Architectural Styles

Modified from Bernd Bruegge and Allen H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition, Pearson, 2013.

System Design Concepts: Layers and Partitions

- A hierarchical decomposition of a system yields an ordered set of layers
- A **layer** is a grouping of subsystems providing related services
- Layers are ordered in that each layer can depend only on lower level layers and has no knowledge of the layers above it
- In a **closed architecture**, each layer can access only the layer immediately below it
- In an **open architecture**, a layer can also access layers at deeper levels
- **Partitions** vertically divide a system into several independent (or weakly-coupled) subsystems that provide services on the same level of abstraction

Figure 6-9: Subsystem Decomposition of a System into 3 Layers

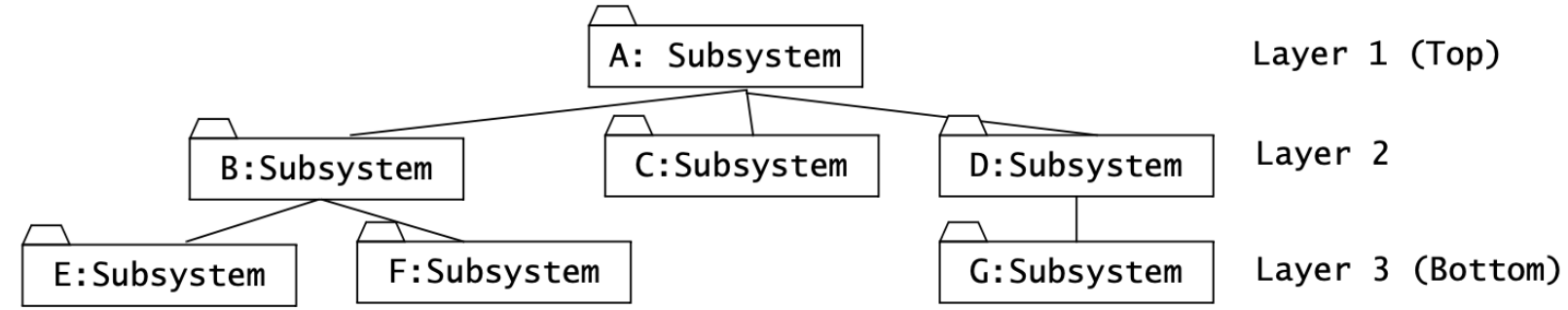

Figure 6-9 Subsystem decomposition of a system into three layers (UML object diagram, layers depicted as packages). A subset from a layered decomposition that includes at least one subsystem from each layer is called a vertical slice. For example, the subsystems A, B, and E constitute a vertical slice, whereas the subsystems D and G do not.

Example: Open Systems Interconnection model

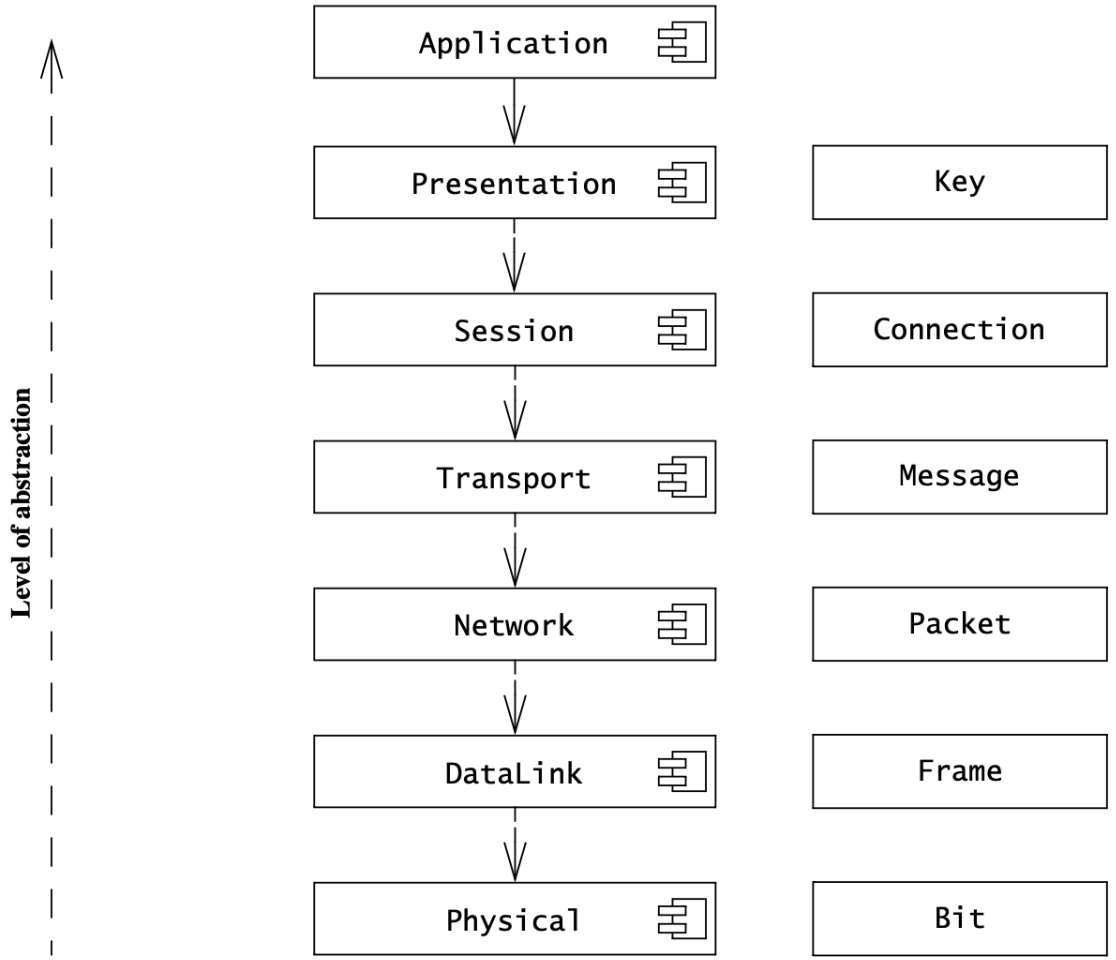

Figure 6-10 An example of closed architecture: the OSI model (UML component diagram). The OSI model decomposes network services into seven layers, each responsible for a different level of abstraction.

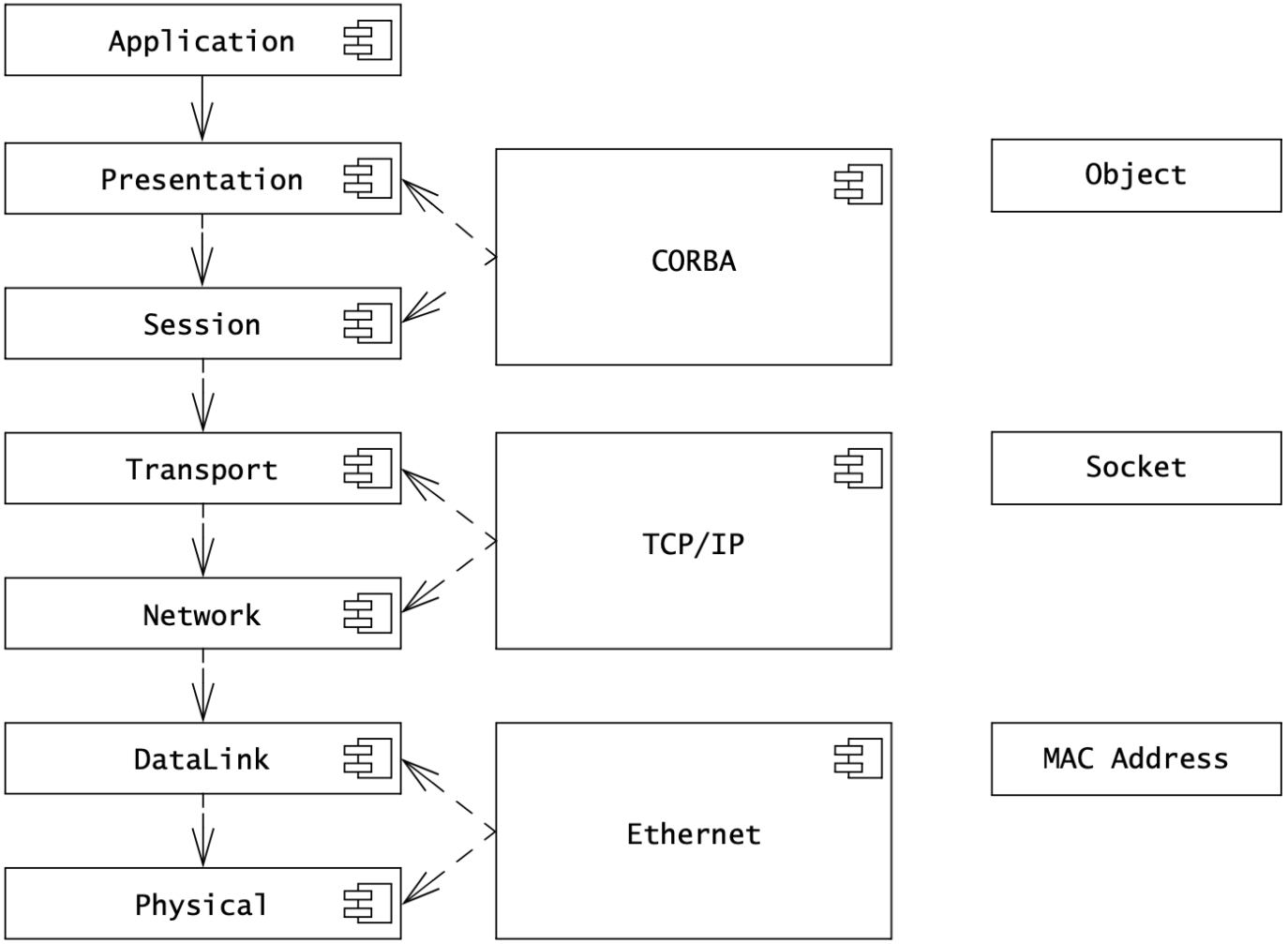

Figure 6-11 An example of closed architecture (UML component diagram). CORBA enables the access of objects implemented in different languages on different hosts. CORBA effectively implements the `Presentation` and `Session` layers of the OSI stack.

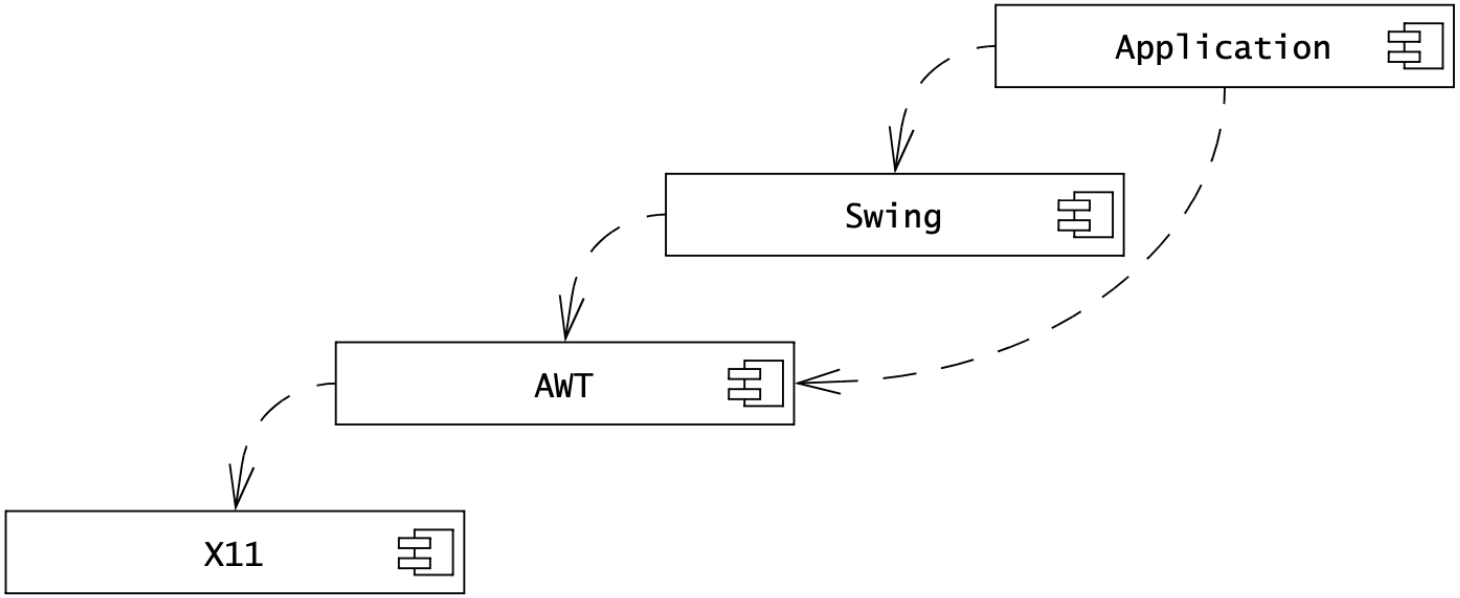

Figure 6-12 An example of open architecture: the `Swing` user interface library on an `X11` platform (UML component diagram). `X11` provides low-level drawing facilities. `AWT` is the low-level interface provided by Java to shield programmers from the window system. `Swing` provides a large number of sophisticated user interface objects. Some `Applications` often bypass the `Swing` layer.

- The Abstract Window Toolkit (AWT) is Java's original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing.

System Design Concepts: Architectural Styles Repository

- Subsystems access and modify data from a single data structure called the central repository
- Subsystems are loosely coupled (interact only through the repository)
- Control flow is dictated by central repository (triggers) or by the subsystems (locks, synchronization primitives)

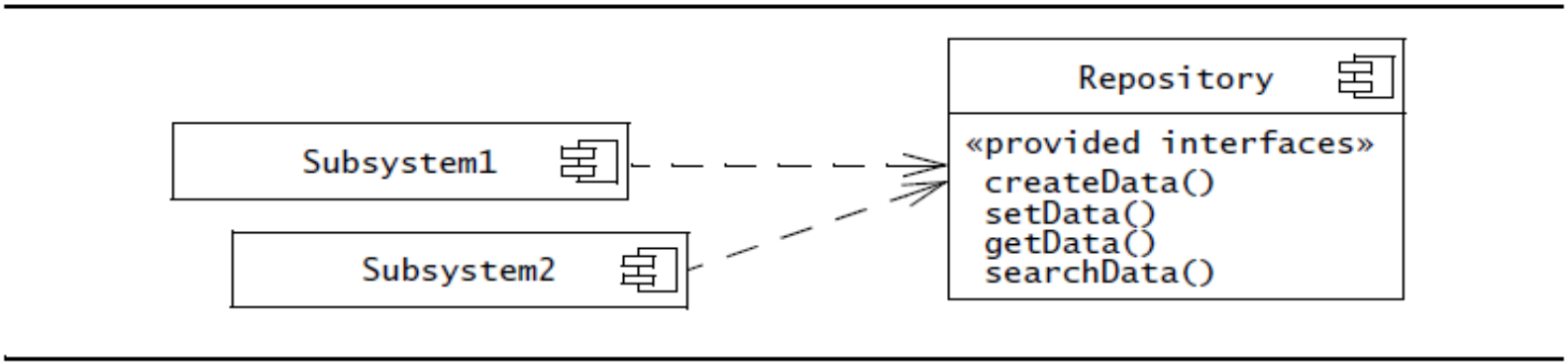

Figure 6-13 Repository architectural style (UML component diagram). Every Subsystem depends only on a central data structure called the Repository. The Repository has no knowledge of the other Subsystems.

System Design Concepts: Architectural Styles Repository (cont.)

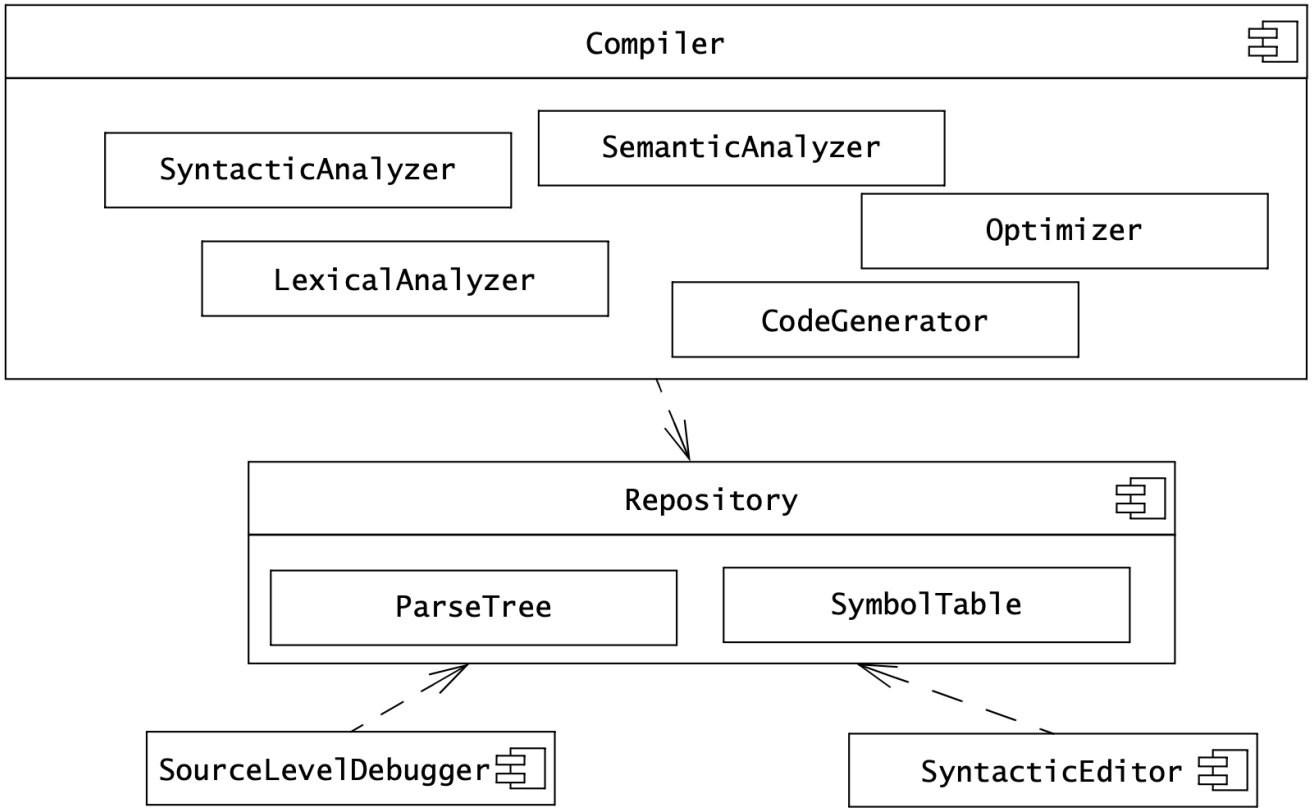

Figure 6-14 An instance of the repository architectural style (UML component diagram). A `Compiler` incrementally generates a `ParseTree` and a `SymbolTable` that can be used by `SourceLevelDebuggers` and `SyntaxEditors`.

System Design Concepts: Architectural Styles Model/View/Controller

- Subsystems are classified into 3 different types
  - **Model** subsystem: Responsible for application domain knowledge
  - **View** subsystem: Responsible for displaying application domain objects to the user
  - **Controller** subsystem: Responsible for sequence of interactions with the user and notifying views of changes in the model
- MVC is a special case of a repository architecture:
  - Model subsystem implements the central data structure, the Controller subsystem explicitly dictate the control flow

Figure 6-15: MVC Architectural Style

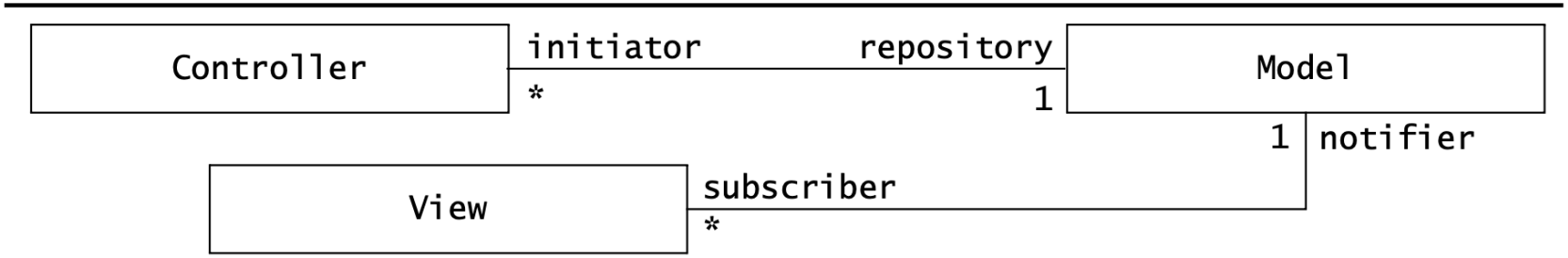

Figure 6-15 Model/View/Controller architectural style (UML class diagram). The `Controller` gathers input from the user and sends messages to the `Model`. The `Model` maintains the central data structure. The `Views` display the `Model` and are notified (via a subscribe/notify protocol) whenever the `Model` is changed.

Figure 6-16: Example of a File System Based on the MVC Architectural Style

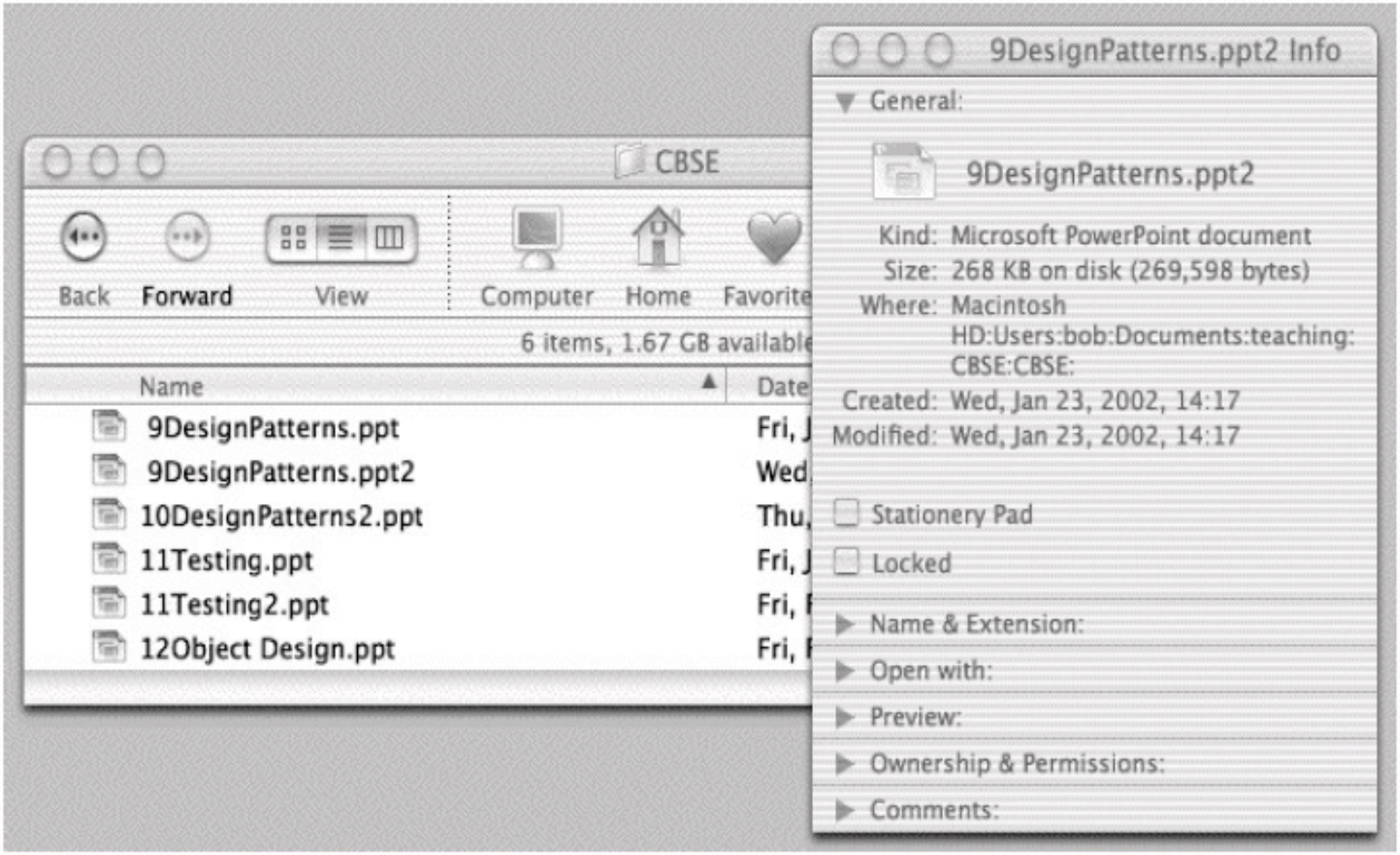

Figure 6-17: Sequence of Events in the MVC

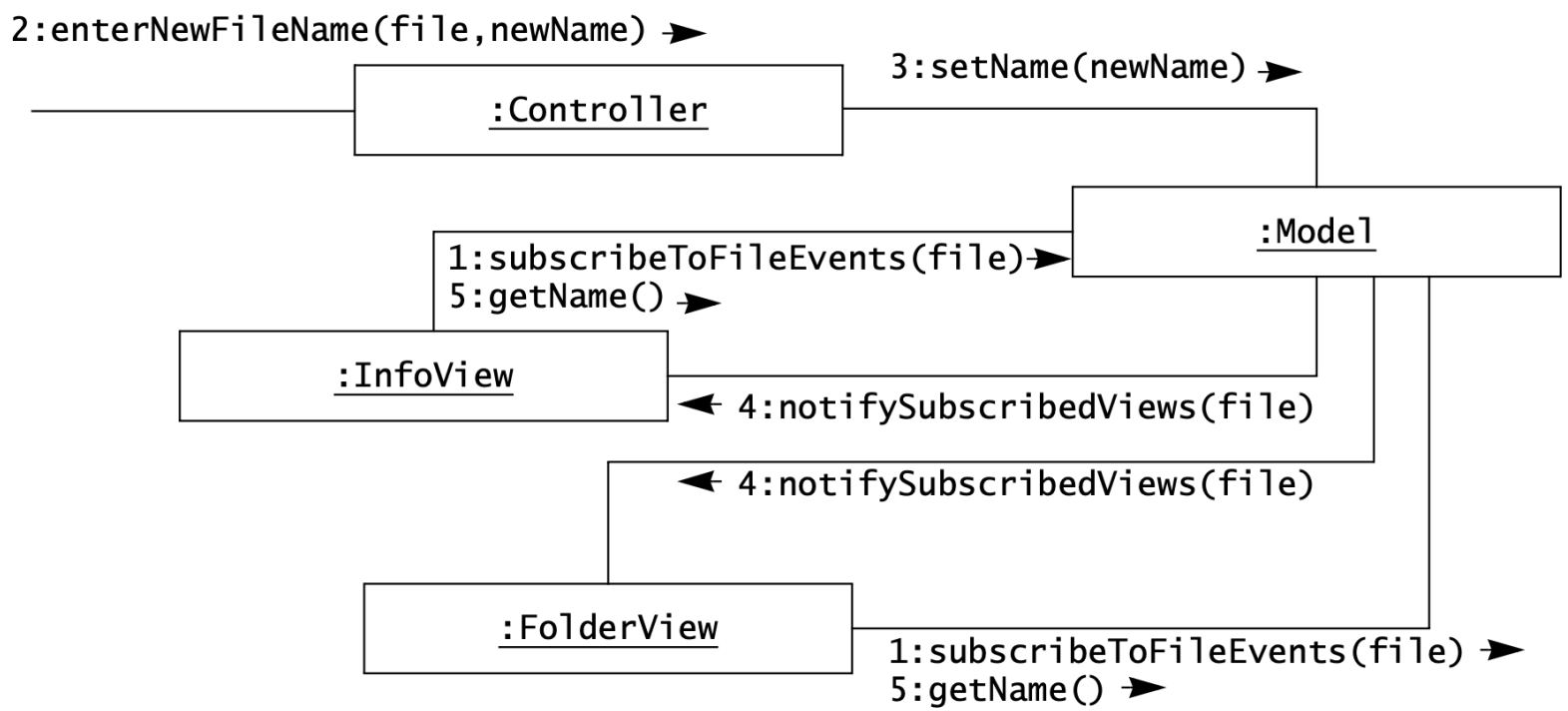

Figure 6-17 Sequence of events in the Model/View/Control architectural style (UML communication diagram).

System Design Concepts: Architectural Styles - Client/Server

- One or many servers provides services to instances of subsystems, called clients.
- Client calls on the server, which performs some service and returns the result
  - Client knows the interface of the server (its service)
  - Server does not need to know the interface of the client
- Response in general immediately
- Users interact only with the client

System Design Concepts: Architectural Styles - Client/Server

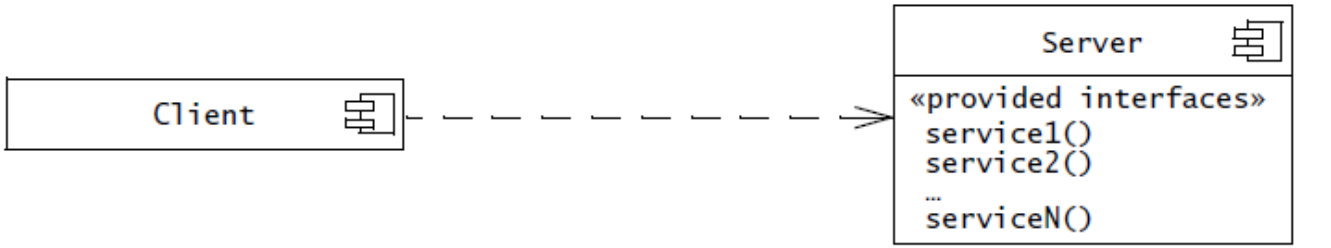

Figure 6-18 Client/server architectural style (UML component diagram). `Clients` request services from one or more `Servers`. The `Server` has no knowledge of the `Client`. The client/server architectural style is a specialization of the repository architectural style.

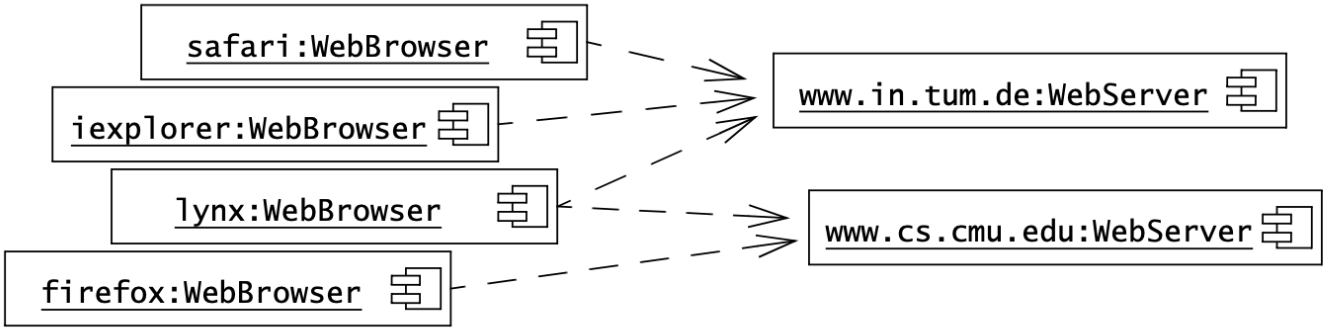

Figure 6-19 The Web as an instance of the client/server architectural style (UML deployment diagram).

System Design Concepts: Architectural Styles Peer-to-peer

- Generalization of Client/Server Architecture
- Subsystems can be servers and servers can be clients (each subsystem can request and provide services)
- More difficult because of possibility of deadlocks
- Example: a database that both accepts requests from the application and notifies to the application whenever certain data are changed

System Design Concepts: Architectural Styles
Peer-to-peer

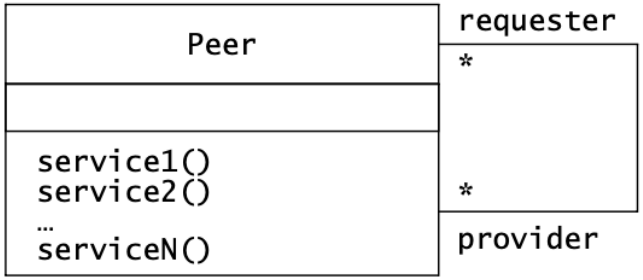

Figure 6-20 Peer-to-peer architectural style (UML class diagram). Peers can request services from and provide services to other peers.

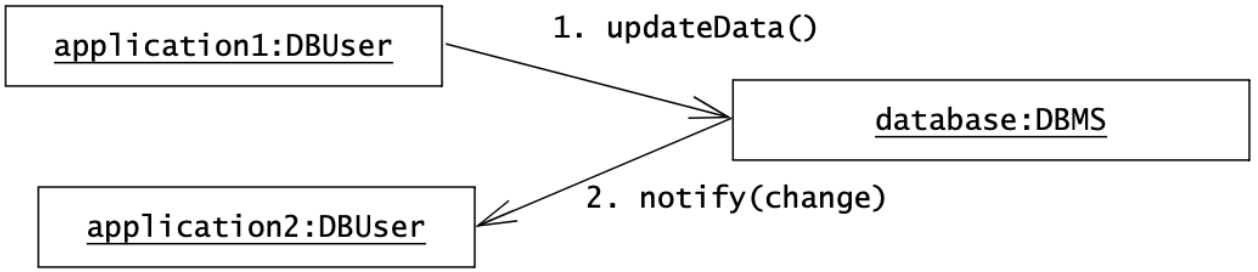

Figure 6-21 An example of peer-to-peer architectural style (UML communication diagram). The database server can both process requests from and send notifications to applications.

System Design Concepts: Architectural Styles Three-tier

- Subsystems are organized into 3 layers:
  - Interface layer: includes boundary objects that deal with the user, windows, forms, web pages
  - Application logic layer: includes control and entity objects
  - Storage layer: realizes the storage, retrieval, and query of persistent objects

Figure 6-22: Three-tier Architectural Style

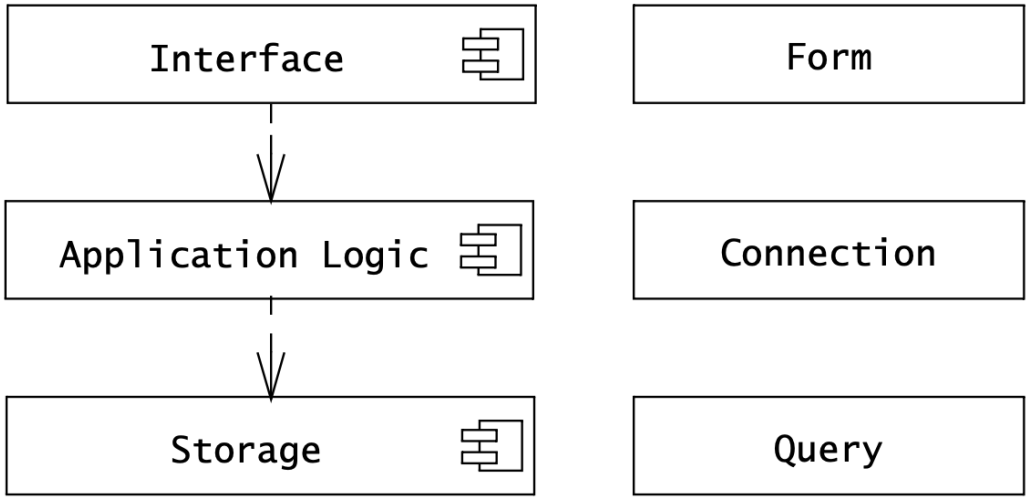

Figure 6-22 Three-tier architectural style (UML component diagram). Objects are organized into three layers realizing the user interface, the processing, and the storage.

System Design Concepts: Architectural Styles Four-tier

- Interface layer is decomposed into
  - presentation client layer: located on the user machines
  - presentation server layer: located on one or more servers

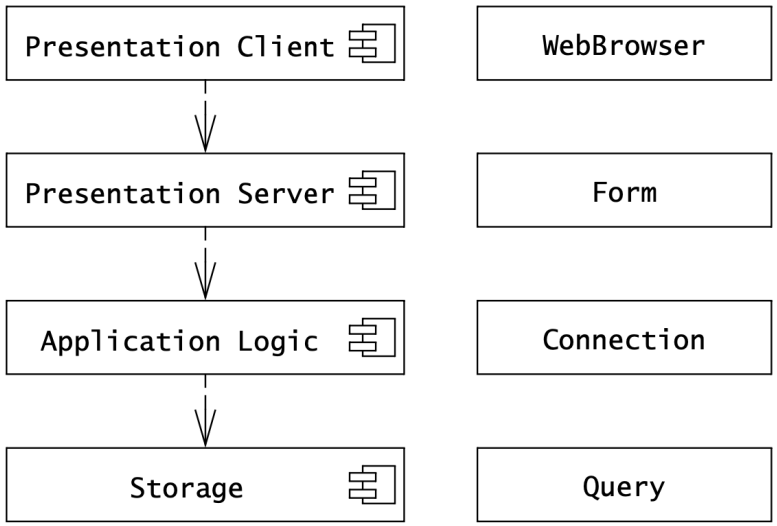

Figure 6-23 Four-tier architectural style (UML component diagram). The `Interface` layer of the three-tier style is split into two layers to enable more variability on the user interface style.

System Design Concepts:Architectural Styles Pipe and Filter

- Subsystems process data received from a set of inputs and send results to other subsystems via a set of outputs
- The subsystems are called “filters”, the associations between the subsystems are called “pipes”

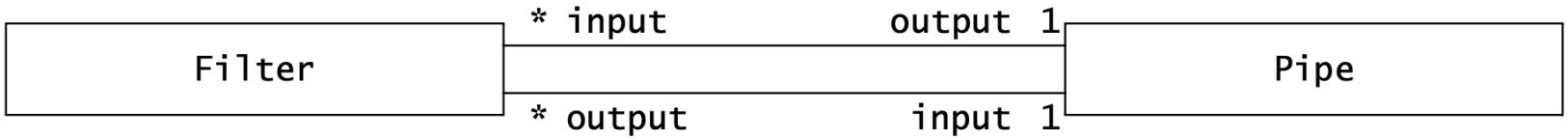

Figure 6-24 Pipe and filter architectural style (UML class diagram). A `Filter` can have many inputs and outputs. A `Pipe` connects one of the outputs of a `Filter` to one of the inputs of another `Filter`.

System Design Concepts:Architectural Styles

Pipe and Filter (cont.)

- Suited for systems that apply transformations to streams of data.
- Example: Unix Shell

```
% ps auxwww | grep dutoit | sort | more

dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```

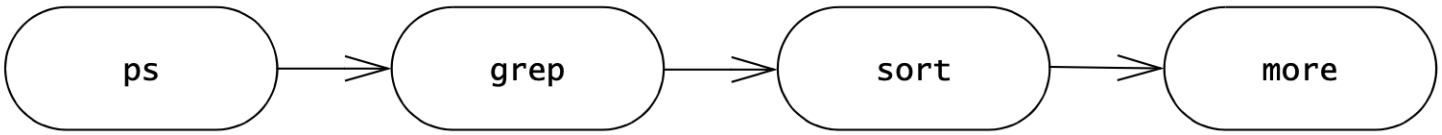

Figure 6-25 Unix command line as an instance of the pipe and filter style (UML activity diagram).