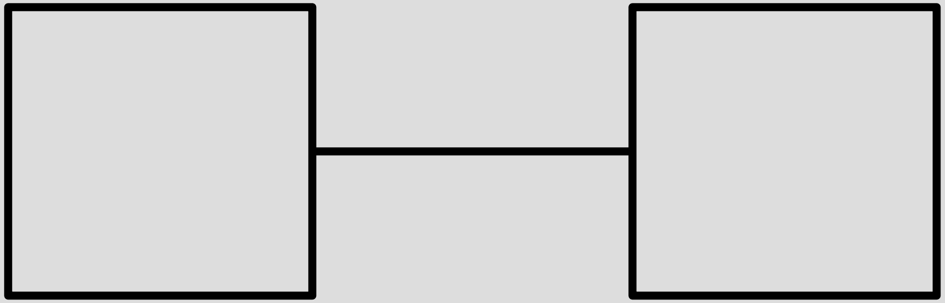
Software Architecture Structures and Views



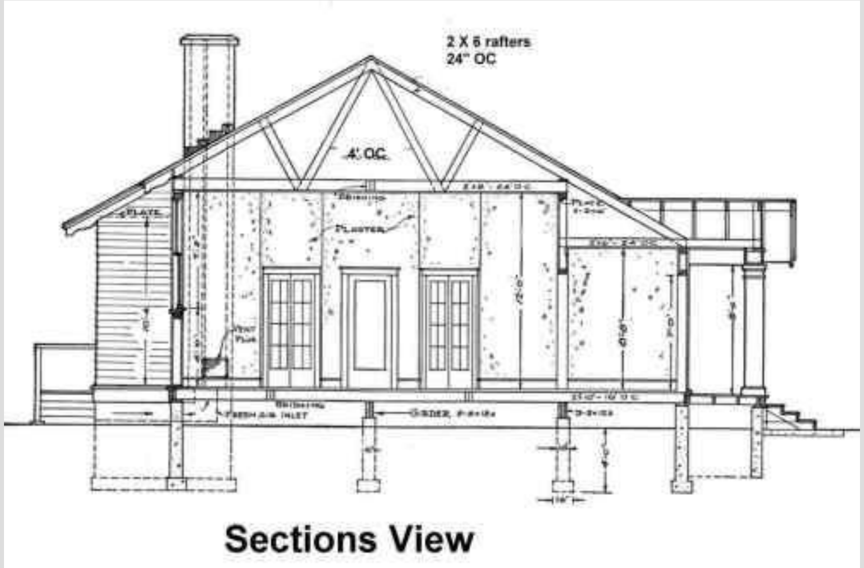
Topics

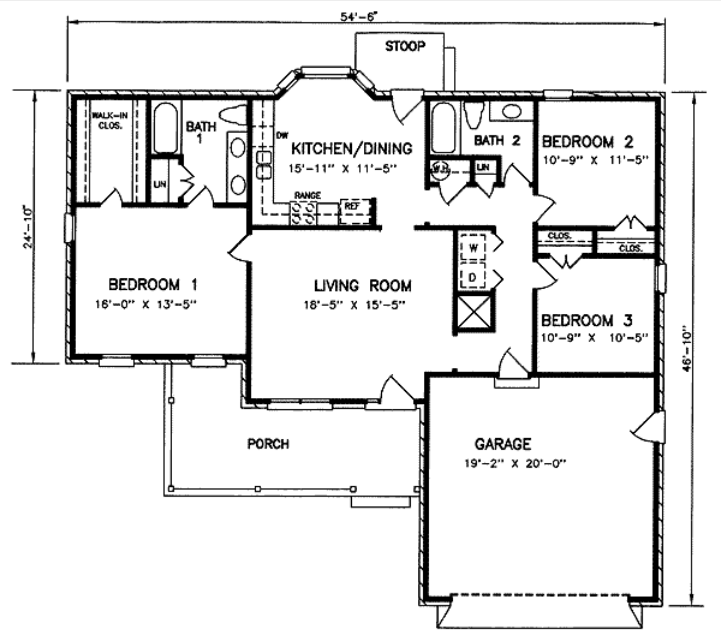
- Structures and views
  - Modules
  - Component and connector
  - Allocation
- Examine some software architecture view examples

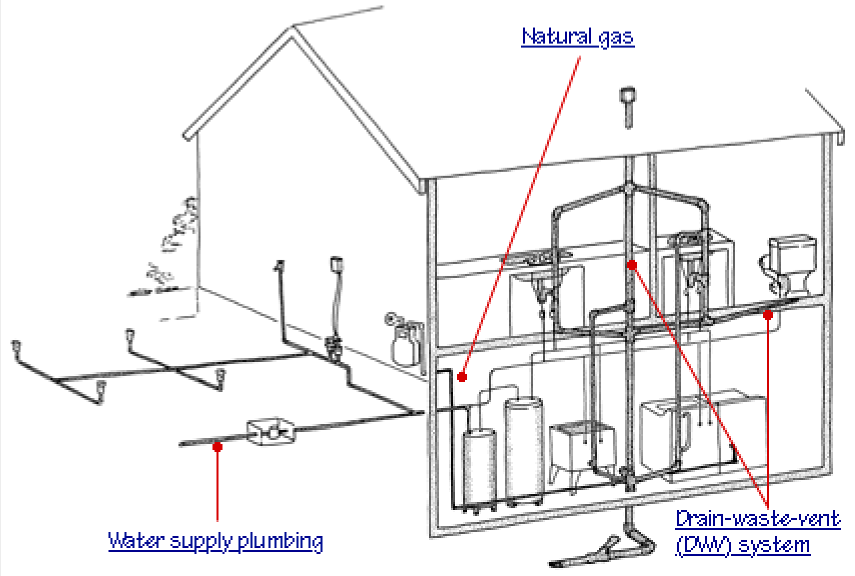
Structures and Views

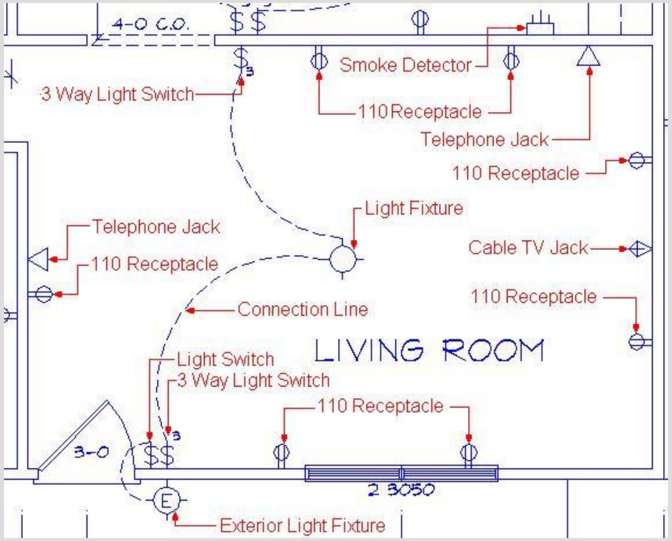
- Problem: **difficult** to **comprehend** and discuss **all system structures at once**
- **Structure**: The set of elements itself, as they exist in software or hardware
- **View**: a representation of a **coherent set of architectural elements and their relationships**

> *“Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view.”*


2 X 6 rafters
24" OC
4' OC
Sections View


54'–6"
STOOP
WALK–IN CLOS.
BATH 1
LIN
DW
KITCHEN/DINING
15'–11" X 11'–5"
RANGE
REF
BATH 2
LIN
BEDROOM 2
10'–9" X 11'–5"
CLOS.
CLOS.
24'–10"
BEDROOM 1
16'–0" X 13'–5"
LIVING ROOM
18'–5" X 15'–5"
W
D
BEDROOM 3
10'–9" X 10'–5"
46'–10"
PORCH
GARAGE
19'–2" X 20'–0"


Natural gas
Water supply plumbing
Drain-waste-vent (DWV) system


4-0 C.O.
Smoke Detector
3 Way Light Switch
110 Receptacle
Telephone Jack
110 Receptacle
Light Fixture
Telephone Jack
110 Receptacle
Cable TV Jack
Connection Line
110 Receptacle
Light Switch
LIVING ROOM
3 Way Light Switch
110 Receptacle
3-0
2 3050
E
Exterior Light Fixture

Possible Views (Viewpoints)

- Functional/logic view
- Module/code view
- Development/structural view
- Concurrency/process/runtime/thread view
- Physical/deployment/install view
- User action/feedback view
- Data view/data model

- Which of the views is the architecture? **None of them**
- Which views convey the architecture? **All of them**

4+1 View Model

[Philippe Kruchten, 1995]

- **Logical view-** e.g. object model using object oriented design method
- **Process view** – concurrency and synchronization aspects
- **Physical view** – mapping of components to hardware, distribution aspect
- **Development view** – organization of the actual software modules – libraries, packages, subsystems
- **+ Use case view**

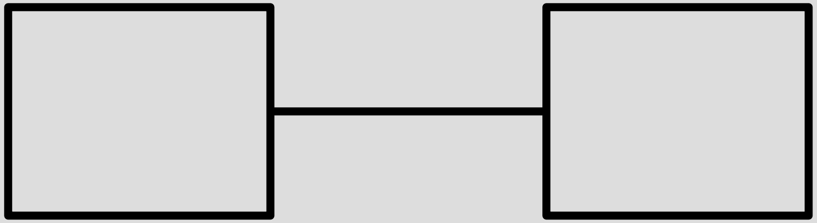
System: Containers, Components, Classes

- Start with a **context diagram** for the system big picture
- **System** is decomposed into containers
- **Containers** – high level technology choices, “anything that can host code or data”
- **Components** – decompose each container into logical modules and their relationships
- **Classes** – decompose components into classes (UML) as needed

Software Architecture for Developers, Simon Brown, LeanPub.com

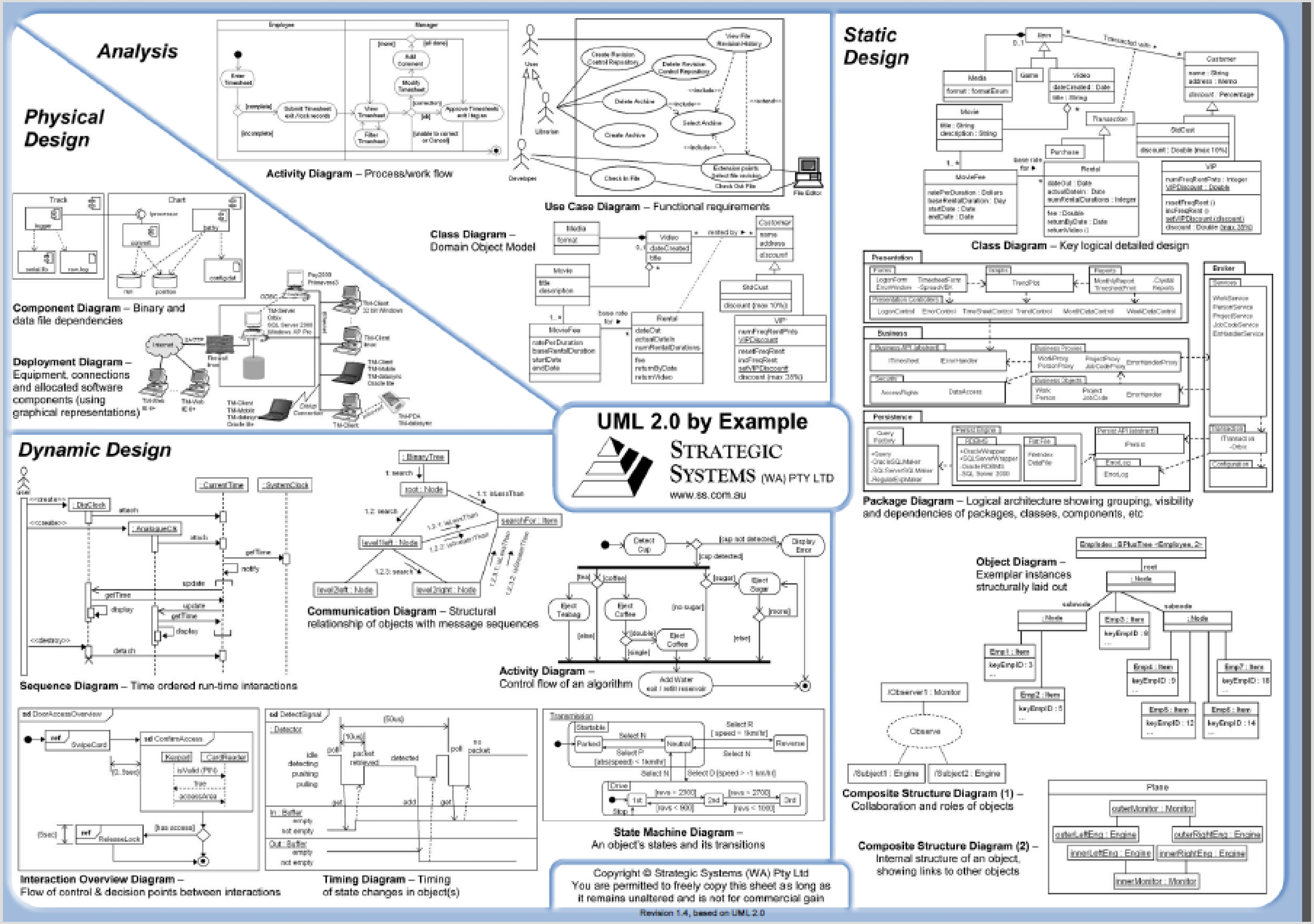
View Notations

- **Informal** – ad hoc conventions using graphical editing tools and natural language descriptions



Provide a key!

- **Semiformal** – prescribed graphical element conventions and rules of construction; e.g., **UML**
- **Formal** – views are expressed in a notation that has a precise (math based) semantics that allows for formal analysis; architecture description languages (ADL's) – e.g., ABACUS



Analysis

Physical Design

Static Design

Dynamic Design

Activity Diagram – Process/work flow

Use Case Diagram – Functional requirements

Class Diagram – Domain Object Model

Class Diagram – Key logical detailed design

Component Diagram – Binary and data file dependencies

Deployment Diagram – Equipment, connections and allocated software components (using graphical representations)

UML 2.0 by Example

STRATEGIC SYSTEMS (WA) PTY LTD
www.ss.com.au

Package Diagram – Logical architecture showing grouping, visibility and dependencies of packages, classes, components, etc

Object Diagram – Exemplar instances structurally laid out

Communication Diagram – Structural relationship of objects with message sequences

Sequence Diagram – Time ordered run-time interactions

Activity Diagram – Control flow of an algorithm

Interaction Overview Diagram – Flow of control & decision points between interactions

Timing Diagram – Timing of state changes in object(s)

State Machine Diagram – An object's states and its transitions

Composite Structure Diagram (1) – Collaboration and roles of objects

Composite Structure Diagram (2) – Internal structure of an object, showing links to other objects

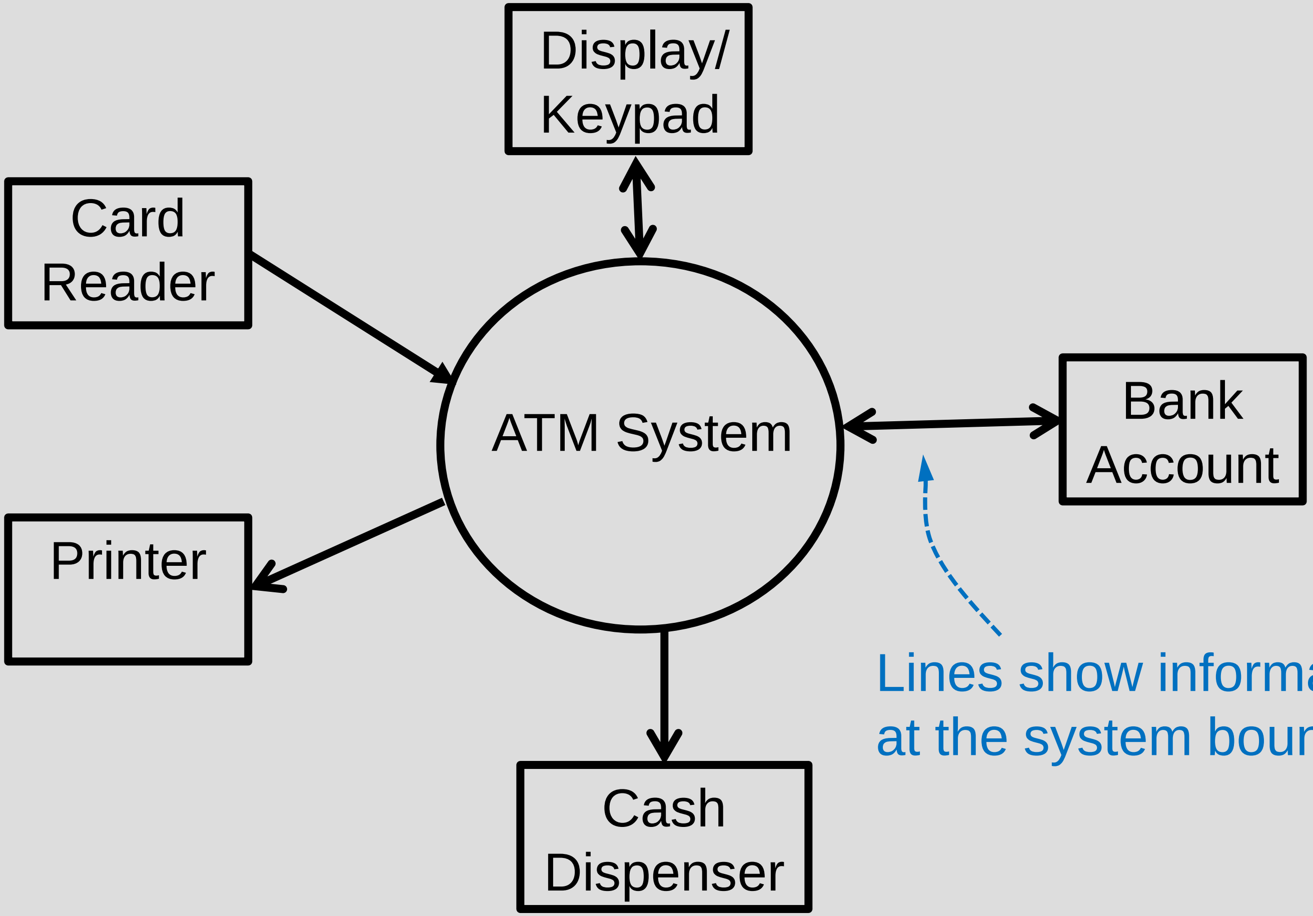
Revision 1.4, based on UML 2.0

Using UML to Represent Software Architecture

- **UML is recommended** notation but…

- **Many** notation **variations** to choose from
- **No** one set of **prescribed** choices
- Select notations that **best fit** what needs to be communicated
- Keep it **simple**
- The following are recommendations

Start with Context Diagram for “Big Picture”



Lines show information flow at the system boundaries

Three Broad Groups of Architectural Decisions

- Address three broad types of **architectural decisions**
  - **Module structures**
    - What are the **static functional code units?**
  - **Component-and-connector structures**
    - What are the **replaceable, distributable, runtime computational elements** that encapsulate module behavior behind **interfaces**?
  - **Allocation structures**
    - What are runtime software artifacts and where are they located in **non-software environmental structures?**

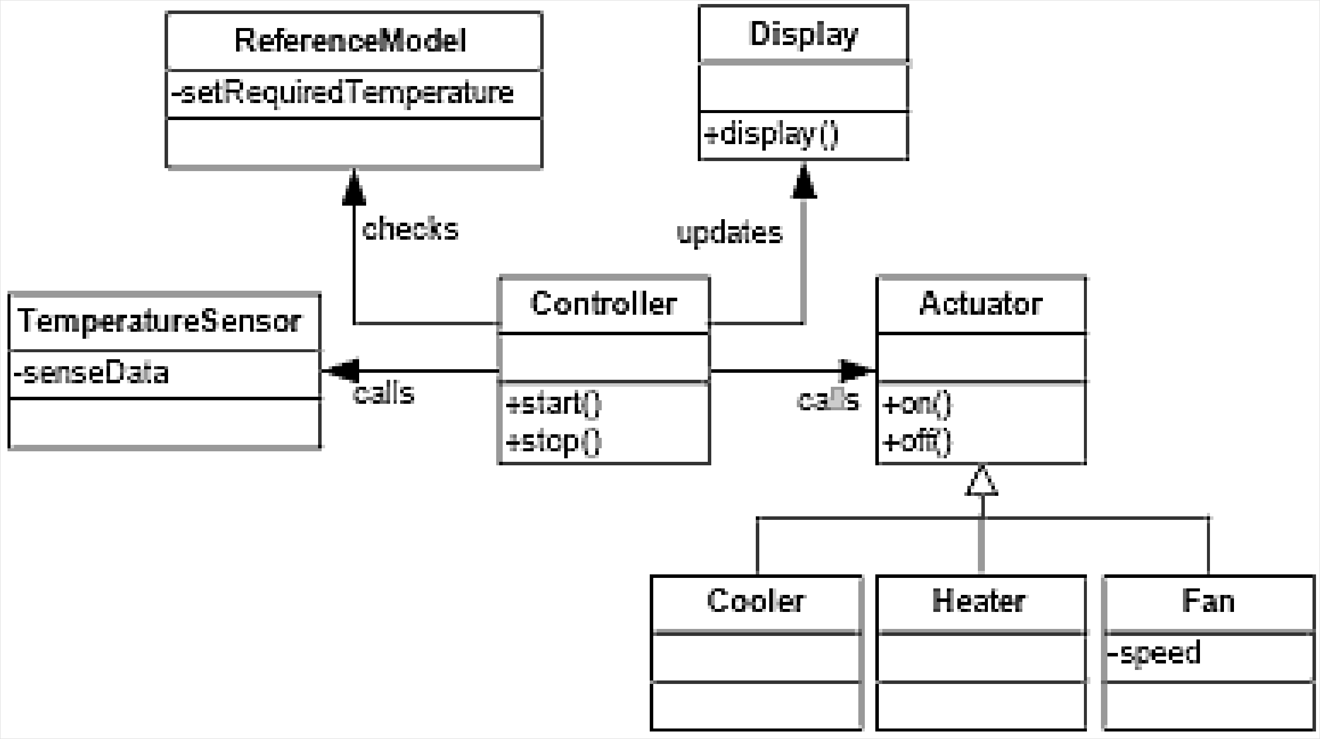
Module Structure Views

- Elements - modules, **implementation units** of software that provide a **coherent set of responsibilities**
- Relations
  - **Object oriented**
    - **Is part of**, a part/whole relationship
    - **Depends on**, a dependency relationship between two modules
    - **Is a**, a generalization/specialization relationship
  - **Layered** – aggregation of modules into layers

UML: Package and class diagrams

Module View Example

Climate control system in vehicles



Usage of Module Views

- Static **functional decomposition**
- System **information architecture**
- Supports the definition of **work assignments, development process and schedules**
  - Blueprint for coding and testing
  - Change-impact analysis
  - Requirements traceability analysis

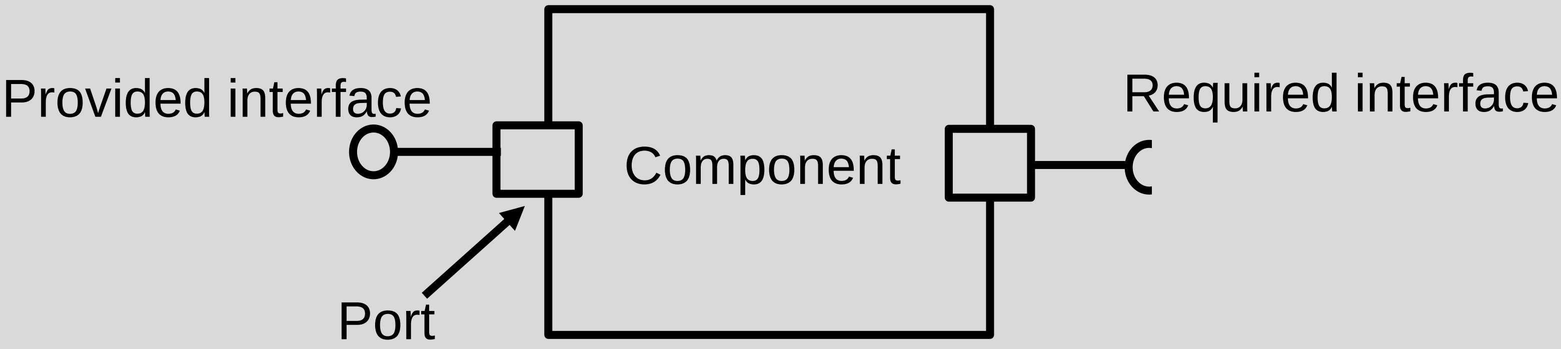
"It is unlikely that the documentation of any software architecture can be complete without at least one module view."

Component and Connector Structure Views

- Elements
  - **Components – encapsulated and replaceable** system elements that have **runtime behavior**
  - **Connectors** - pathways of **interaction between components**.
- Relations (in UML notation)
  - Components have **ports** with associated **connector roles**
  - **Ports have associated interfaces**
  - **Relations represented** as a **graph** of components and connector attachments.
    - E.g., client – server invoke-services role
  - **Interface delegation** - **component ports** may be **associated** with one or more **"internal" ports**

UML: Class, Package, and/or Component diagrams

Component and Connector UML Notation

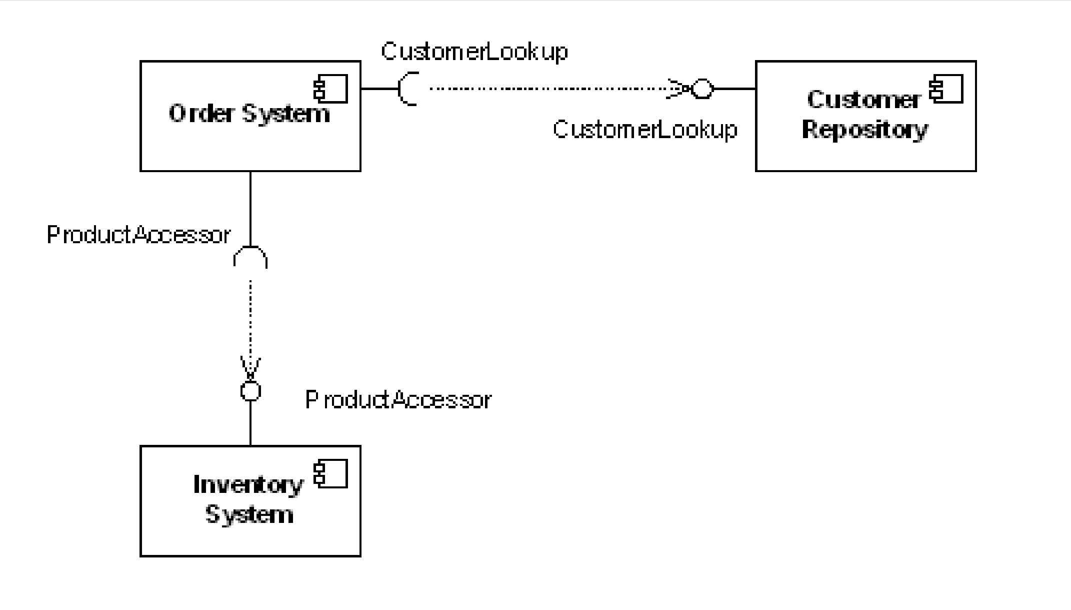


Provided interface

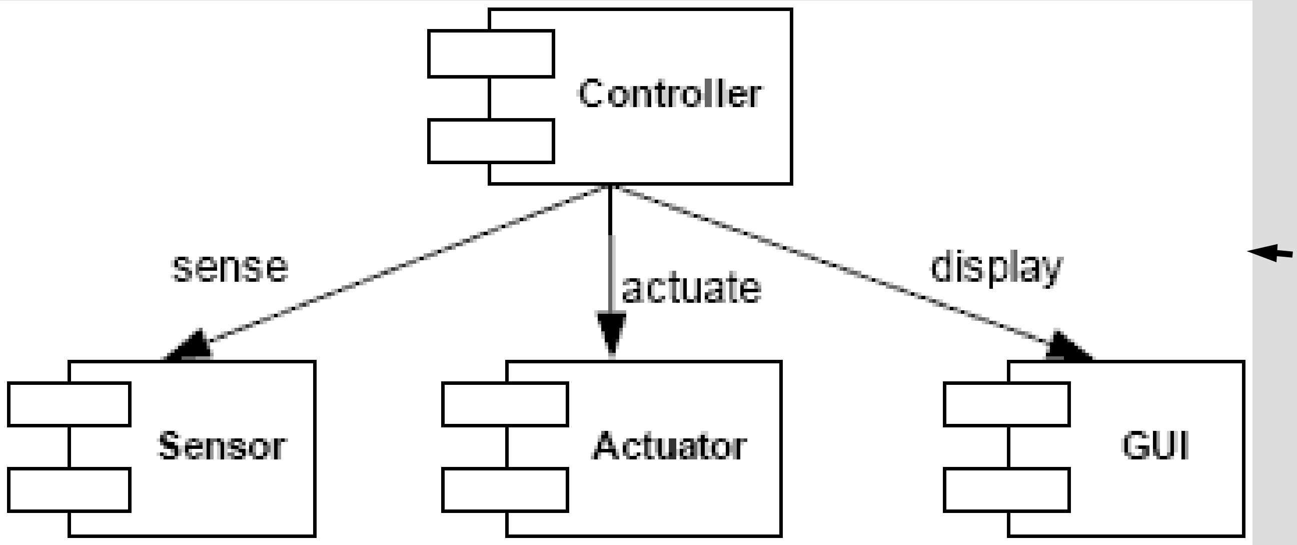
Component

Required interface

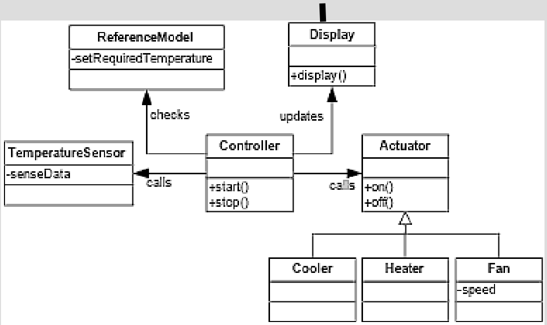
Port



Component-and-Connector View Example



(Can show simplified relationships)



C & C Views – Constraints and Usage

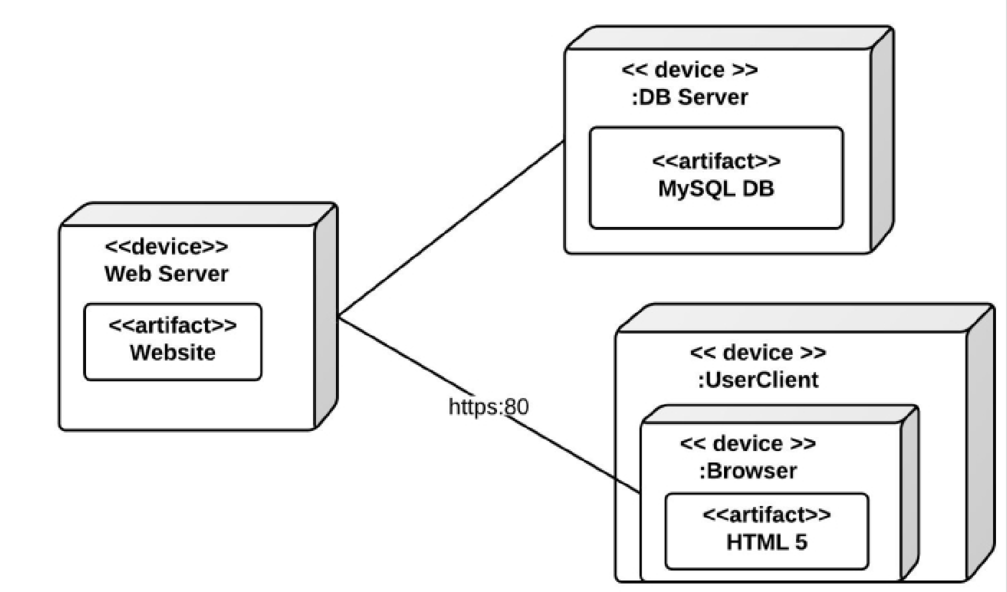
- Usage
  - Major **executing components**
  - Major **shared data stores**
  - **Runtime interaction**; e.g., control and data flow, parallelism
  - **Connector mechanisms** – e.g., service invocation, asynchronous messaging, event subscription, …
- Constraints
  - All attachments are only between components and connectors
  - Attachments must be between compatible ports and roles

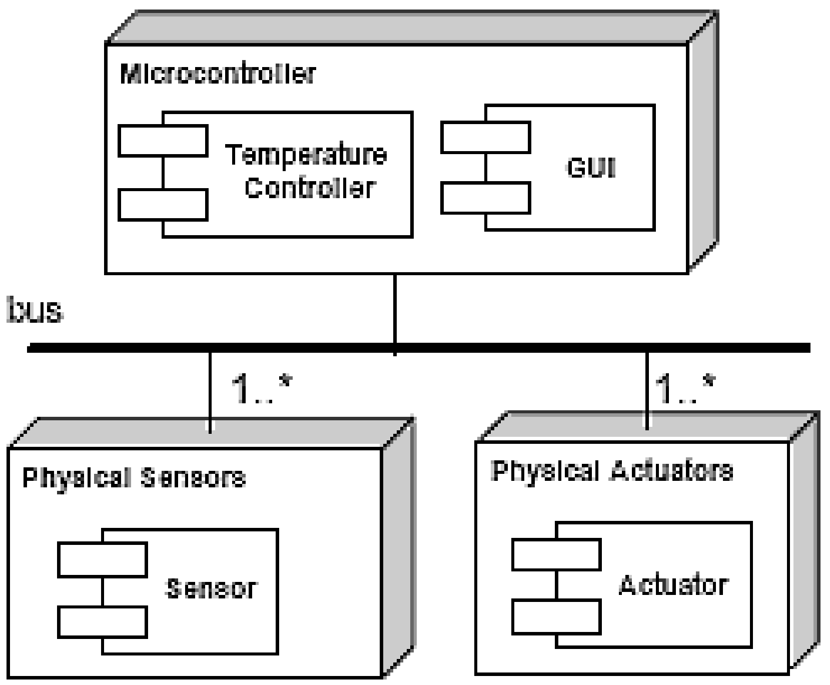
Allocation Views

- Elements
  - **Software element**
    - Some runtime packaging of logical modules and components (e.g., processes)
  - **Environmental element** - **execution** (hardware, runtime operation) or **development** (file structure, deployment, development organization)
    - Properties that are provided to the software; e.g., bandwidth
- Relations
  - **Allocated to** - a **software element** is mapped (allocated to) an **environmental element**
  - Static or dynamic (e.g., resource allocation)

UML: Deployment diagrams

Allocation View Example





Usage of Allocation Views

- Specify **structure and behavior of runtime elements** such as processes, objects, servers, data stores
- Reasoning and decisions about …
  - What hardware and software is needed
  - Distributed development and allocation of work to teams.
  - Builds, integration testing, version control
  - System installation

Augment with “Quality” Views

- **More specific views** may be needed for **specific stakeholders** or to address **specific concerns**
- The solution may be **cross cutting** across **multiple structural views**
  - By analogy – plumbing or electrical systems for buildings
- A quality view **extracts** relevant **pieces of structural views** and **packages** them together
  - E.g., show just those components that have a role in satisfying security requirements

Relating Structures to Each Other

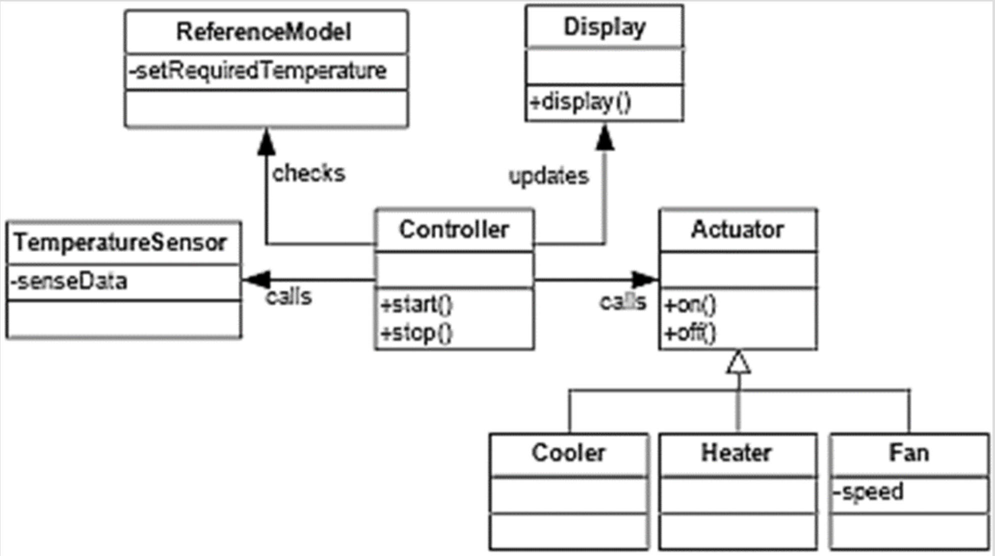
- **Each structure** provides a **different perspective** and design handle on a system
  - Each is valid and useful on its own
- The structures are **not independent**, just the opposite
  - Elements of one will be related to elements of another
- **Relationships** should be **consistent and rational**

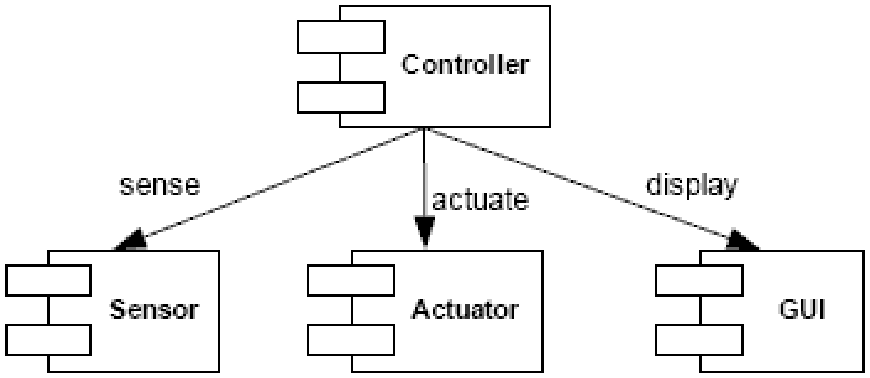
Element names: meaningful and consistent across views!!

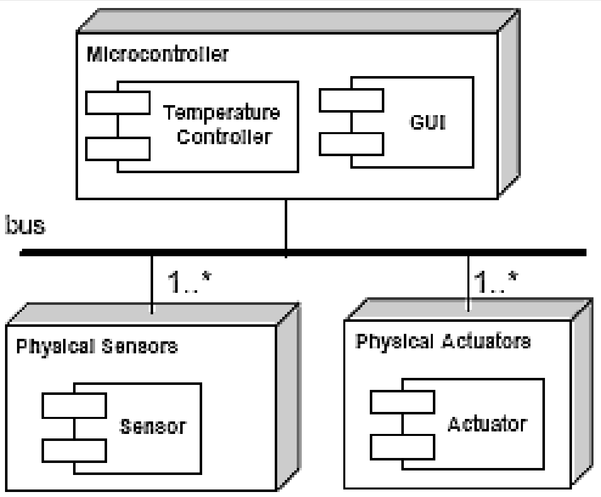
Relating Structures to Each Other

- Example: a code module in a decomposition structure may map to one, part of one, or several run-time components in a component-and-connector structure
- **Sometimes, one structure dominates** (usually decomposition structure)
- For some systems, **some structures** may be **irrelevant or trivial**, such as a single node, single process application

Relating Structures to Each Other







Which Views? The Ones You Need!

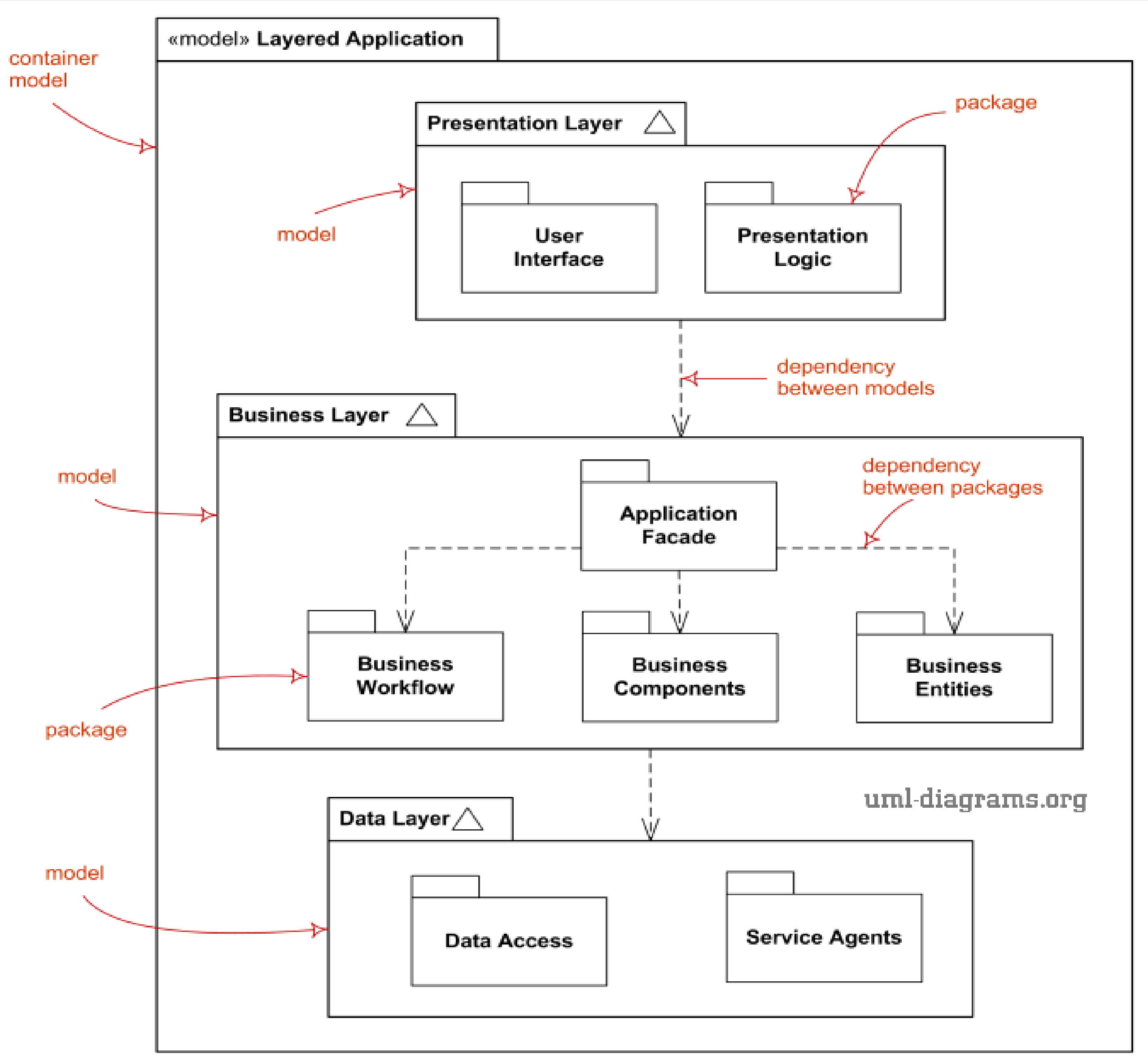
- **Different views** support **different goals and uses**
- The **views** you document **depend** on the **stakeholders** and **uses** of the documentation.
- Each view has **a cost and a benefit**; the benefits of maintaining a view should outweigh its costs
- At a minimum, at least on module view and one component and connector view

Supplemental Material

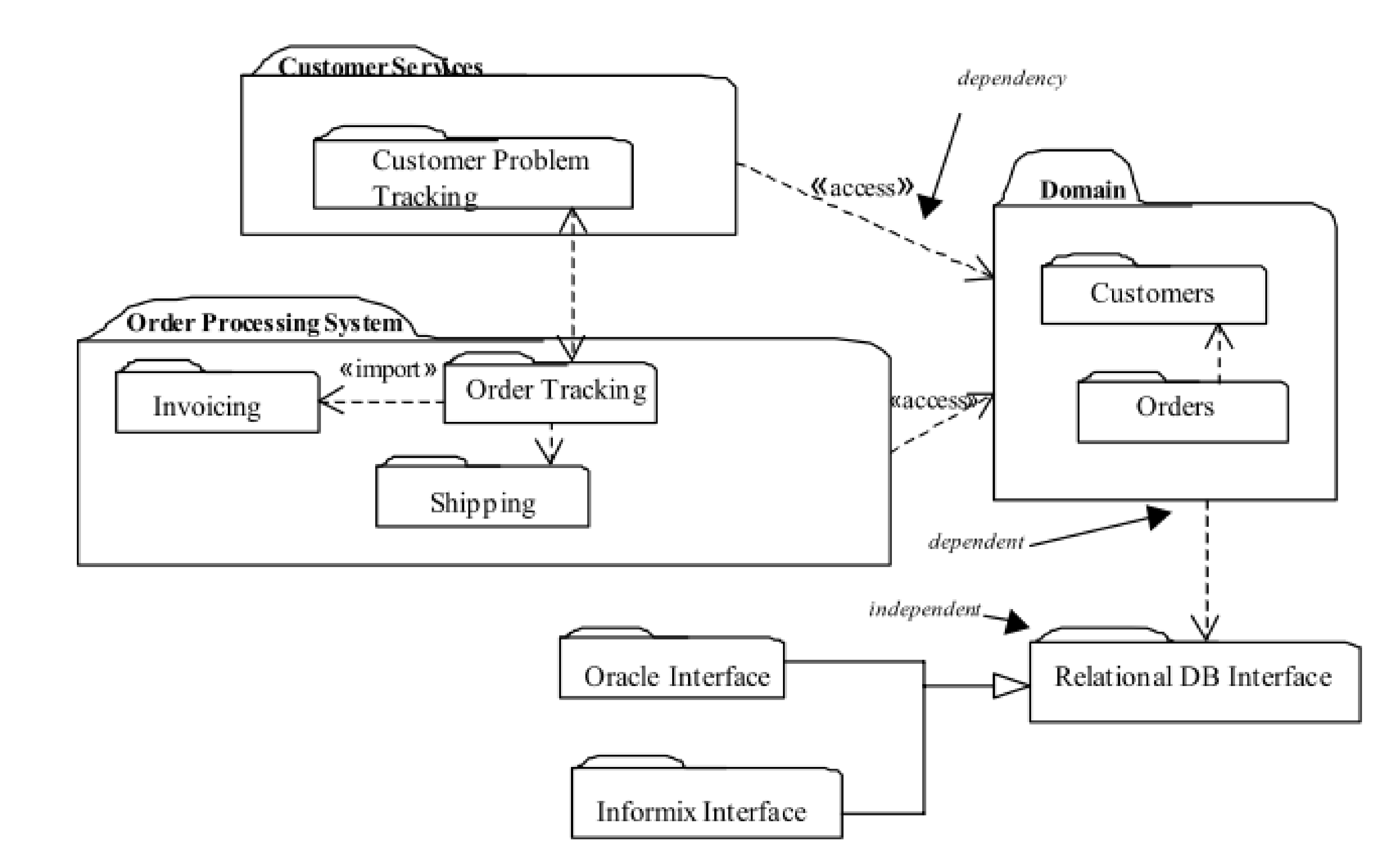
Examples of Views

Module View Example

UML Module Diagram

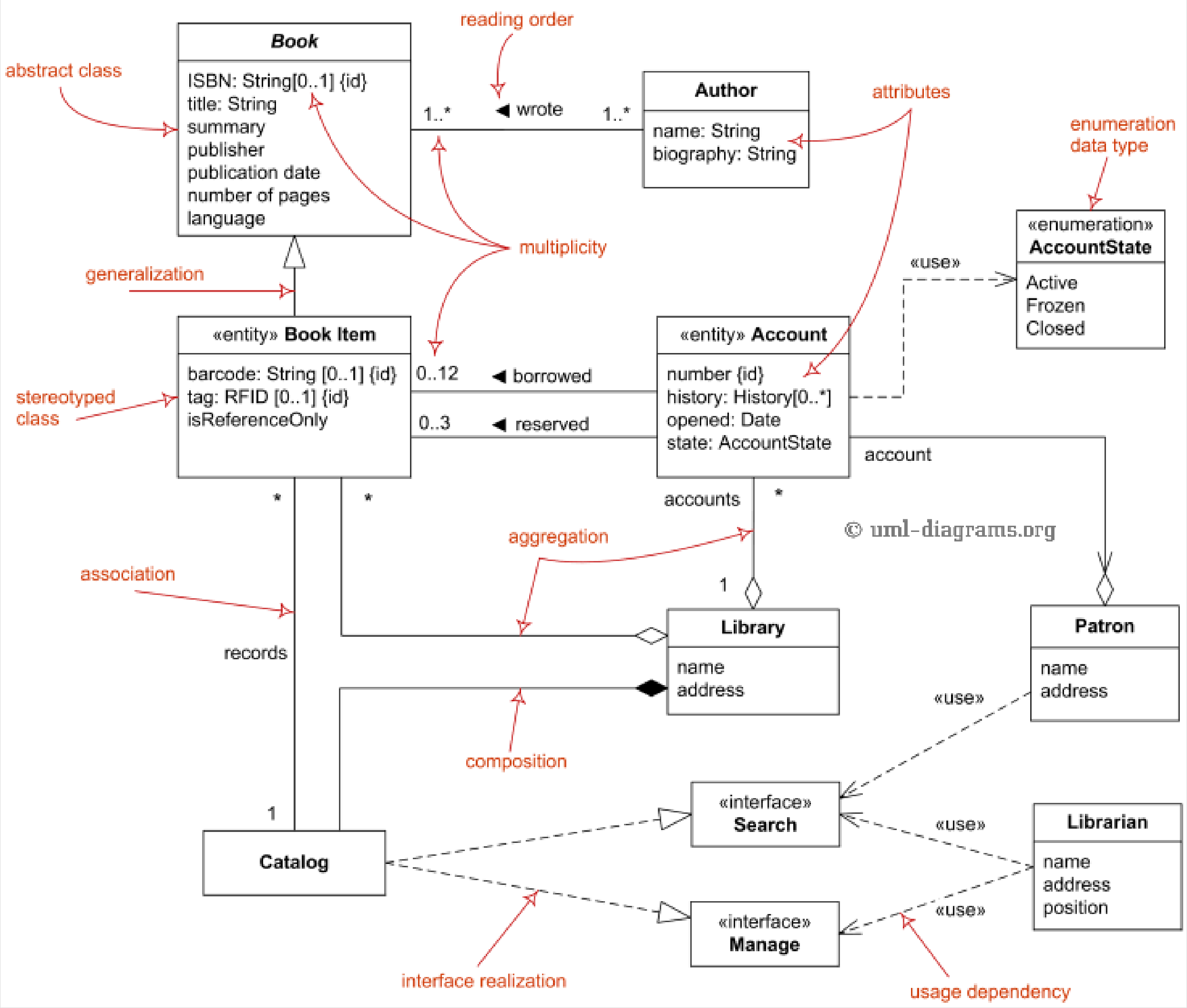


Module View Example



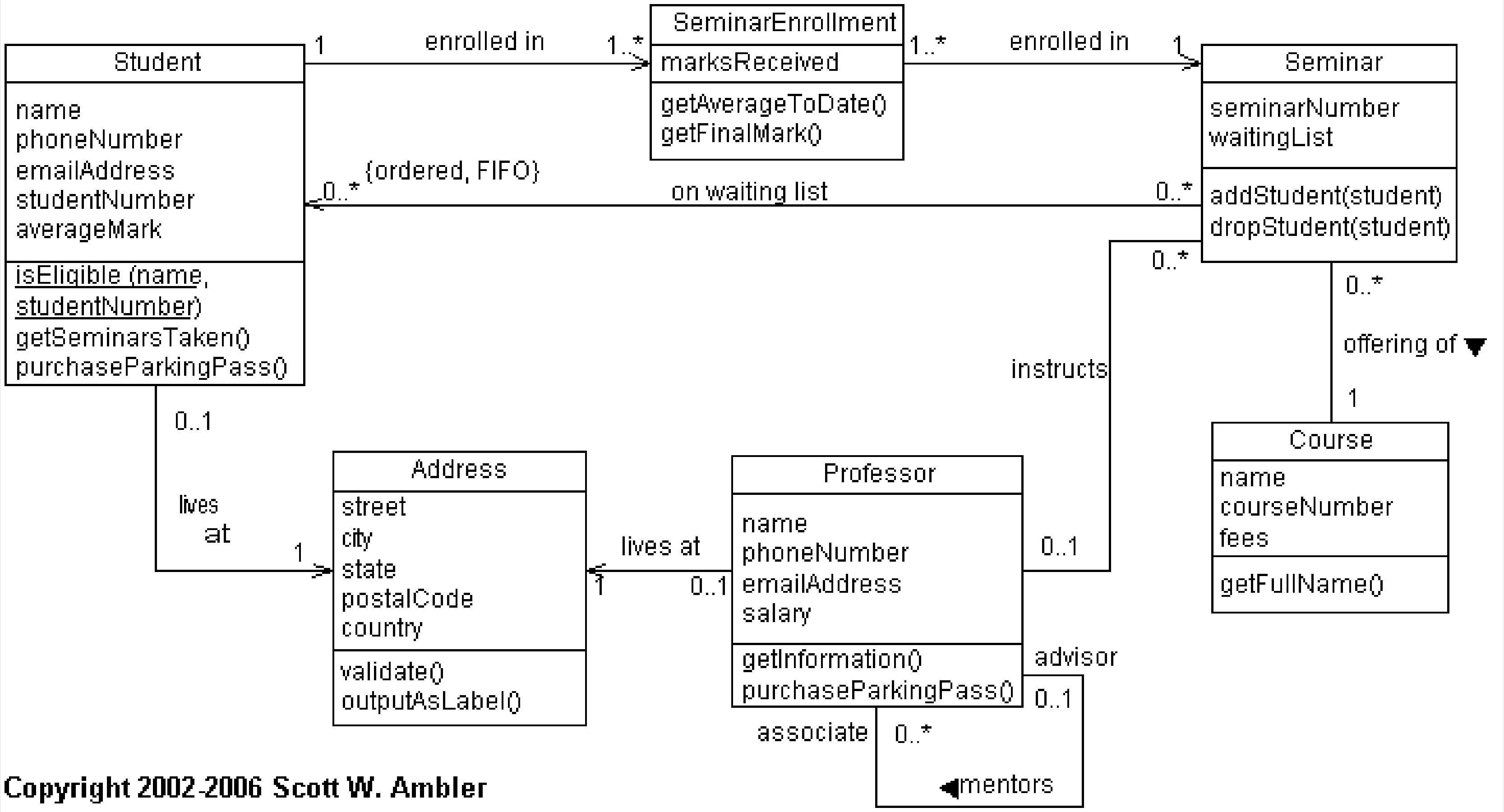
Module View Example

UML Domain Model Class Diagram



Module View Example

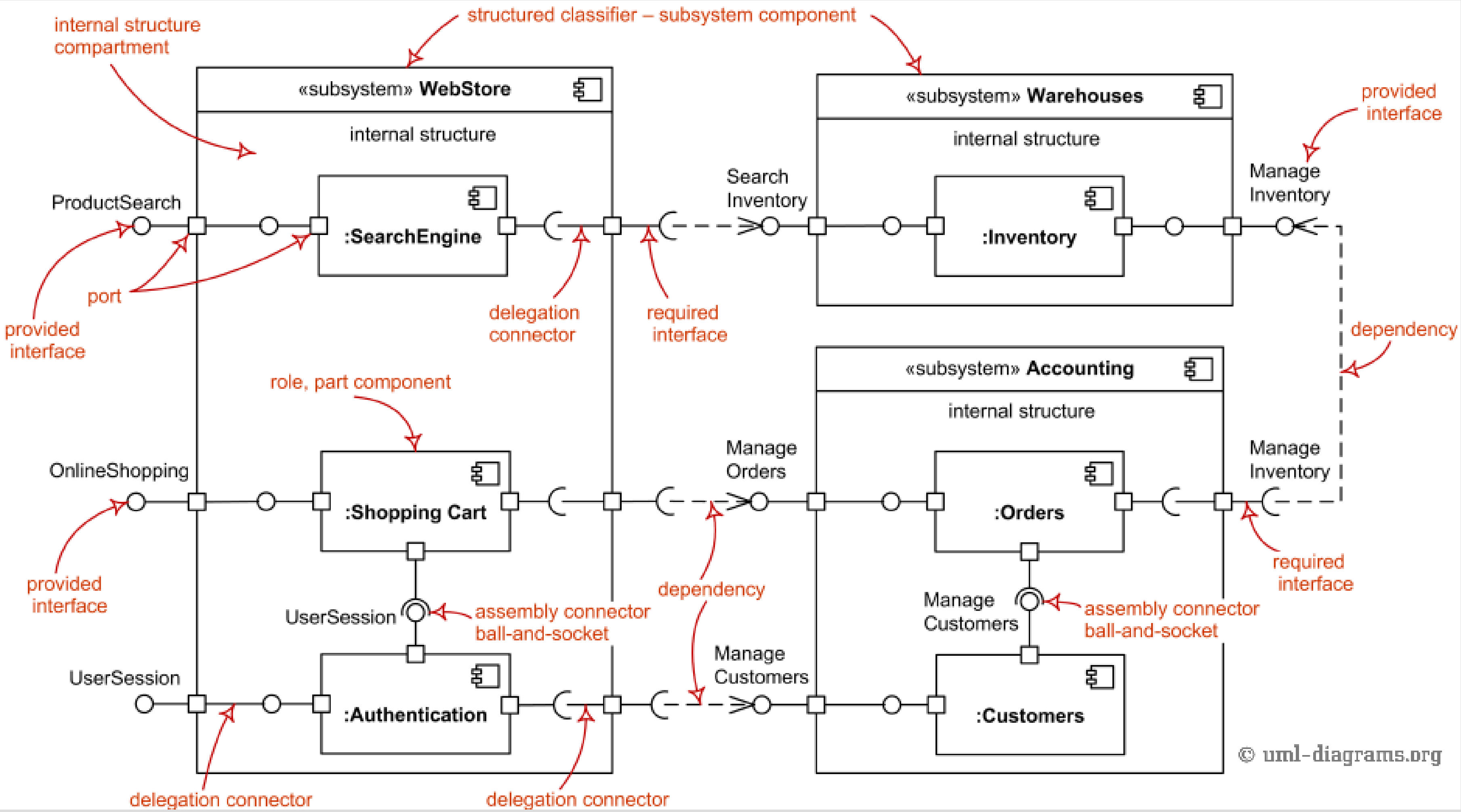
UML Class Diagram



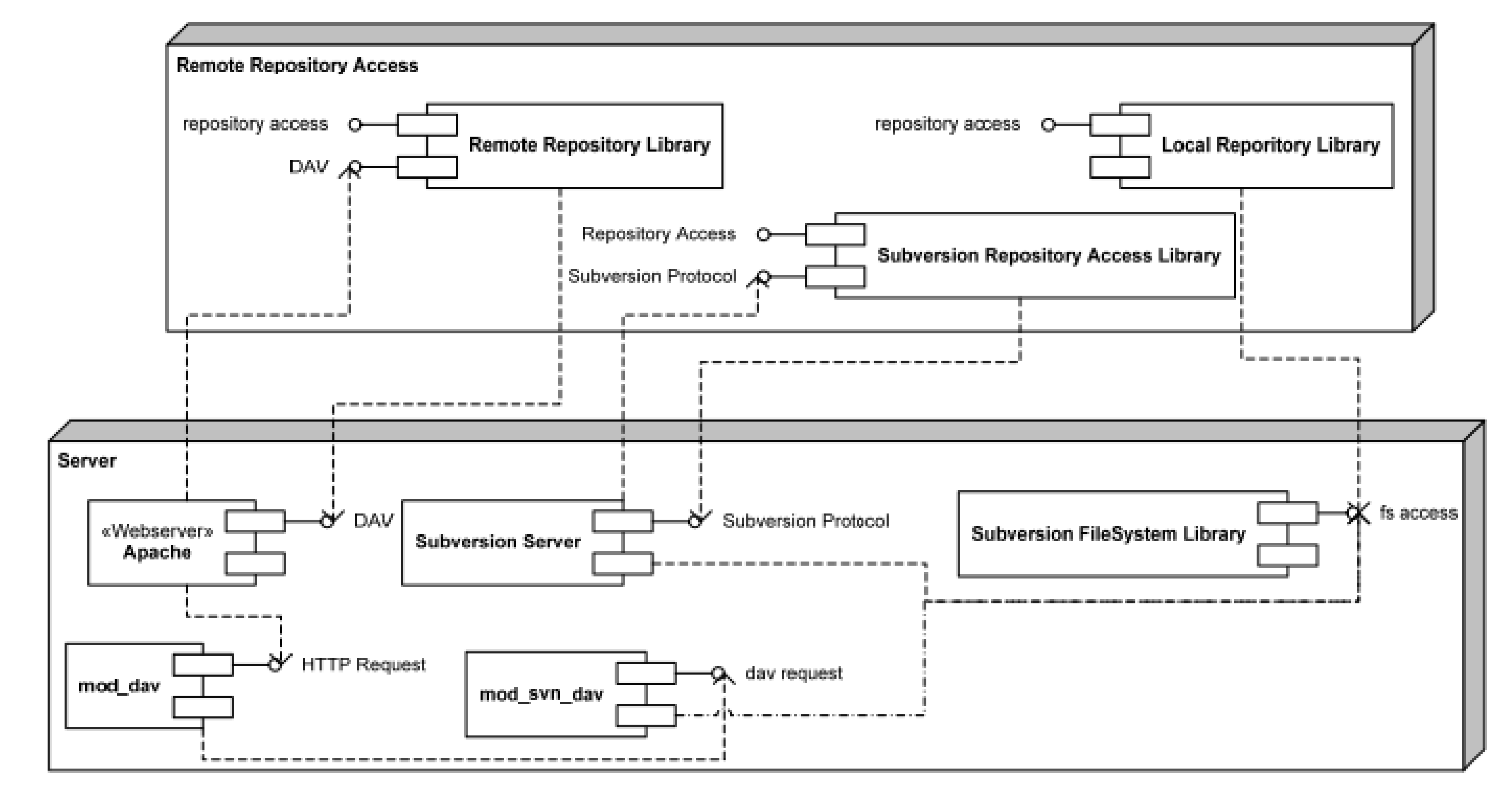
Copyright 2002-2006 Scott W. Ambler

Component-and-Connector

UML Component Diagram

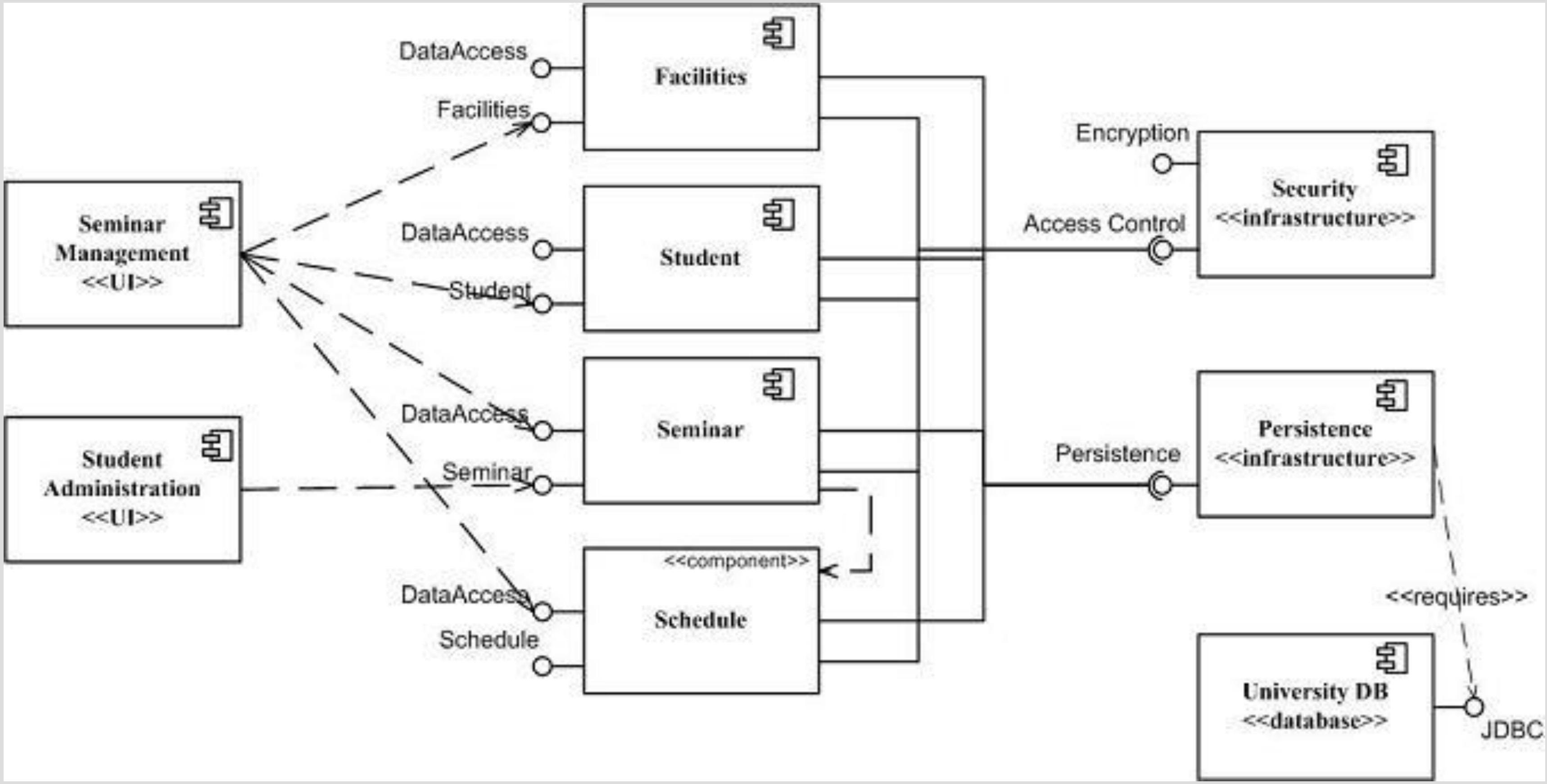


Component-and-Connector
Client Server View Example

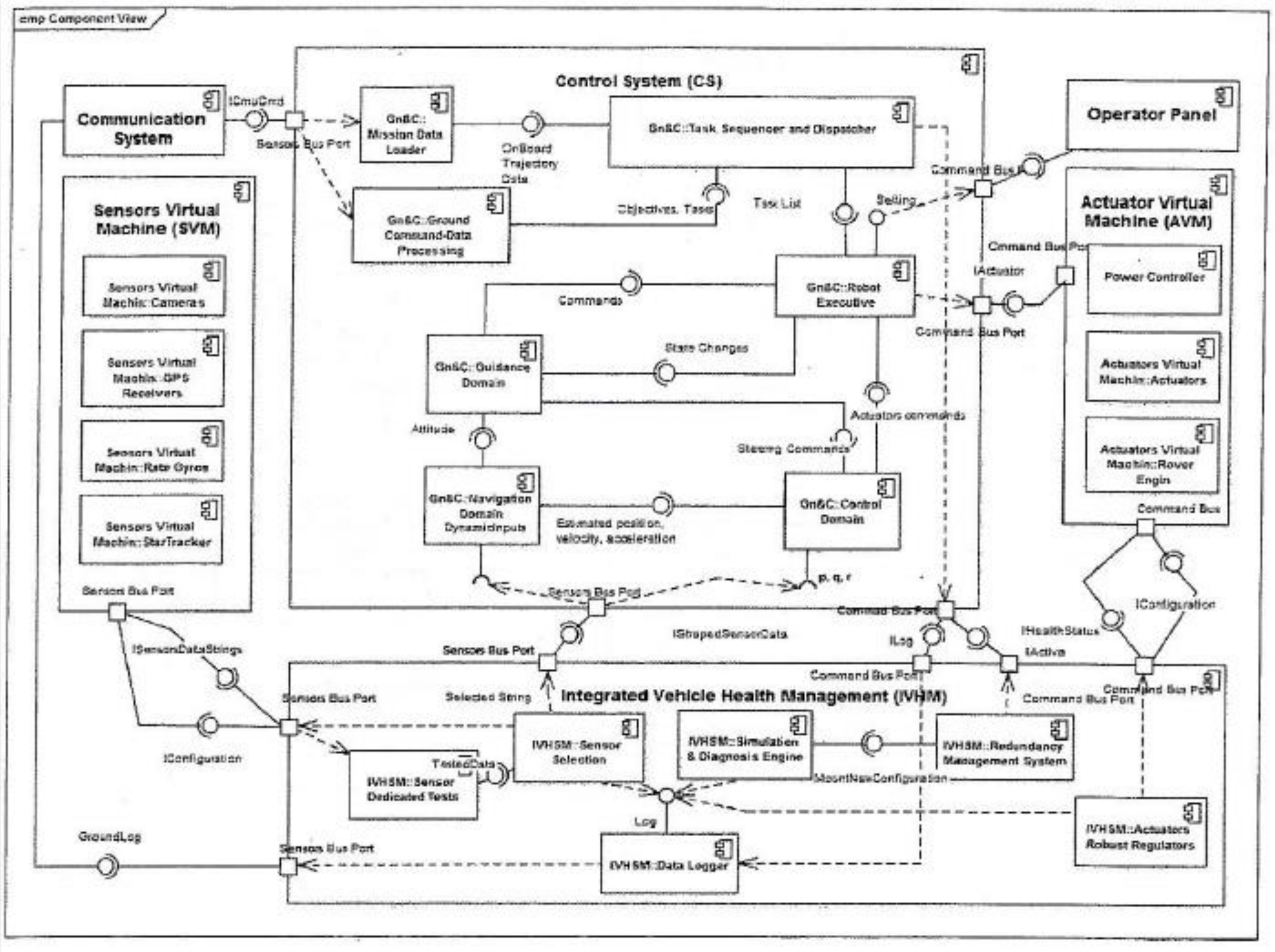


Component-and-Connector

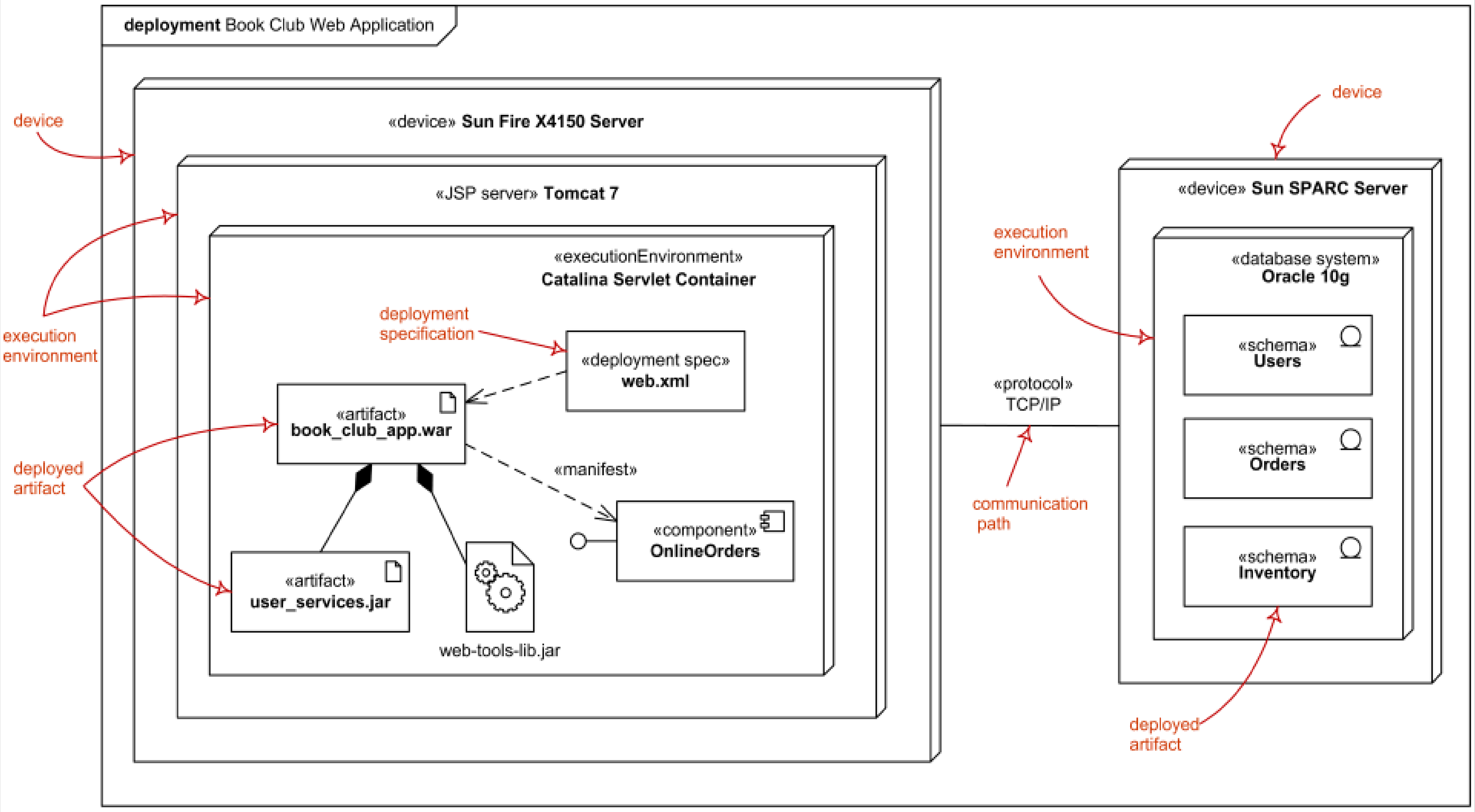
Another Example



Component-Connector
Embedded Example

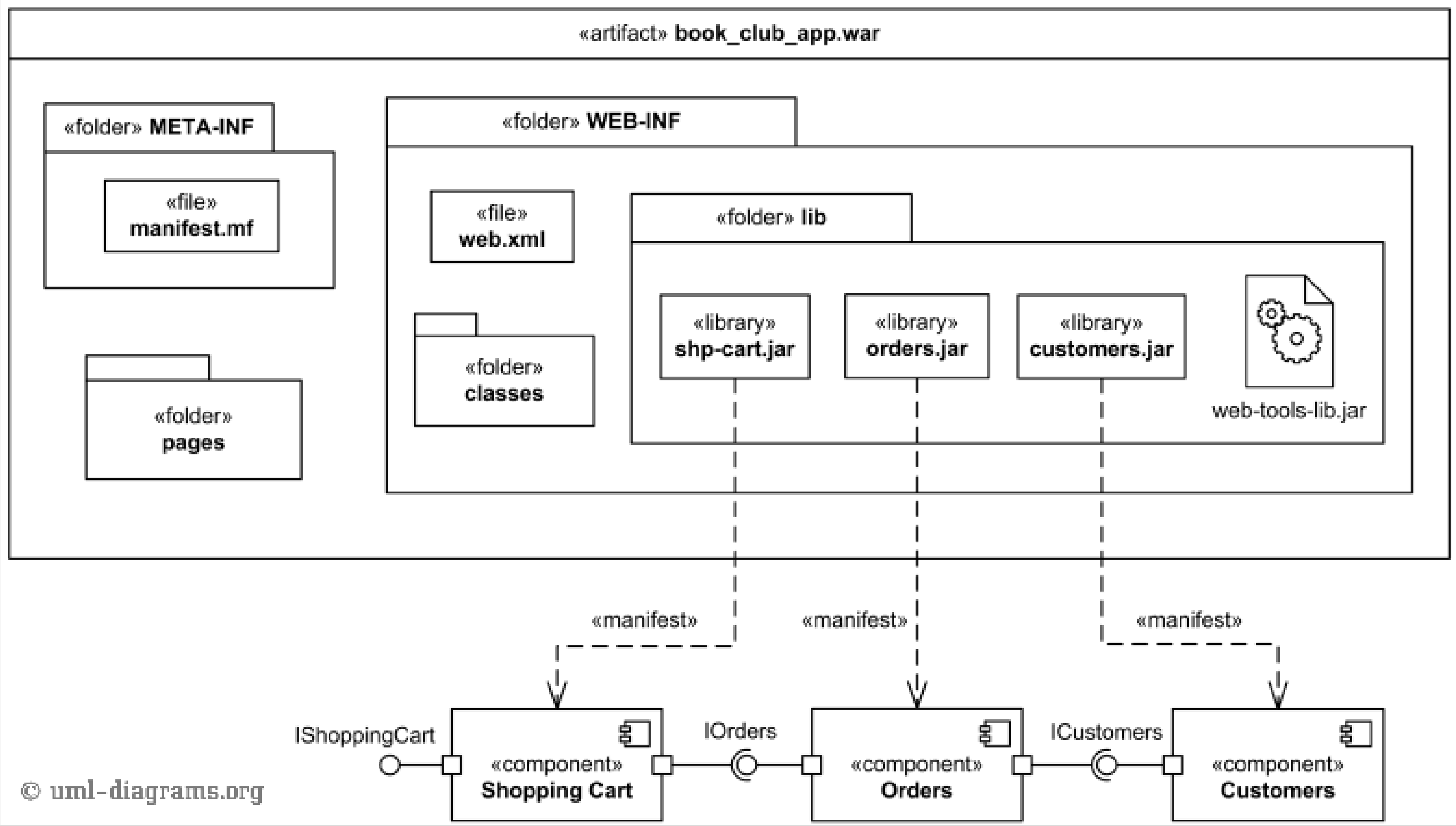


Allocation View
UML Deployment Diagram Example



Allocation View

UML Implementation Diagram


«artifact» book_club_app.war
«folder» META-INF
«file» manifest.mf
«folder» pages
«folder» WEB-INF
«file» web.xml
«folder» classes
«folder» lib
«library» shp-cart.jar
«library» orders.jar
«library» customers.jar
web-tools-lib.jar
«manifest»
«manifest»
«manifest»
IShoppingCart
«component» Shopping Cart
IOrders
«component» Orders
ICustomers
«component» Customers
© uml-diagrams.org