Designing the Architecture

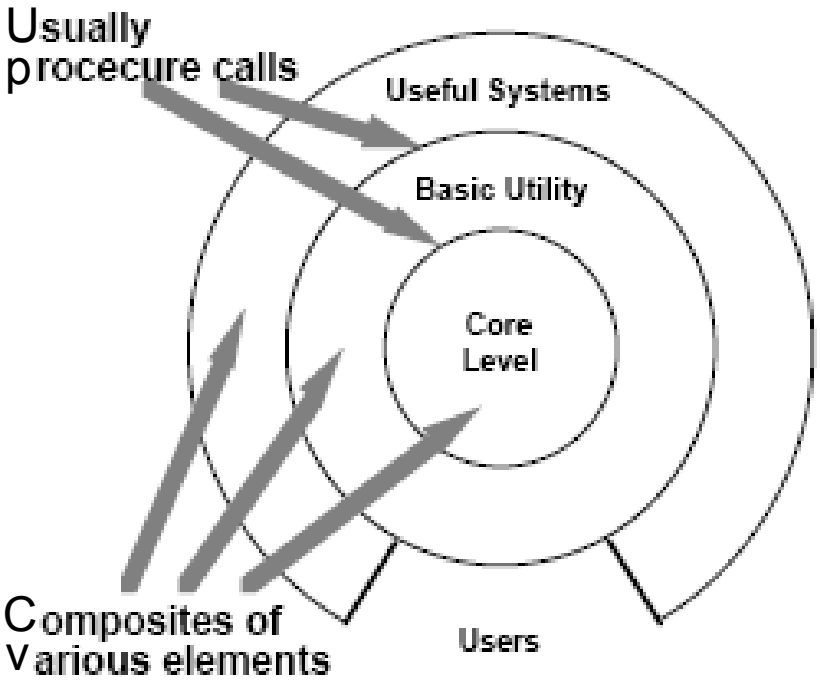
Some of the material in these slides is adapted from *Software Architecture in Practice, 2nd edition* by Bass, Clements and Kazman.
Some of the material in these slides is adapted from "An Introduction to Software Architecture" by Garlan and Shaw.
Some of the material in these slides is adapted from "Pattern-oriented software Architecture" by Buschmann et al.

What is an Architectural Style?

Architectural patterns

- Distributed
- Event-driven
- Frame-based
- Batch
- Pipes and filters
- Repository-centric
- Blackboard
- Rule-based
- Layered
- MVC
- IR-centric
- Subsumption
- Bridge
- Microkernel
- Reflection

Layered

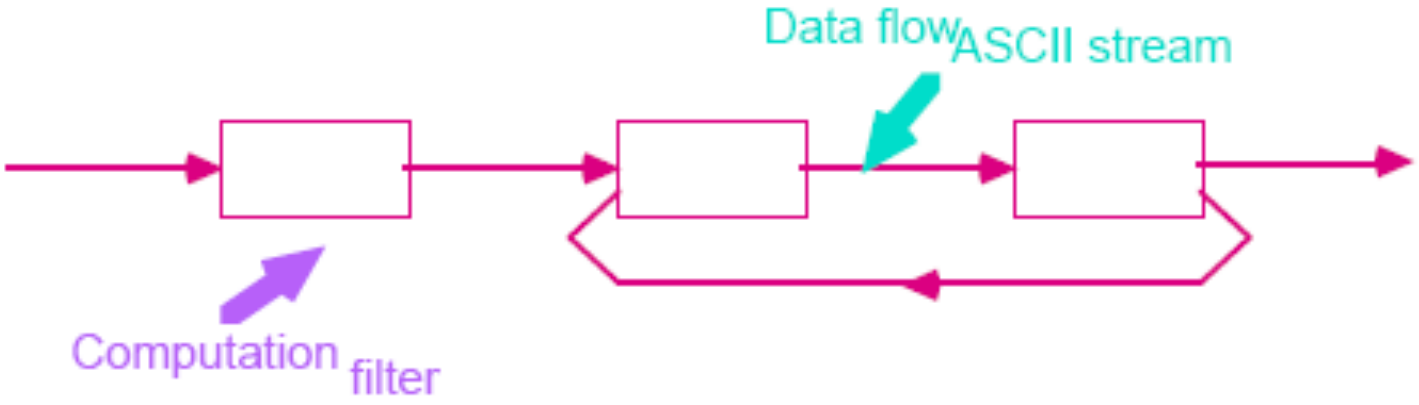


- Helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction

Layered Pattern - Implementation

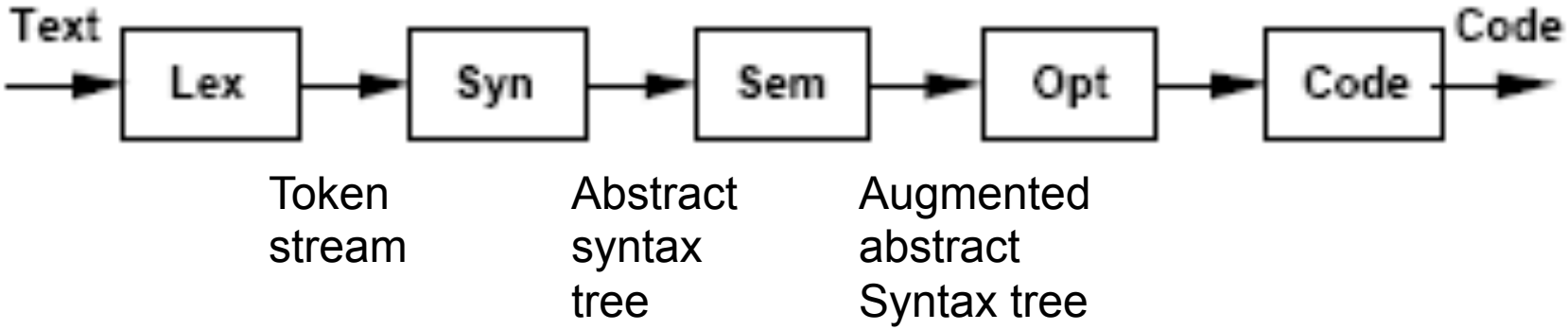
- Define the abstraction criterion for grouping tasks into layers
- Determine the number of abstraction levels
- Name the layers and assign tasks to each of them
- Specify the services
- Refine the layering
- Specify an interface for each layer
- Structure individual layers
- Specify the communication between adjacent layers
- Decouple adjacent layers
- Design an error-handling strategy

Pipes and Filters

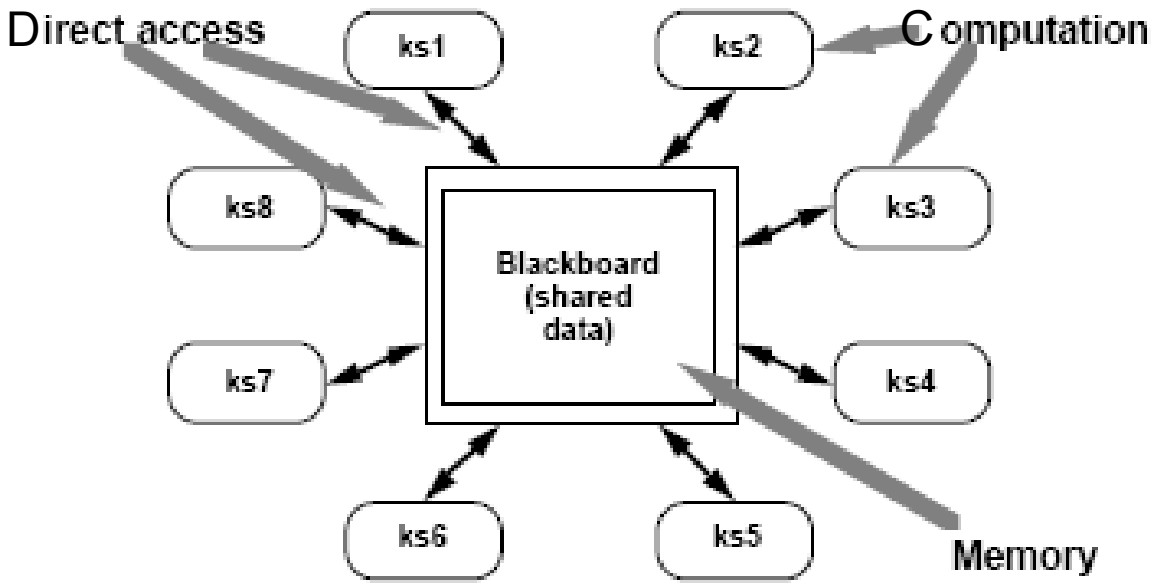


- Each processing step is encapsulated in a filter component
  - Enrich
  - Refine
  - Transform
- Data is passed through pipes between adjacent filters

Compiler – Pipes and Filters example

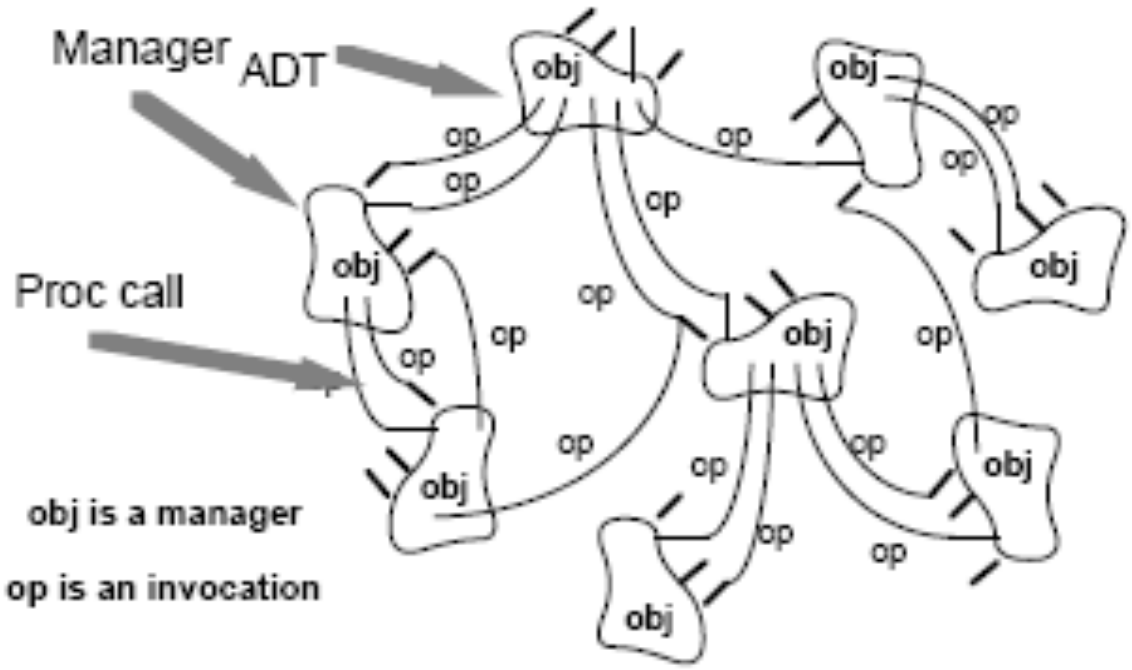


Blackboard



- Useful for problems for which no deterministic solution strategies are known.
- Several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution

Data Abstraction (Object Oriented)



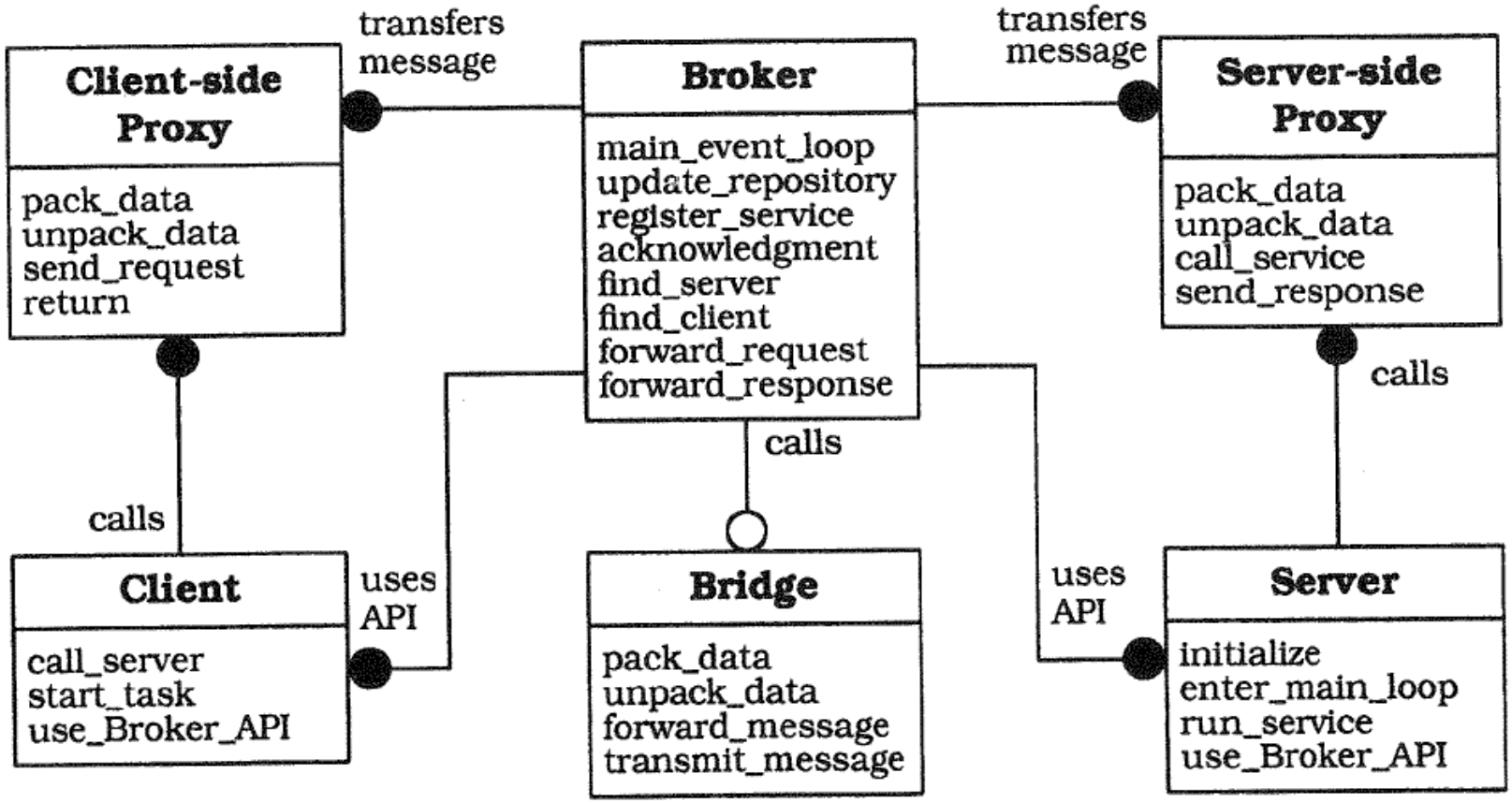
Distributed Processes

- Often described in terms of their topology:
  - ring
  - star
- Sometimes described in terms of communication protocols:
  - client/server

Distributed Systems – Broker Pattern

- Structure distributed software systems with decoupled components that interact by remote service invocation
- A broker component is responsible for coordinating communication
- Abstraction of the common architecture shared by Microsoft OLE (Object Linking and Embedding) and OMG's CORBA (Common Object Request Broker Architecture)

Broker Pattern



Domain-Specific

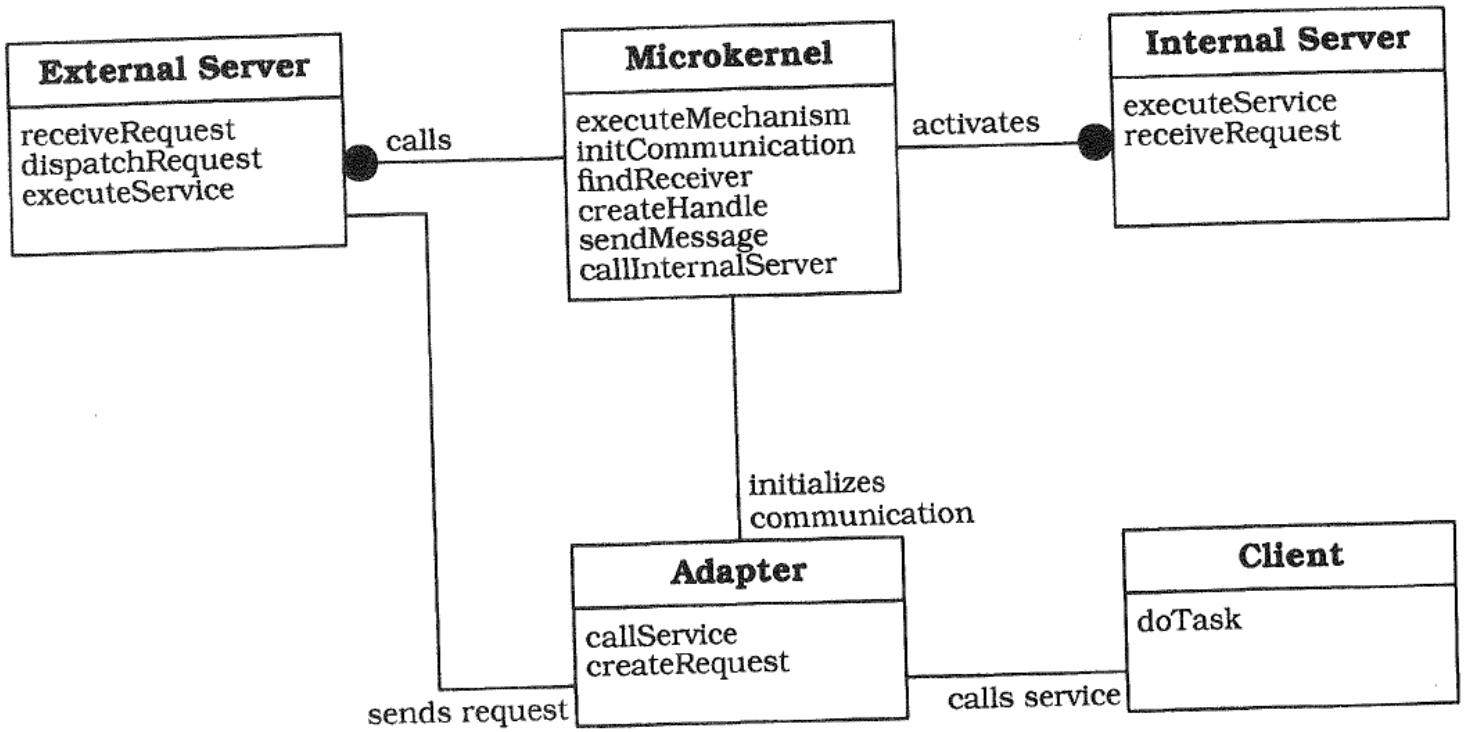
- Specific to a family of systems -- a product line
- Some components are common, but may be implemented in different ways
- Connections between common components are preserved across family

State Transition Systems

- Set of states
- Set of transitions between states
- Good style for reactive systems

Adaptable Systems - Microkernel

- Applies to software systems that must be able to adapt to changing system requirements. Separates a minimal functional core from extended functionality and customer-specific parts



Attribute-Driven Design

- Recursive decomposition
- Start with:
  - Functional requirements (use cases)
  - Constraints
  - Quality attributes (scenarios)

Attribute-Driven Design Process

1. Choose module to decompose
2. Refine module:
   a) Choose architectural drivers
   b) Choose architectural pattern
   c) Instantiate modules, allocate functionality, and represent using multiple views
   d) Define interfaces
   e) Refine use cases and scenarios---make them constraints for sub-modules
3. Repeat until done

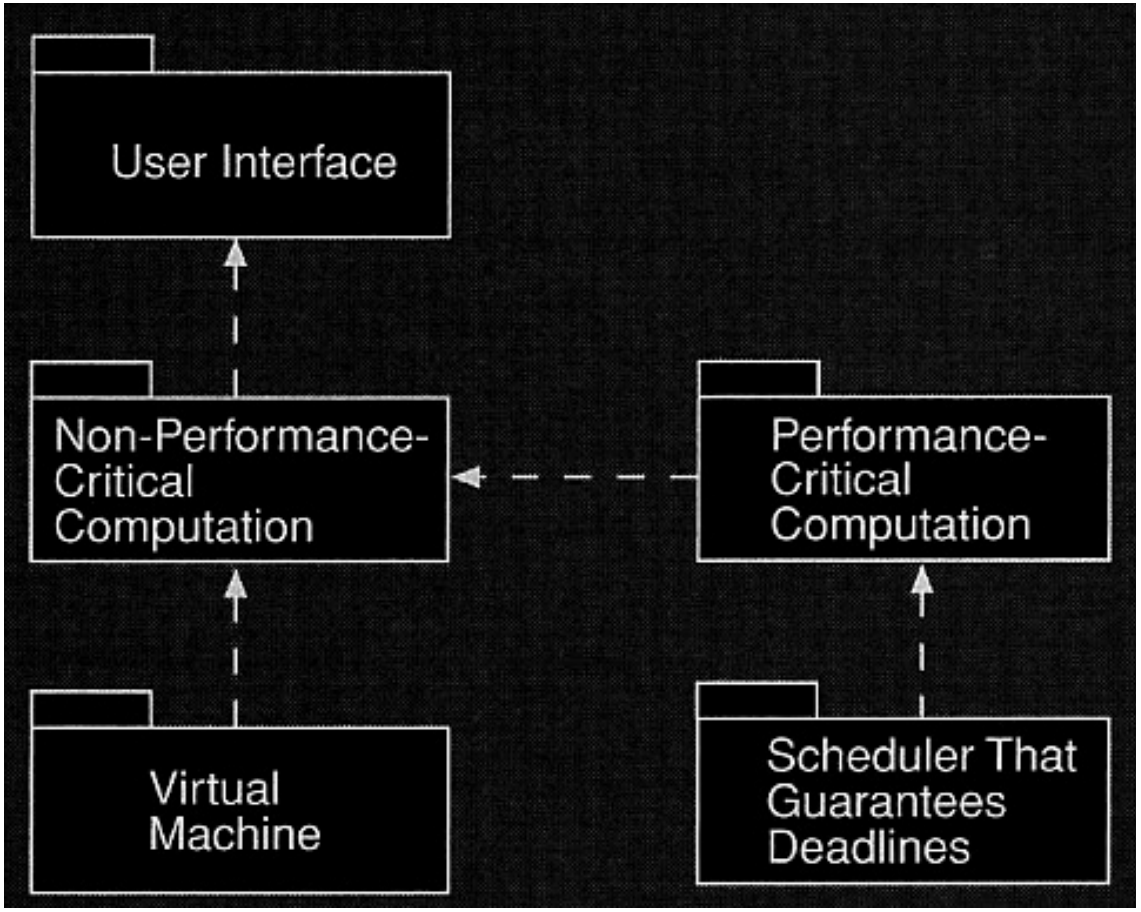
2.a) Choose Architectural Drivers

- Drivers are combination of functional requirements, constraints and quality attributes
- Prioritize requirements and select those that will "shape" the architecture
- May need some investigation to determine drivers

2.b) Choose Architectural Pattern

- Use tactics to achieve quality attributes
- Patterns package tactics
- Document your choice of tactics in your Design Notebook!

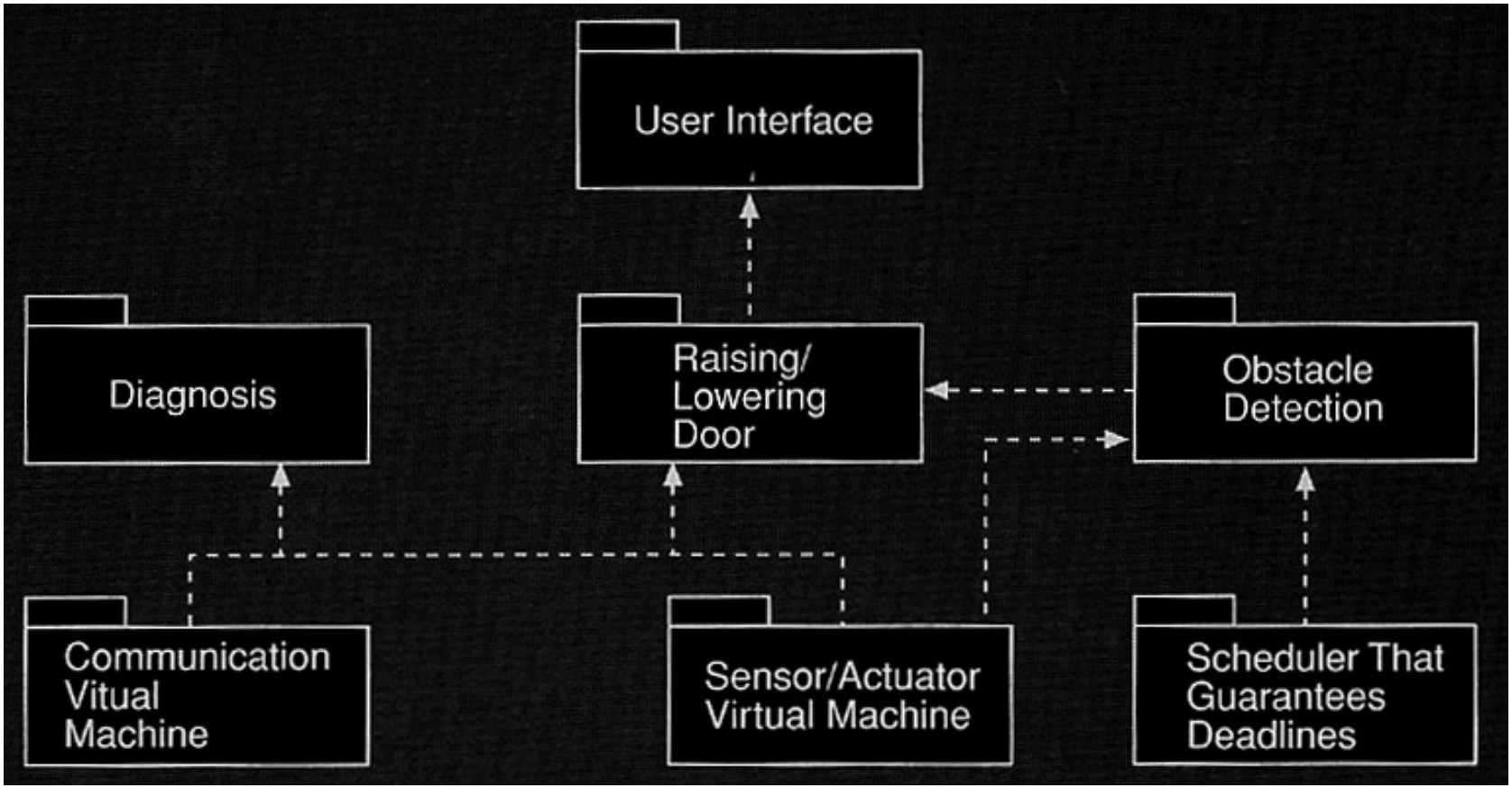
Pattern for Garage Door Opener



2.c) Instantiate Modules

- Refine (or interpret) the pattern for your particular problem
- Result is a decomposition into sub-modules

First-Level Decomposition



2.c) Allocate Functionality

- Use the use cases to identify flow of information
- Assign responsibilities to sub-modules
- Pattern may help this process

2.c) Represent Using Multiple Views

- Pick one view from each major category:
  - Module
    - Decomposition, Uses, Class
  - Component and Connector
    - Client-server, Concurrency, Process, etc.
  - Allocation
    - Work assignment, Deployment, Implementation

2.d) Define Interfaces

- Each view provides information about interfaces
- Need to identify services provided and used

2.e) Refine Use Cases and Quality Scenarios

- Associate use cases and quality attributes to sub-modules
  - may need to break up use cases
- Quality scenarios:
  - may be satisfied by decomposition
  - may impose constraints on sub-modules
  - may be neutral with respect to decomposition
  - may not be satisfiable with decomposition