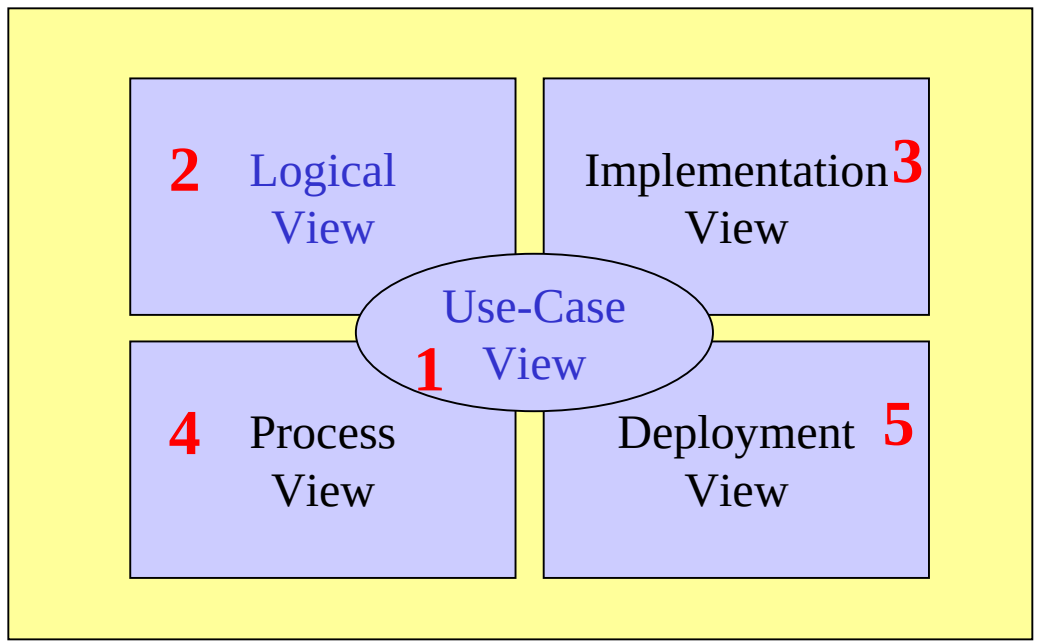
Chap 3. Architectural Views

Part 3.3 Implementation, Process, and Deployment Views

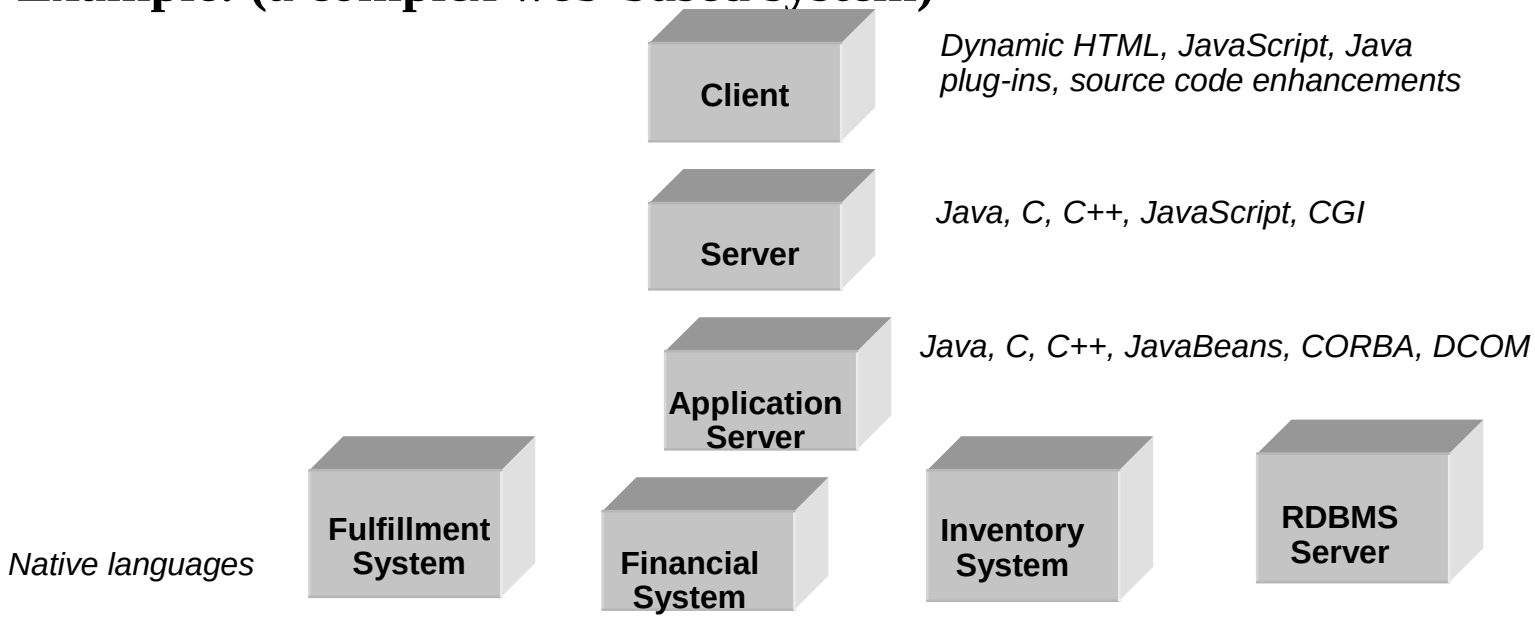
1. Motivation
2. Process View
3. Implementation View
4. Deployment View
5. ATM Example



1. Motivation

÷Complex software systems involve a wide range of functionality, deployed on independent processing nodes, involving a wide variety of languages, platforms, and technologies.

÷Example: (a complex web-based system)



The process, implementation, and deployment views capture this complexity by:

-Describing runtime entities: the threads and processes that form the system's concurrency and synchronization.

- Describing source and executable components, their organization, and their dependencies.

- Describing hardware topology and mapping software components to processing nodes

-Describing build procedures

2. Process View

Derives from the Logical view the concurrency and synchronization mechanisms underlying the software product.

Overview

•Consists of the ***processes*** and ***threads*** that form the system's ***concurrency*** and ***synchronization*** mechanisms, as well as their ***interactions***

•**Addresses issues** such as:

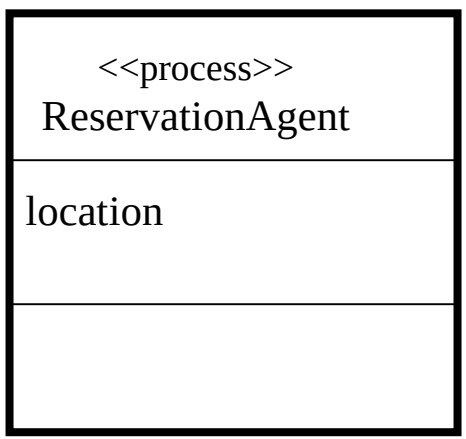
-Concurrency and parallelism (e.g., synchronization, deadlocks etc.)
-System startup and shutdown
-Performance, scalability, and throughput of the system.

•Is captured using ***class, interaction and state transition diagrams*** with a ***focus on active classes and objects***.

Processes and Threads

- ***Process*: a heavyweight flow of control that can execute independently and concurrently with other processes.**
- ***Thread*: a lightweight flow that can execute independently and concurrently with other threads within the same process.**

- Independent flows of control such as ***threads*** and ***processes*** are modeled as *active objects*. An active object is an instance of an *active class*. You may specify a process using the stereotype *process* and a thread using the stereotype *thread*.

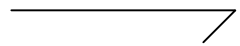


An ***Active Object*** is an object that owns a process or thread and can initiate control activity.

-Graphically an Active Class is represented as a class with thick lines.

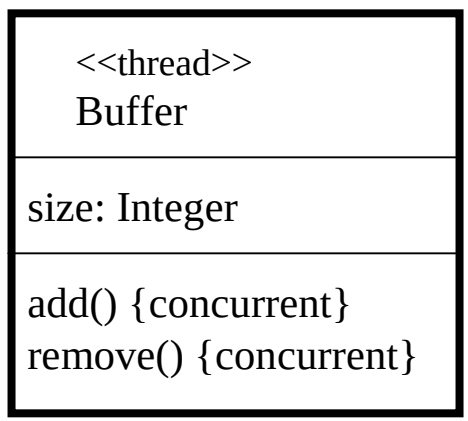
Plain classes are called ***passive*** because they cannot independently initiate control.

Communication

- You model interprocess communication using interaction diagrams:
  - ***Synchronous communication***
  - ***Asynchronous communication*** 
- Two approaches: *RPC* (synchronous) and *message passing* (asynchronous)

Synchronization

- Modeled by adding constraints to the operations; there are three kinds of synchronization:
  - Sequential
  - Guarded
  - Concurrent



sequential

÷Callers must coordinate outside the object so that only one flow is in the object at a time. If simultaneous calls occur, then the semantics and integrity of the system cannot be guaranteed.

guarded

÷The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by sequentializing all calls to all objects' guarded operations. In effect, exactly one operation at a time can be invoked on the object, reducing this to sequential semantics.

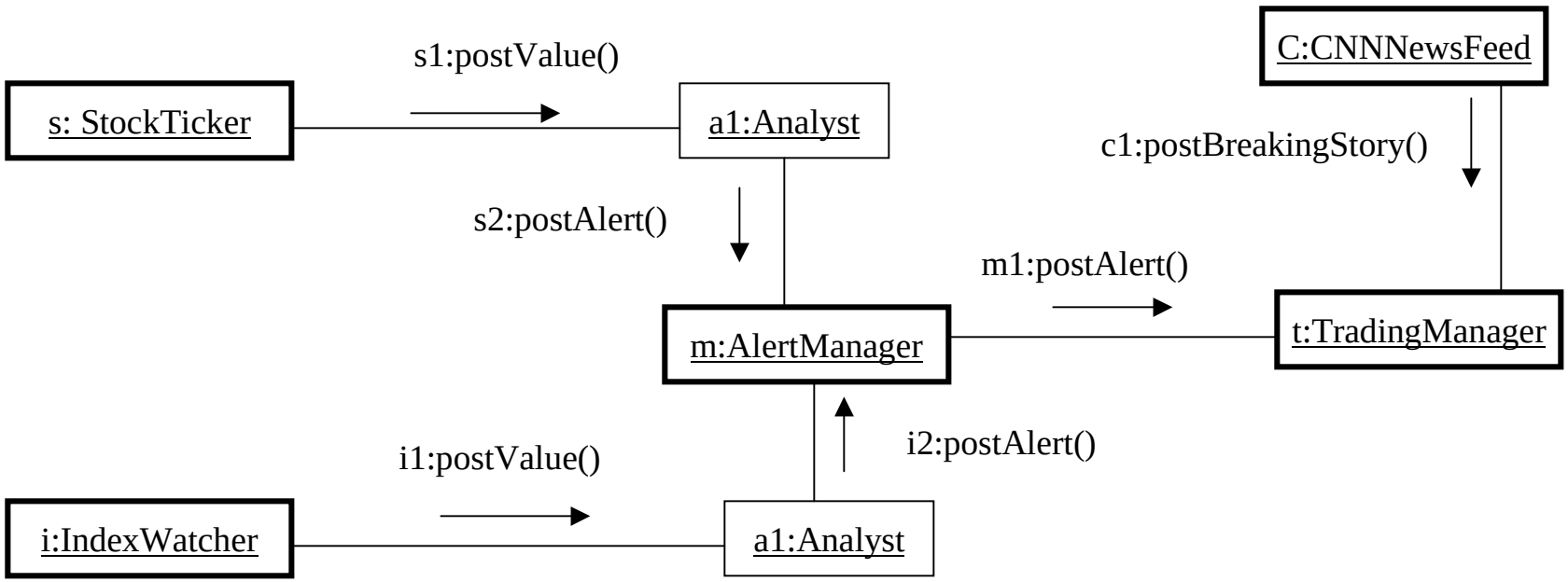
concurrent

÷Multiple calls from concurrent threads may occur simultaneously to one ***Instance*** (on any concurrent ***Operations***). All of them may proceed concurrently with correct semantics. The semantic and integrity of the object is guaranteed in the presence of multiple flows of control by treating the operation as atomic.

Note: ***Java use the Synchronized modifier, which maps to UML Concurrent property.***

Example

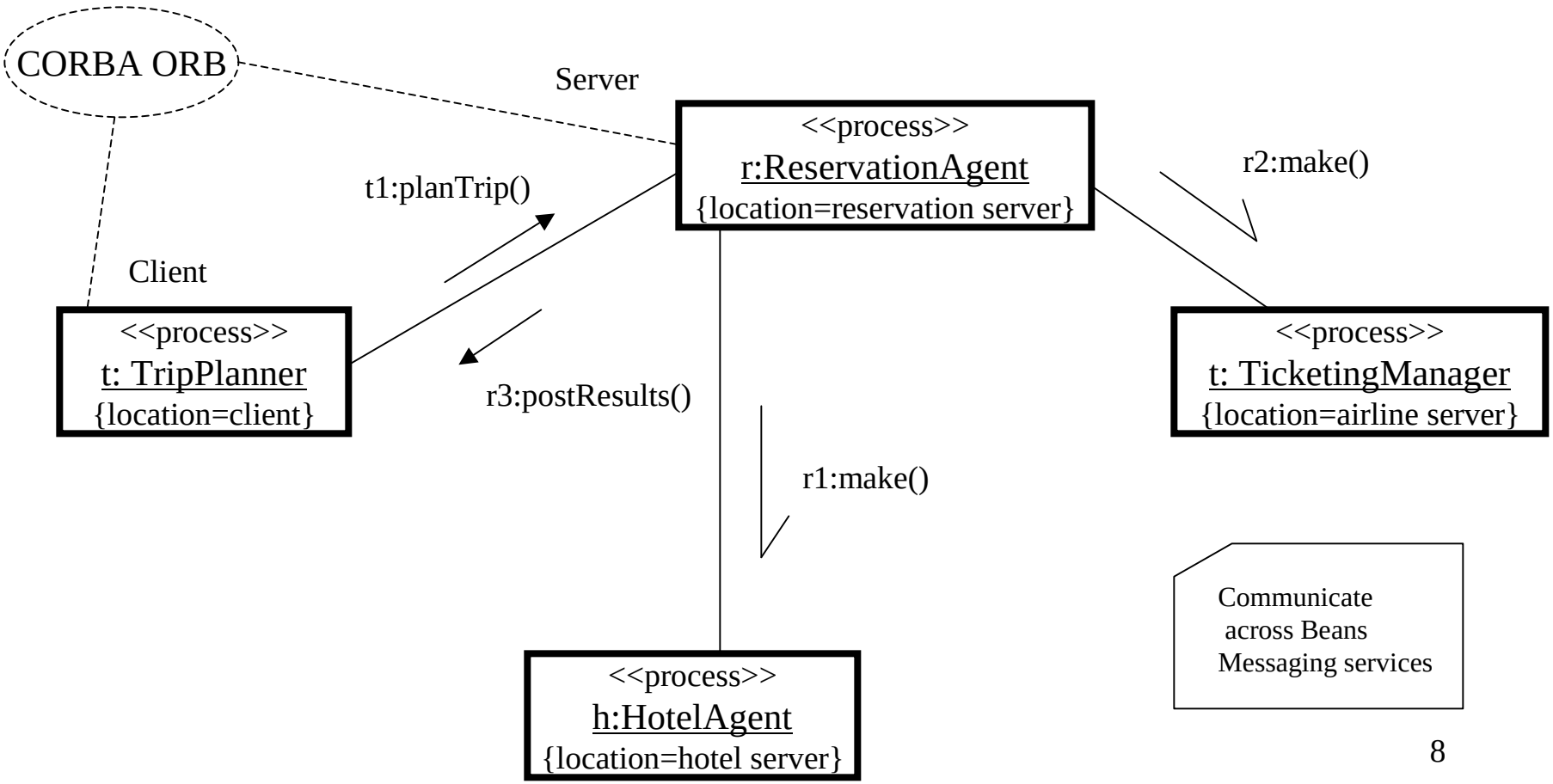
Consider a trading system, where trading decisions are based on information collected from three different sources: a stock ticker, an index watcher, and a CNNNewsFeed. Information from the stock ticker and the index watcher are first analyzed and then forwarded to the trading manager via an alert manager. The CNNNewsFeed communicates directly with the Trading manager.



- Interaction diagrams such as these are useful in helping you to visualize where two flows of control might cross paths, and therefore where special attention must be paid to communication and synchronization problems.

Example

Consider a trip planning service (e.g., expedia) that is used by travelers to identify and book all at once the best deal in terms of flight, hotel, car rental etc. Model a basic scenario where a customer uses the system to book flight and hotel room by highlighting the concurrency and synchronization involved.



3. Implementation View

Concentrates on taking the Logical view and dividing the logical entities into actual software components.

Overview

-Describes the ***organization of static software modules*** (source code, data files, executables, documentation etc.) in the development environment in terms of:

- ***Packaging and layering***
- ***Configuration management*** (ownership, release strategy etc.)

-Are modeled using *UML Component Diagrams*.

- UML components are physical and replaceable parts of a system that conform to and provide the realization of a set of interfaces

Three kinds of components:

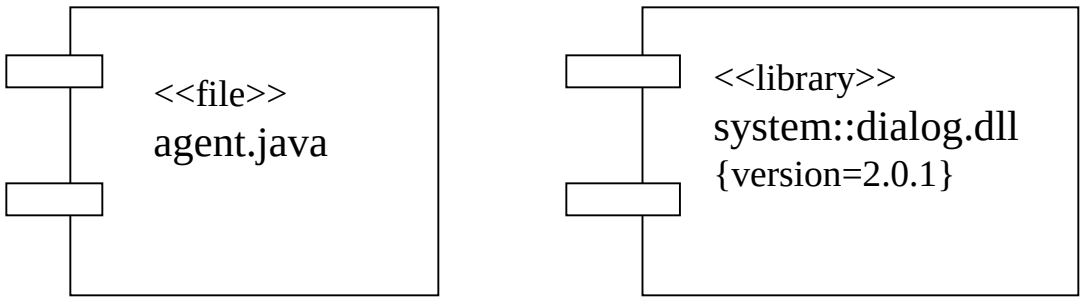
-***Deployment components***: components necessary and sufficient to form an executable system, such as DLLs, executables etc.

-***Work product components***: residue of development process such as source code files, data files etc.

-***Execution components***: created as a consequence of executing system such as COM+ which is instantiated from a DLL.

UML Components

Notation

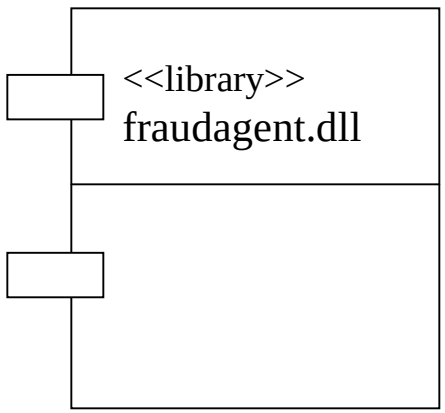


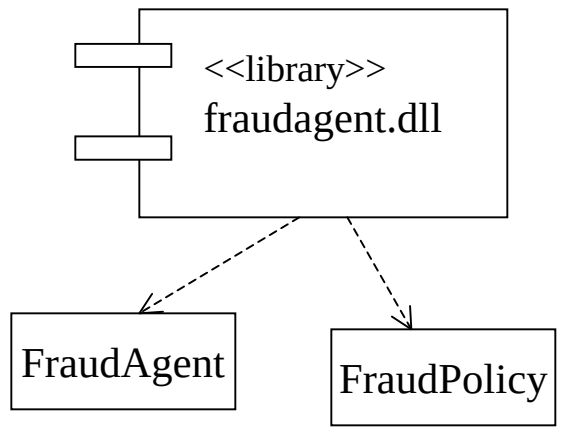
Standard Component Stereotypes

- ***executable***: a component that may be executed on a node
- ***library***: a static or dynamic object library
- ***table***: a component that represents a database table
- ***file***: a component that represents a document source code or data
- ***document***: a component that represents a document

Components and Classes

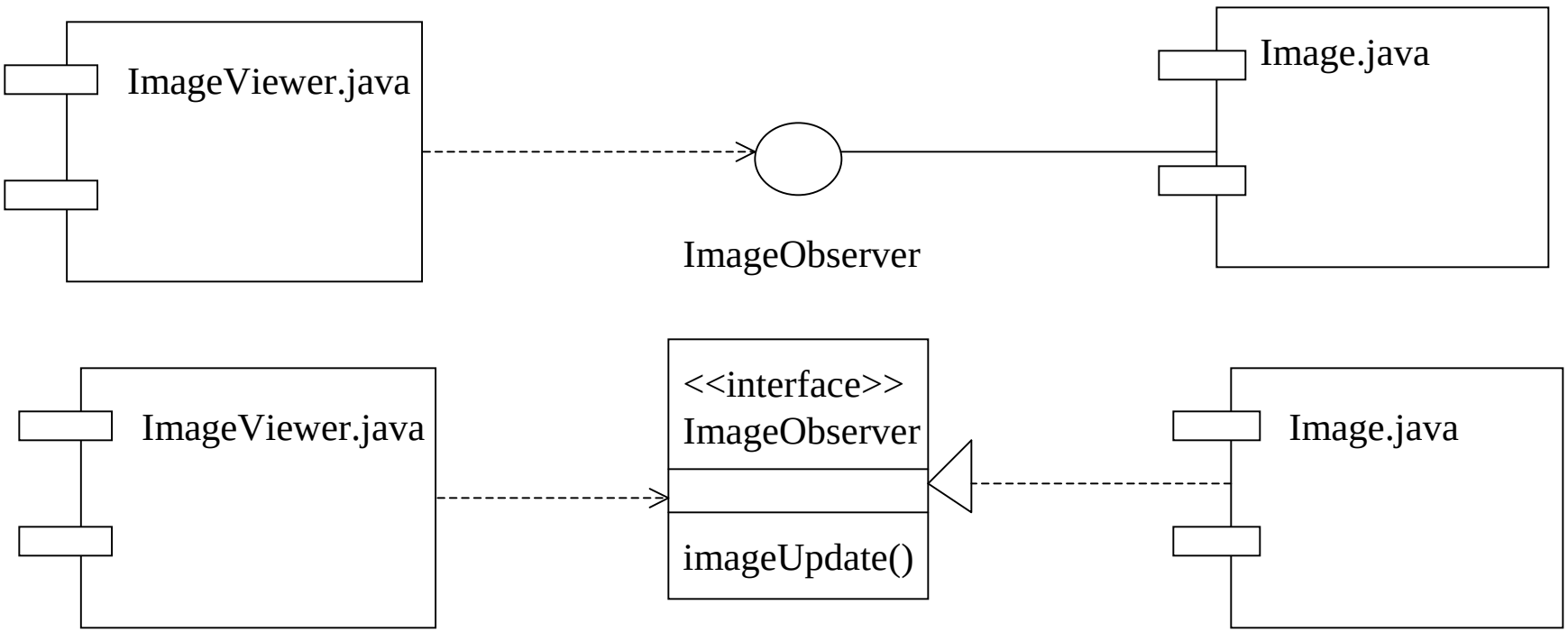
- There are significant differences between components and classes:
  - ***classes represent logical abstractions***
  - ***components represent physical entities that live on nodes***
- A component is a physical element that provides the implementation of logical elements such as classes (that is shown using a dependency relationship)





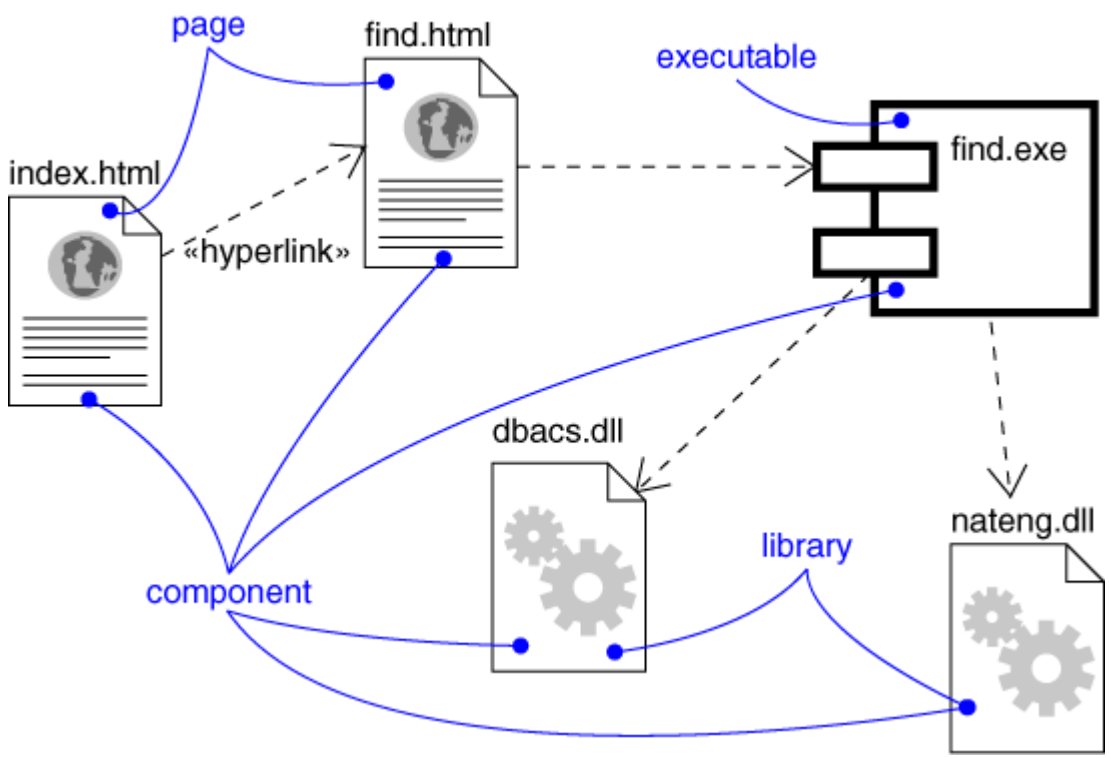
Component Interfaces

- An *interface* is a ***collection of operations that are used to specify a service of a class or a component***.
- *Interfaces* provide the glue that binds components together
- A component may provide the implementation of an interface (realization) or may access its services (dependency).

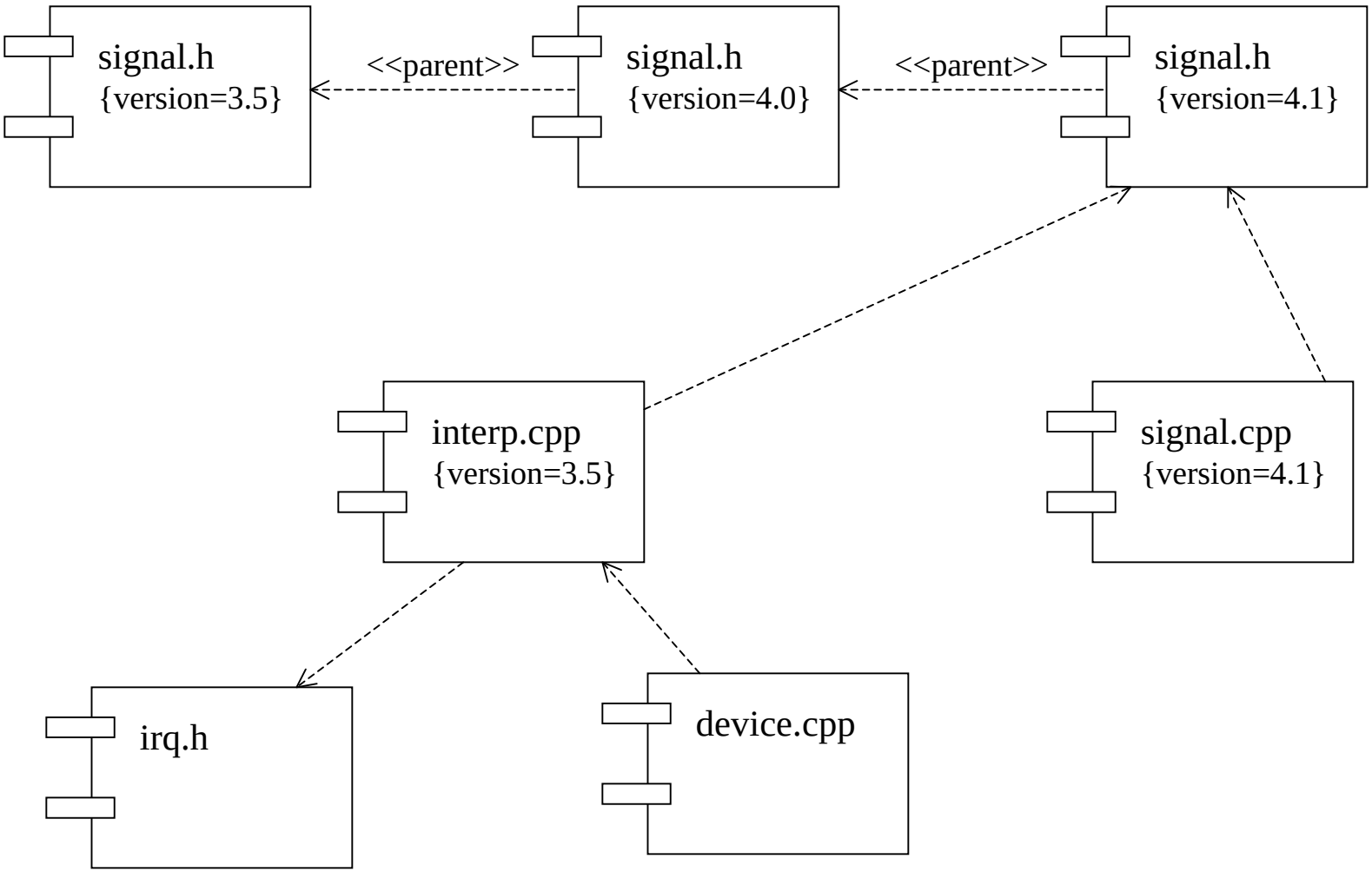


Examples

- ***Executable Release*** (for a web-based application)



- ***Source Code*** (showing different versions of the same program)



- Based on this component diagram, it is easy to trace the impact of changes. For example, changing the source code file signal.h will require recompilation of three other files: signal.cpp, interp.cpp, and transitively device.cpp. However, file irq.h is not affected.

4. Deployment View

Concentrates on how the software is deployed into that somewhat important layer we call 'hardware'.

Overview

-Shows how the ***various executables and other runtime entities*** are ***mapped to the underlying platforms or computing nodes.***

-Addresses issues such as:

- ***Deployment***
- ***Installation***
- ***Maintenance***

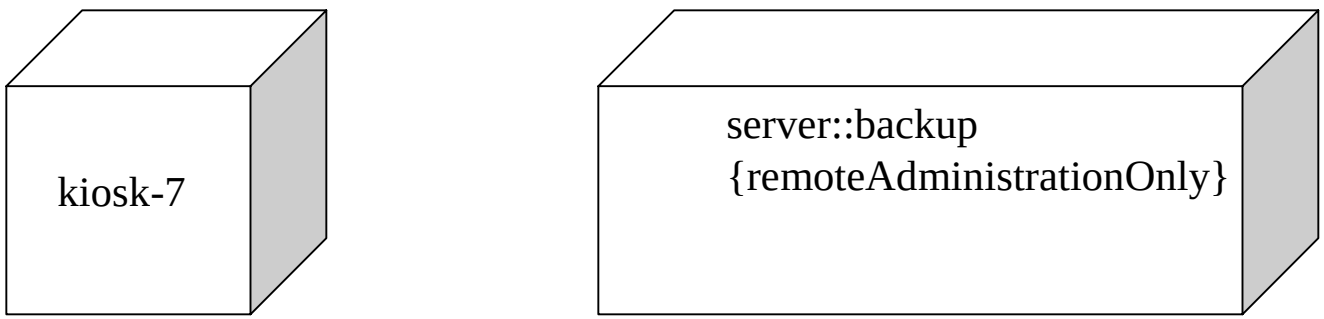
Exposes:
- System performance
- Object/data distribution
- Quality of Service (QoS)
- Maintenance frequency and effects on uptime
- Computing nodes within the system

Deployment Diagram

Notation

- A ***node*** is a ***physical element representing a computational resource***, generally having some memory and processing capability.

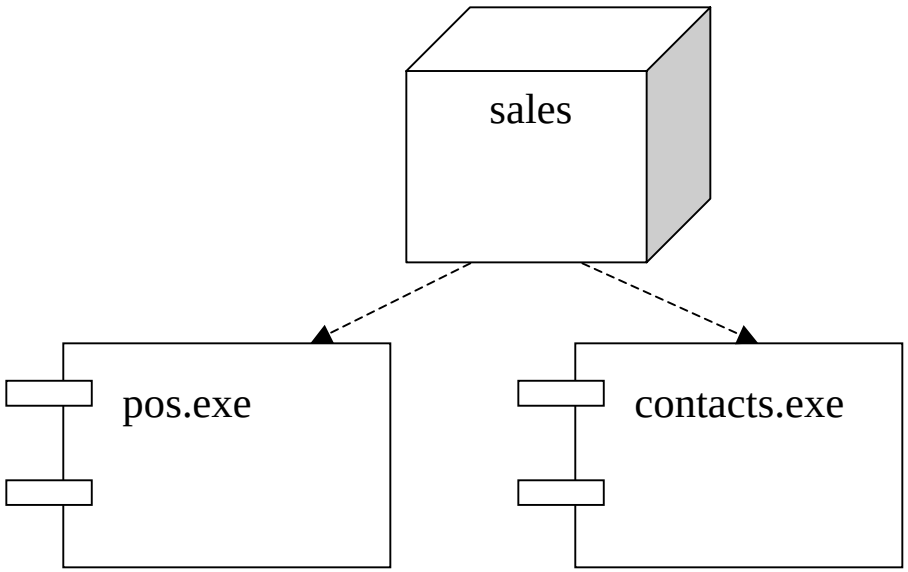
- Nodes are used to model the topology of the hardware on which the system executes: processor or device on which components may be deployed.

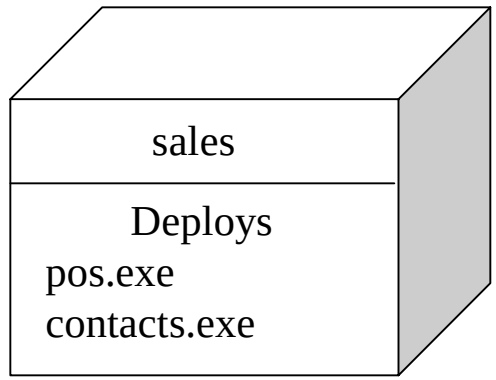


- You may ***organize nodes by specifying relationships*** among them.

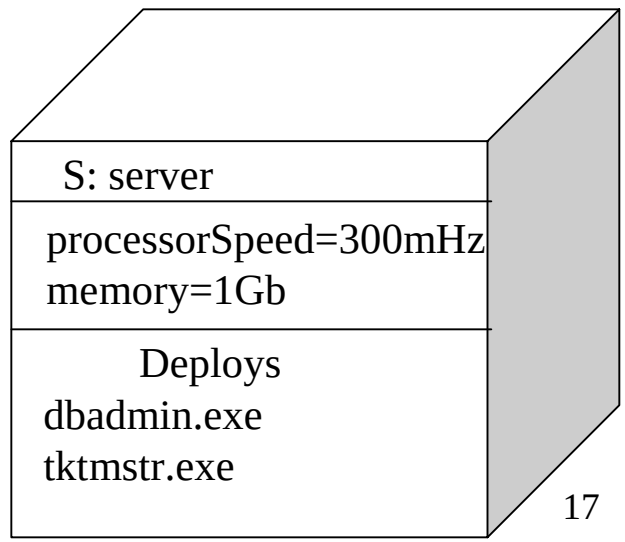
Nodes and Components

- Nodes are locations upon which components are deployed.
- A set of objects or components that are allocated to a node as a group is called a ***distribution unit***.





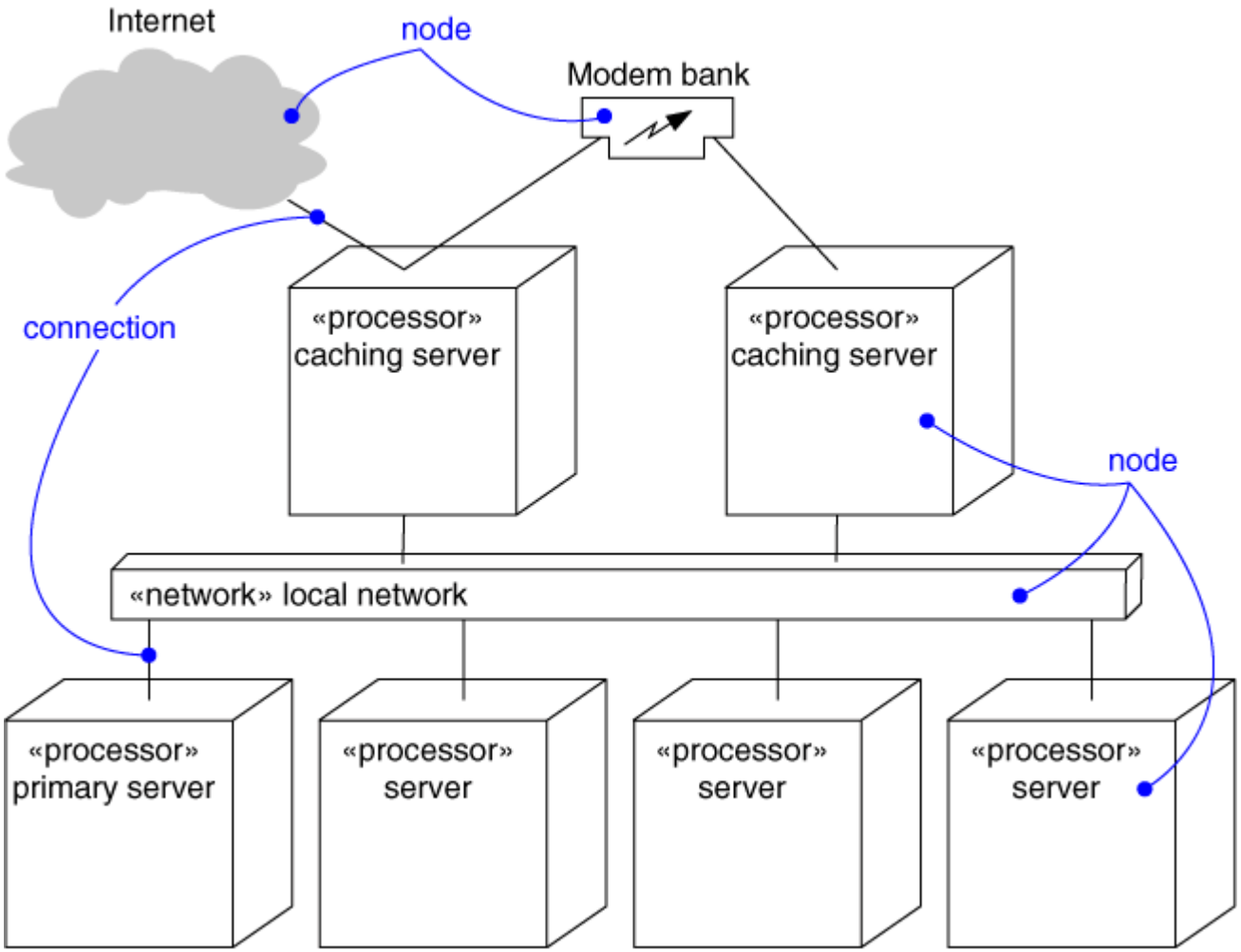
- You may also ***specify attributes and operations*** for them: *speed*, *memory*



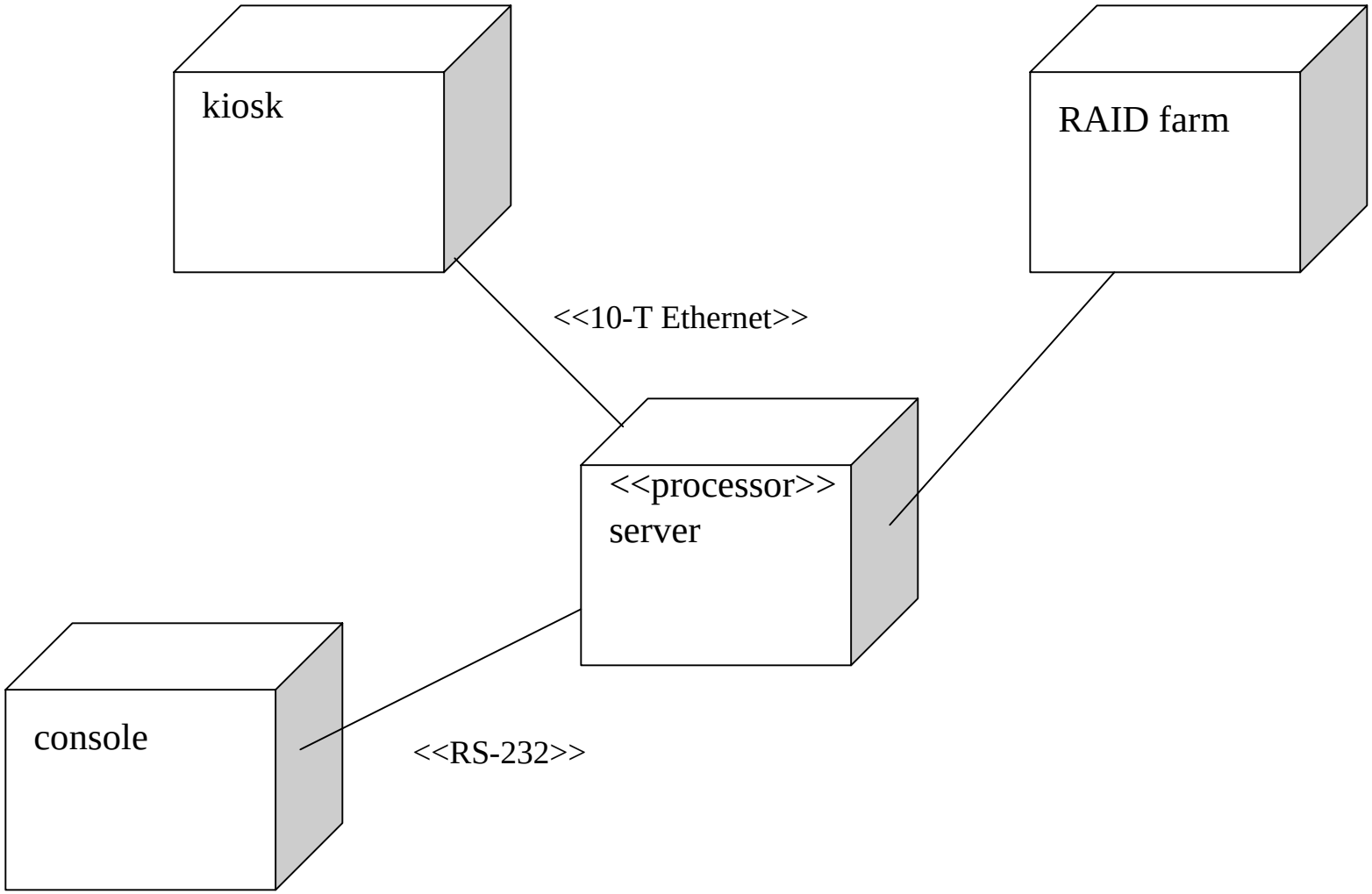
Deployment Diagram

•You use a deployment diagram to model the static deployment view of a system.

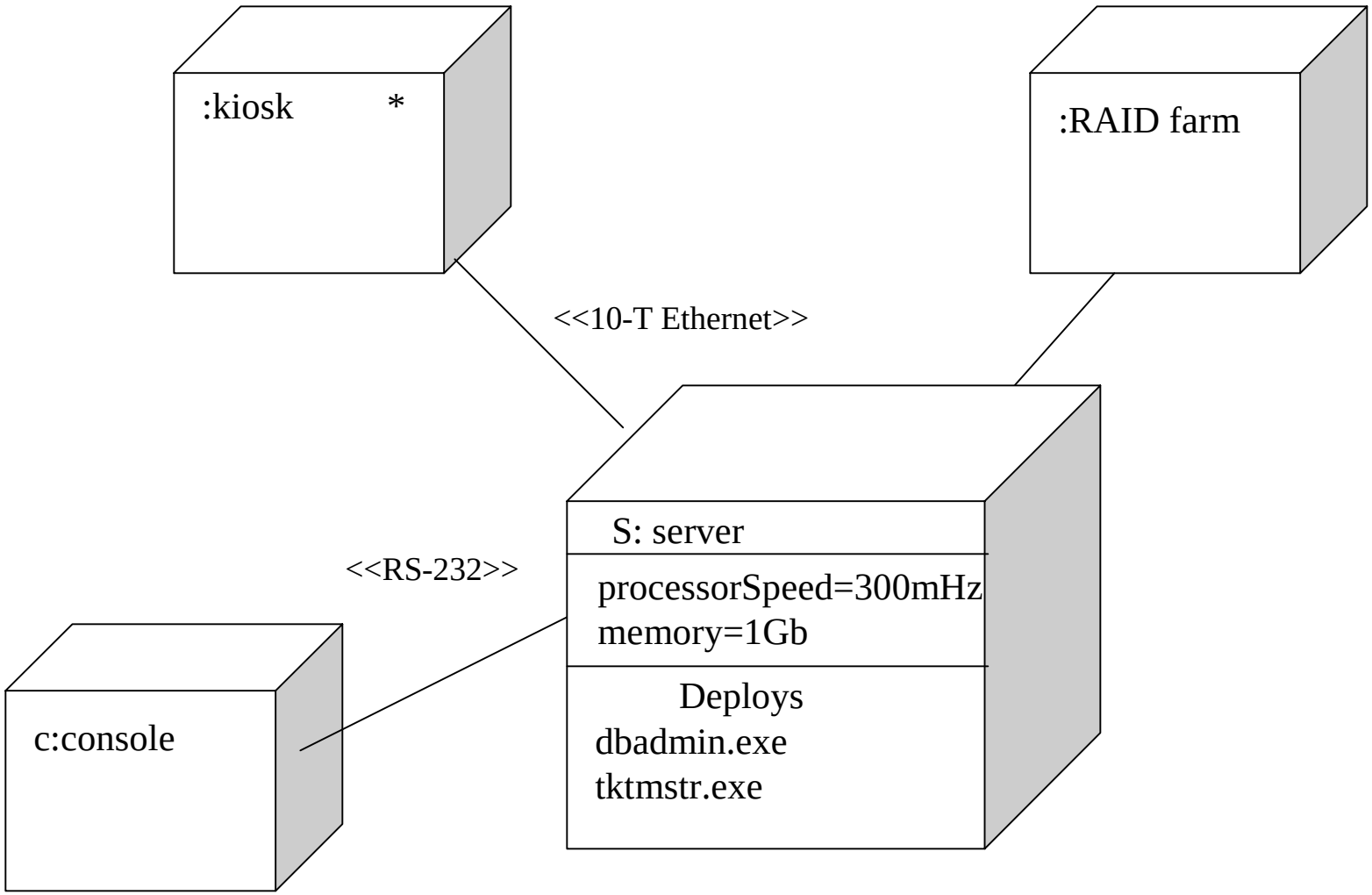
- *Example 1*



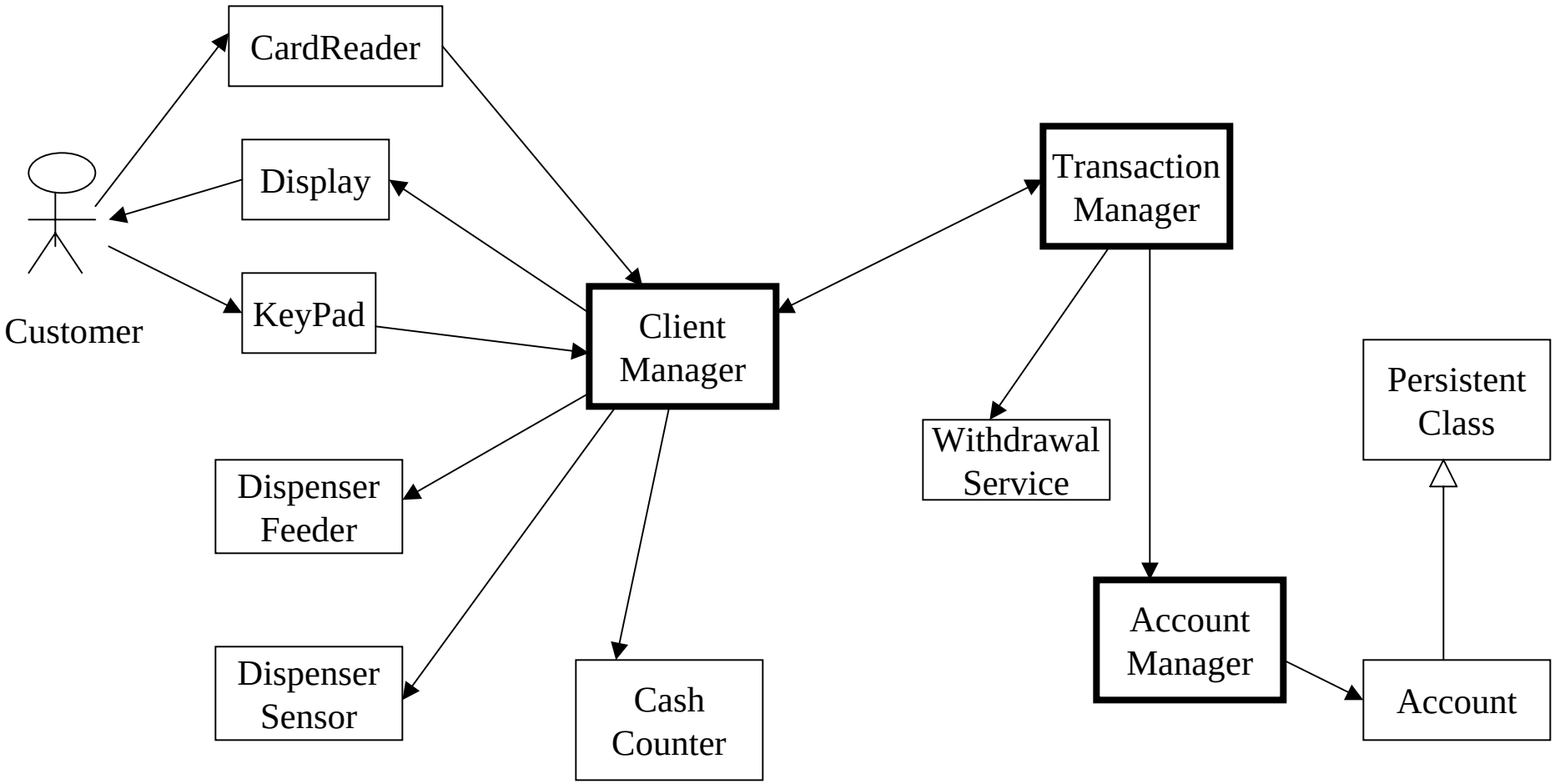
- *Example 2*


kiosk
RAID farm
<<10-T Ethernet>>
<<processor>>
server
console
<<RS-232>>

- *Distribution of Components*

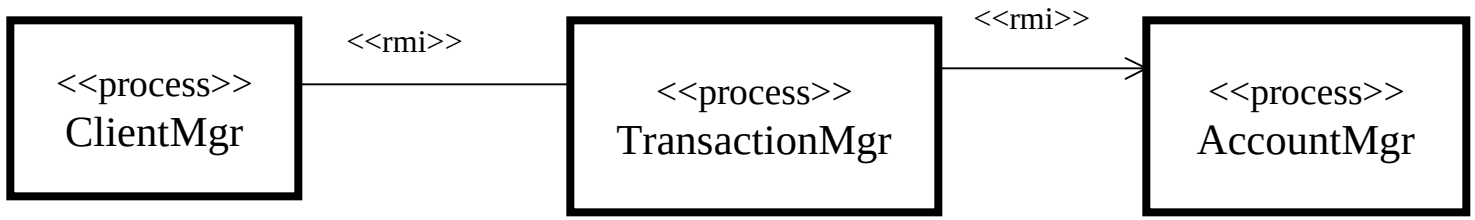


5. ATM Example



Process View

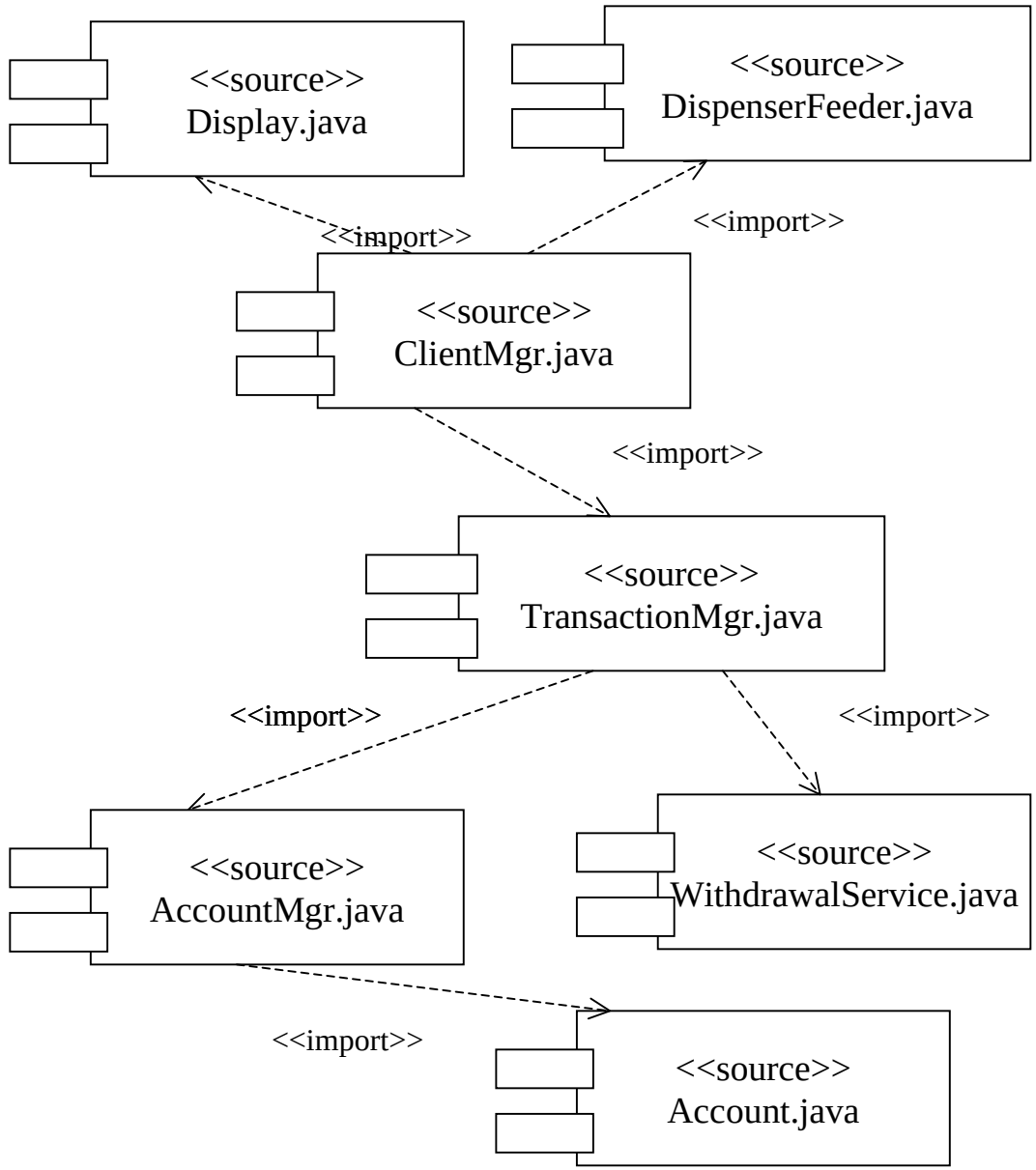
Class diagram



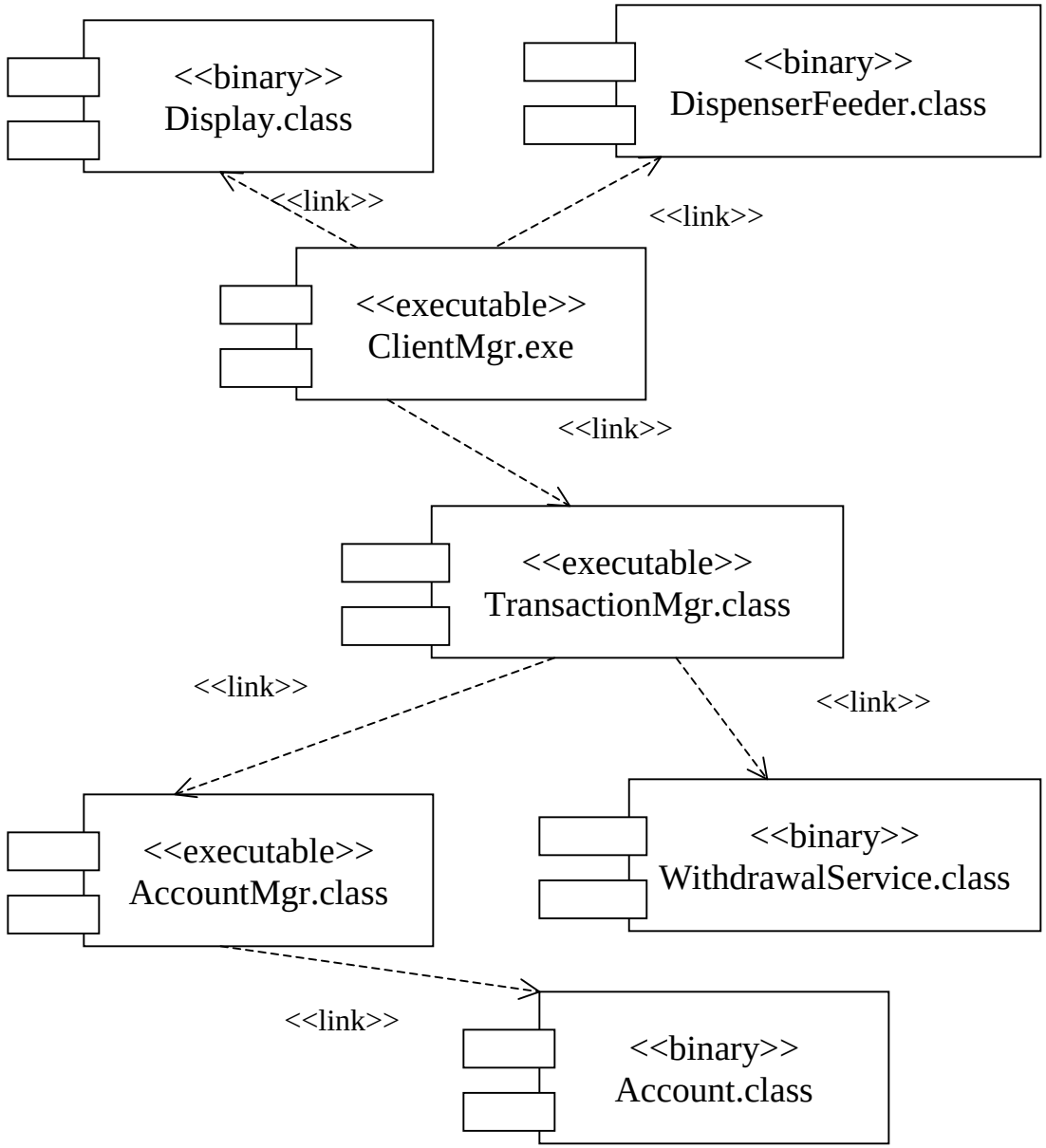
Classes	**Processes**
Display, KeyPad, CardReader, ClientMgr, DispenserFeeder, DispenserSensor, CashCounter	ClientMgr
Transaction Mgr, WithdrawalService	TransactionMgr
AccountMgr, Account, Persistent class	AccountMgr

Implementation View
-Source Components

Classes	Source Components
CardReader, Display, KeyPad	Display.java
DispenserFeeder, DispenserSensor, CashCounter	DispenserFeeder.java
ClientMgr	ClientMgr.java
TransactionMgr	TransactionMgr.java
AccountMgr	AccountMgr.java
WithdrawalServic e	WithdrawalService.java
Account, Persistent class	Account.java

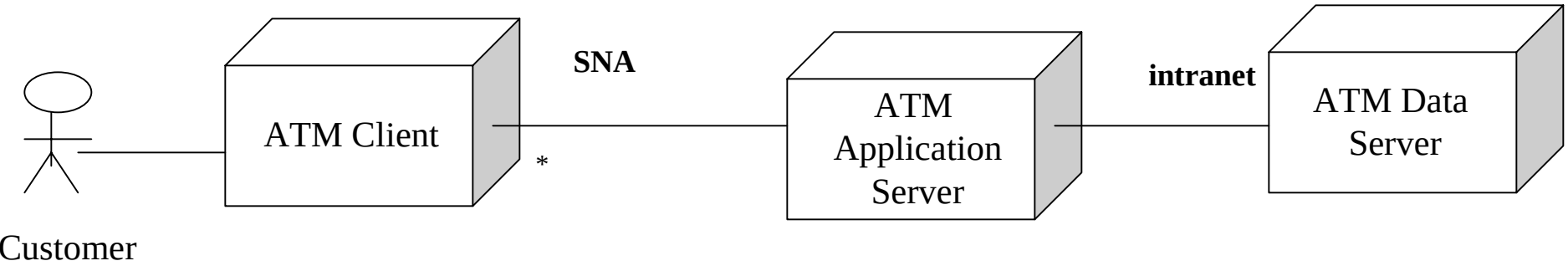


-Executable Release



Deployment View

- Deployment Diagram



- Deployment of Active Objects