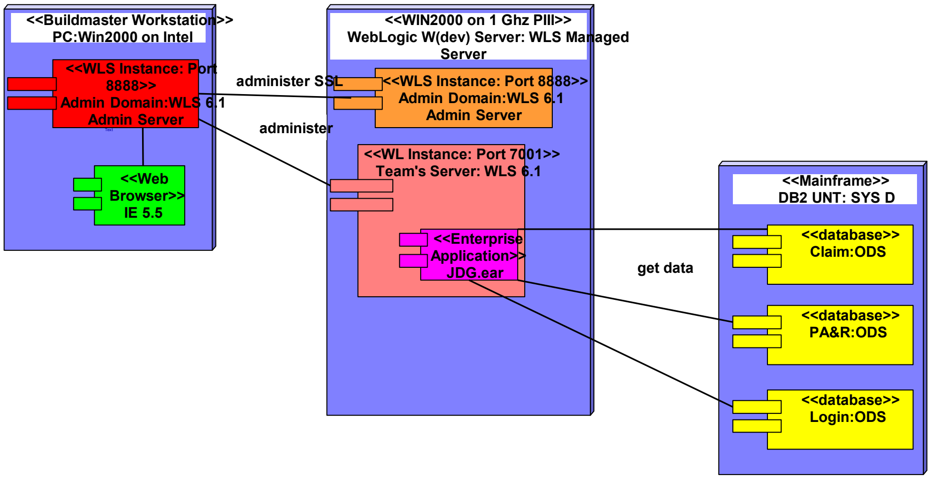
:PC

:Planner

Deployment Diagram

Sample Deployment Diagram

This diagram shows 3 machines: 2 PC workstations running WebLogic and a Mainframe running the ODSs in DB2



<<Buildmaster Workstation>>
PC:Win2000 on Intel

<<WLS Instance: Port 8888>>
Admin Domain:WLS 6.1
Admin Server

<<Web Browser>>
IE 5.5

administer SSL

administer

<<WIN2000 on 1 Ghz PIII>>
WebLogic W(dev) Server: WLS Managed Server

<<WLS Instance: Port 8888>>
Admin Domain:WLS 6.1
Admin Server

<<WL Instance: Port 7001>>
Team's Server: WLS 6.1

<<Enterprise Application>>
JDG.ear

get data

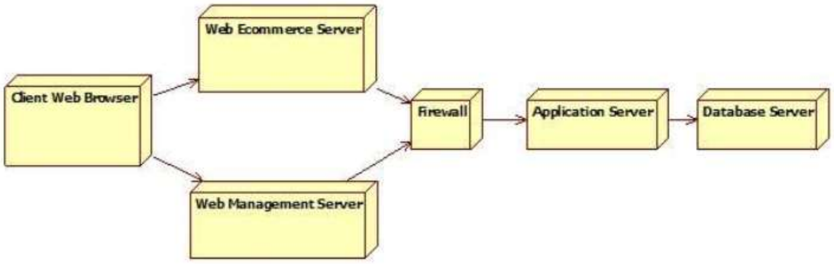
<<Mainframe>>
DB2 UNT: SYS D

<<database>>
Claim:ODS

<<database>>
PA&R:ODS

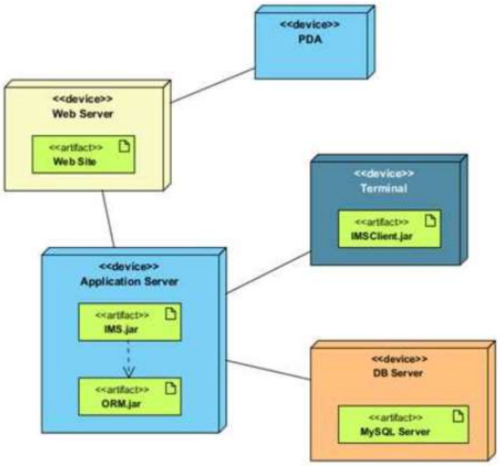
<<database>>
Login:ODS

Deployment Diagram



51

Deployment + Component Diagram



52